MODULE 1

Chapter 1

Digital & Analog System

1.0 INTRODUCTION

TYPES OF ELECTRONICS SYSTEM

a) **DIGITAL SYSTEM**
b) **ANALOG SYSTEM**

Digital System: It is an electronic system which consists of digital circuits and/or devices.

Analog System: It is an electronic system which consists of analog circuits and/or devices.

Following are the various terms related to digital system:-

i) **Digital electronics** is a branch of electronics which deals with the digital devices, circuits and system. Digital electronics are representations of Boolean algebra and are used in computers, mobile phones, medical instrumentation, industrial process control and other consumer products. In a digital circuit, a signal is represented in one of two states or logic levels.

ii) **Digital Signal** is a signal with only two discrete (step by step) values.

These are usually represented by LOW and HIGH or 0 and 1.

iii) **Logic Gate** is a logic circuit in which the output depends upon the inputs according

to some logic rules.

iii) **Digital circuit** are usually made from large assemblies of logic gates for electronic

representations of Boolean logic functions which processes digital signals.

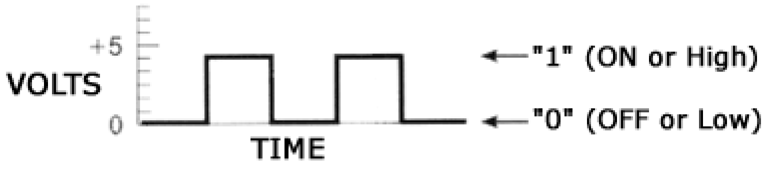
In other words Logic Gates are digital circuits. All digital circuits are either ON or OFF.

A light switch in your house can be used as an example of a digital circuit. The light is either ON or OFF depending on the switch position.

When the Light is ON the output value is '1'.

When the Light is OFF the output value is '0'.

The inputs are the position of the light switch. The switch is placed either in the ON or OFF position to activate the Light.



Following are the various terms related to analog system:-

i) **Analog Signal** is a continuous signal that can have any value in a given range.

ii) **Analog Circuit** is an electronics circuit that processes analog signals.

1.1 Advantages of Digital System

Less affected by noise

One advantage of digital system when compared to analog system is that signals represented digitally can be transmitted without degrading because of noise. For example, a continuous audio signal, transmitted as a sequence of 1s and 0s, can be reconstructed without error provided the noise picked up in transmission is not enough to prevent identification of the 1s and 0s. An hour of music can be stored on a compact disc as about 6 billion binary digits.

Accuracy and precision are greater

In a digital system, a more precise representation of a signal can be obtained by using more binary digits to represent it. While this requires more digital circuits to process the signals, each digit is handled by the same kind of hardware. In an analog system, additional resolution requires fundamental improvements in the linearity and noise characteristics of each step of the signal chain.

Information storage is easy

Information storage can be easier in digital systems than in analog ones. The noise-immunity of digital systems permits data to be stored and retrieved without degradation. In an analog system, noise from aging and wear degrade the information stored. In a digital system, as long as the total noise is below a certain level, the information can be recovered perfectly.

Easier to design

Digital System can be easier to design as only the range (HIGH or LOW) in which they fall are important where as in analog system exact values of voltage or current are important.

More digital circuitry can be fabricated on IC chips.

1.2 Disadvantages

Digital System may need more energy

In some cases, digital circuits use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well. In portable or battery-powered systems this can limit use of digital systems. For example, battery-powered cellular telephones often use a low-power analog front-end to amplify and tune in the radio signals from the base station. However, a base station has grid power and can use power-hungry, but very flexible software radios. Such base stations can be easily reprogrammed to process the signals used in new cellular standards. Digital circuits are sometimes more expensive, especially in small quantities.

Quantization error may occur

Most physical quantities in real world are analog in nature (like light, temperature, sound, electrical conductivity, electric and magnetic fields are analog) and these quantities are often the inputs and outputs that are being monitored, operated on, and controlled by a system. Thus conversion to digital format and re-conversion to analog format is needed. This may causes quantization errors

Quantization error can be reduced but single-bit error may occur

In some systems, if a single piece of digital data is lost or misinterpreted, the meaning of large blocks of related data can completely change. It can be difficult for users to tell if a particular system is right on the edge of failure, or if it can tolerate much more noise before failing. Digital memory and transmission systems can use techniques such as error detection and correction to use additional data to correct any errors in transmission and storage.

On the other hand, some techniques used in digital systems make those systems more vulnerable to single-bit errors. These techniques are acceptable when the underlying bits are reliable enough that such errors are highly unlikely. but a single-bit error in audio data stored directly as linear pulse code modulation (such as on a CD-ROM causes, at worst, a single click). Instead, many people use audio compression to save storage space and download time, even though a single-bit error may corrupt the entire song.

Multiple Choice Questions

Q1 The number of levels in a digital signal is

a) Ten
b) One
c) Four
d) Two

Q2 The slow turning of a potentiometer is

a) Digital input
b) Analog out put
c) Nature of the output depend on voltage
d) It depends on resolution of the potentiometer.

Q3 Which of the following can provide a digital signal?

a) Slow change in the value of a resistor
b) sinusoidal wave
c) Square wave
d) Gradual turning of a potentiometer.

Q4 Digital System has following advantages:-

a) Less affected by noise
b) Accuracy and precision are greater
c) None of the above
d) Both of the above

Q5 Analog circuit process:

a) Analog signal
b) Discrete signal
c) Digital signal
d) All of the above

Q6 A signal having continuous values is known as ___________

a) a digital signal
b) an analog signal
c) a natural signal
d) none of the above

Chapter 2

Number System

2.0 Introduction

The representation of numbers using any system (like decimal, binary etc.) is known as number system. In any number system there is an ordered set of symbols known as digits. A collection of these digits makes a number.

Here, a number may have two parts:-

a) Integer

b) and fractional

that is

$(N)_b = d_{n-1} d_{n-2} \ldots\ldots\ldots\ldots d_1 d_0 . d_{-1} d_{-2} \ldots\ldots\ldots\ldots d_{-f} \ldots\ldots d_{-m}$

Where N= a number

b=radix

n=number of digit in integer portion

m=number of digit in fractional portion

d_{n-1}= most significant digit

d_{-m}= least significant digit

Many number systems 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 a tool that we use every day. Examining some of its characteristics will help us to better understand the other systems.

In the next few pages we shall introduce four numerical representations.

- Decimal
- Binary
- Octal
- Hexadecimal

2.1.1 Decimal Number System

The number system can be represented with base or radix 10.The representation of number with base 10 is known as decimal number system.

Decimal number system contains 10 unique numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 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

Examples:

- $(4.34)_{10}$
- $(162)_{10}$

2.1.2 Binary Number System

The number system can be represented with base or radix 2.The representation of number with base 2 is known as binary number 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

Binary Number and their corresponding decimal numbers

2^3	2^2	2^1	2^0	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Why Binary Numbers

Modern computers don't work with decimal numbers. Instead they process binary numbers, groups of 0s and 1s.because electronics devices are most reliable when designed for two state (binary) operation.

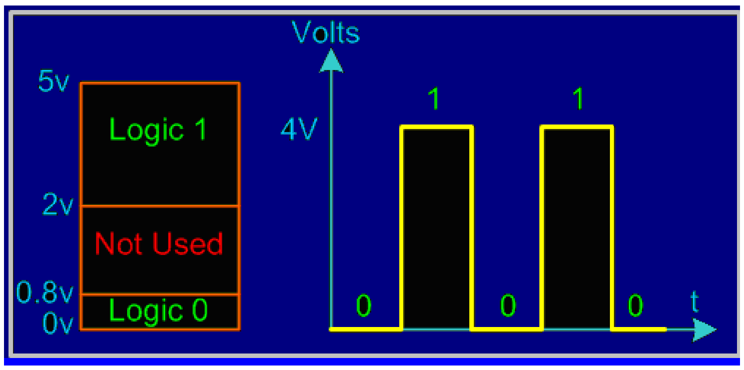
Typical Voltage Assignment

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 circuit's works at 1.8 volts, so this statement may not hold true for all logic circuits. *(See the picture)*

We can see another significant difference between digital and analog systems. In digital systems, the exact voltage value is not important; e.g., a voltage of 3.6V means the same as a voltage of 4.3V. In analog systems, the exact voltage value is important.



2.1.3 Octal Number System

The number system can be represented with base or radix 8.The representation of number with base 8 is known as octal number system.

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

8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
512	64	8	1	.	1/8	1/64	1/512
Most significant digit				Octal Point			Least Significant Digit

Octal to Decimal Conversion

- $117_8 = 1 \text{ x } (8^2) + 1\text{x } (8^1) + 7 \text{ x } (8^0) = 79_{10}$
- $24.6_8 = 2 \text{ x } (8^1) + 4 \text{ x } (8^0) + 6 \text{ x } (8^{-1}) = 20.75_{10}$

Octal Number and their corresponding decimal and binary numbers

Octal	Decimal	Binary
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
10	8	001000
11	9	001001
12	10	001010
13	11	001011
14	12	001100
15	13	001101
16	14	001110
17	15	001111

2.1.4 Hexadecimal Number System

The number system can be represented with base or radix 16.The representation of number with base 16 is known as hexadecimal number 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

Hexadecimal to Decimal Conversion

- $20.6_{16} = 2 \text{ x } (16^1) + 0\text{x } (16^0) + 6 \text{ x } (16^{-1}) = 16.375_{10}$
- $12.1_{16} = 1 \text{ x } (16^1) + 2 \text{ x } (16^0) + 1 \text{ x } (16^{-1}) = 17.0625_{10}$

Hexadecimal Number and their corresponding decimal and binary numbers

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110

7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

2.2 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.

2.2.1 Binary-To-Decimal Conversion / Decimal-To-Binary 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 contains a 1.

Binary	**Decimal**
$(10011)_2$	
$2^4+0+0+2^1+2^0$	$=16+0+0+2+1$
Result	$(19)_{10}$

And

Binary	**Decimal**
$(10110101)_2$	
$2^7+0+2^5+2^4+0+2^2+0+2^0$	$=128+0+32+16+0+4+0+1$
Result	$(181)_{10}$

You should have noticed that the method is to find the weights (i.e., powers of 2) for each bit position that contains a 1 and then to add them up.

Decimal-To-Binary Conversion

There are 2 methods

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

Reverse of Binary-To-Decimal Method

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

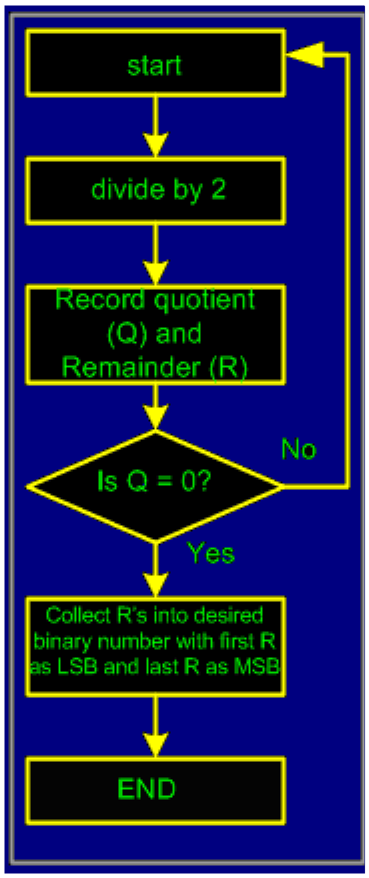
Repeat Division-Convert decimal to binary

This method uses repeated division by 2.

E.g. Convert $(25)_{10}$ to binary

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
½	= 0 + remainder of 1	1 (Most Significant Bit)
Result	$(25)_{10}$	$(11001)_2$

The Flow chart for repeated-division method is as follows:


start
divide by 2
Record quotient (Q) and Remainder (R)
Is Q = 0?
No
Yes
Collect R's into desired binary number with first R as LSB and last R as MSB
END

2.2.2 Binary-To-Octal / Octal-To-Binary Conversion

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

Each Octal digit is represented by three binary digits.

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

Repeat Division-Convert decimal to octal

This method uses repeated division by 8.

E.g. Convert 177_{10} to octal and binary

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	$(177)_{10}$	$(261)_8$
Binary		$(010110001)_2$

2.2.3 Hexadecimal-To-Decimal Conversión / Decimal To Hexadecimal Conversión

Hexadecimal-To-Decimal Conversión

E.g. $2AF_{16} = 2 \text{ x } (16^2) + 10 \text{ x } (16^1) + 15 \text{ x } (16^0) = 687_{10}$

Decimal to Hexadecimal Conversion

This method uses repeated division by 16. E.g. convert $(378)_{10}$ to hexadecimal

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

2.2.4 Hexadecimal to Binary Conversion

Hexadecimal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	0000	0001	0010	0011	0100	0101	0110	0111

Hexadecimal Digit	8	9	A	B	C	D	E	F
Binary Equivalent	1000	1001	1010	1011	1100	1101	1110	1111

Each Hexadecimal digit is represented by **four** bits of binary digit.

Example:

$1011\ 0010\ 1111_2 = (1011)\ (0010)\ (1111)_2 = B\ 2\ F_{16}$

2.2.5 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.

Example:

Convert $5A8_{16}$ to Octal.

Hexadecimal	Binary/Octal
$5A8_{16}$	= **0101** 1010 **1000** (Binary)
	= **010** 110 **101** 000 (Binary)
Result	= 2 6 5 0 (Octal)

2.3 Signed binary fractions

Signed binary fractions are formed much like signed integers. We will work with a single digit to the left of the decimal point, and this will represent the number -1 (= - (2^0)). The rest of the representation of the fraction remains unchanged. Therefore this leftmost bit represents a sign bit just as with two's complement integers. If this bit is set, the number is negative, otherwise the number is positive. The largest positive number that can be represented is still $1\text{-}2^{-m}$ but the largest negative number is -1. The resolution is still $1\text{-}2^{-m}$.

Signed binary fractions are easily extended to include all numbers by representing the number to the left of the decimal point as a 2's complement integer, and the number to the right of the decimal point as a positive fraction. Thus

$$-6.625_{10} = (-7+0.375)_{10} = 1001.011_{2}$$

Note that as with two's complement integers, the leftmost digit can be repeated any number of times without affecting the value of the number

2.4 Binary Codes

Binary codes are codes which are represented in binary system with modification from the original ones. Below we will be seeing the following:

- Weighted Binary Systems
- Non Weighted Codes

2.4.1 Weighted Binary Systems

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. The binary counting sequence is an example.

Decimal	8421	2421	5211	Excess-3
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0011	0101
3	0011	0011	0101	0110
4	0100	0100	0111	0111
5	0101	1011	1000	1000
6	0110	1100	1010	1001
7	0111	1101	1100	1010
8	1000	1110	1110	1011
9	1001	1111	1111	1100

8421 Code/BCD Code

The BCD (Binary Coded Decimal) is a straight assignment of the binary equivalent. It is possible to assign weights to the binary bits according to their positions. The weights in the BCD code are 8,4,2,1.

Example: The bit assignment 1001 can be seen by its weights to represent the decimal 9 because:

1x8+0x4+0x2+1x1 = 9

2421 Code

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

5211 Code

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

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 Codes

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.

2.4.2 Non Weighted Codes

Non weighted codes are codes that are not positional weighted. That is, each position within the binary number is not assigned a fixed value.

Excess-3 Code

Excess-3 is a non weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the 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. The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code. In digital Gray code has got a special place.

Decimal Number	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary to Gray Conversion

- Gray Code MSB is binary code MSB.
- Gray Code MSB-1 is the XOR of binary code MSB and MSB-1.
- MSB-2 bit of gray code is XOR of MSB-1 and MSB-2 bit of binary code.
- MSB-N bit of gray code is XOR of MSB-N-1 and MSB-N bit of binary code.

2.5 Floating Point Numbers

A real number or floating point number is a number which has both an integer and a fractional part. Examples for real decimal numbers are 223.45, 0.1234, -0.123, etc. Examples for real binary numbers are 1101.1101, 0.1000, -1.001, etc. In general, floating point numbers are expressed in exponential notation.

For example the decimal number

- 500.0 can be written as 5 x 10^2.
- 622.45 can be written as 6.2245 x 10^2.

Similarly, the binary number 1010.001 can be written as 1.010001 x 10^3.

The general form of a number N can be expressed as

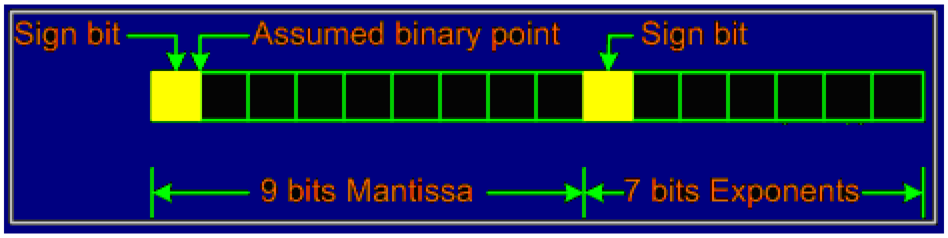
$N = \pm m \times b^{\pm e}$.

Where m is mantissa, b is the base of number system and e is the exponent. A floating point number is represented by two parts. The number first part, called mantissa, is a signed fixed point number and the second part, called exponent, and specifies the decimal or binary position.

Binary Representation of Floating Point Numbers

A floating point binary number is also represented as in the case of decimal numbers. It means that mantissa and exponent is expressed using signed magnitude notation in which one bit is reserved for sign bit.

Consider a 16-bit word used to store the floating point numbers; assume that 9 bits are reserved for mantissa and 7 bits for exponent and also assume that the mantissa part is represented in fraction system. This implies the assumed binary point is at the mantissa sign bit immediate right.



Example

A binary number 1101.01 is represented as

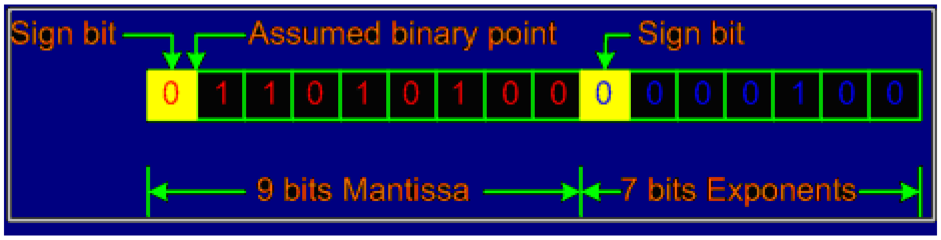
Mantissa = 110101 = $(1101.01)_2$ = 0.110101 X 2^4

Exponent = $(4)_{10}$

Expanding mantissa to 8 bits we get 11010100

Binary representation of exponent $(4)_{10}$ = 000100

The required representation is



BINARY ARITHMETIC

Multiple Choice Questions:

Q1 The decimal number 122 is equals to the binary number

a) 1111010
b) 111010
c) 1111011
d) 100110

Q2 The BCD number for decimal 473 is__________________

a) 111011010
b) 110001110011
c) 010001110011
d) 010011110011

Q3 $(110)_2 = (?)_8$

a) 27
b) 72
c) 80
d) 08

Q4 The 2's complement of 110001011 is ___________

a) 001110101
b) 001110100
c) 001111100
d) 000110100

Q5 How many nibbles count in the number 1111 0101 1110 1001?

a) 4
b) 8
c) 2
d) 16

Q6 What is the hexadecimal equivalent of $(467)_8$?

a) $(137)_{16}$
b) $(731)_{16}$
c) $(467)_8$
d) $(137)_8$

Q7 If 3 in binary system 011 then 7 will be

a) 111
b) 0111
c) 1110
d) None of the above

Q8 Excess three code of decimal 5 is__________

a) 0001
b) 1000
c) 0111
d) All of the above

Q9 What is binary equivalent of AC6 is__________

a) 1010 1100 0110

b) 011011001010

c) None of the above

d) Both of the above

Q10 What is the decimal equivalent of 1100 is________

a)13

b) 12

c) 9

d) none of the above

Q11 8421 code is

a) Self –complementing code

b) Weighted code

c) Non-weighted code

d) Alphanumeric code

Q12 ASCII code is a

a) 5-bit code

b) 7-bit code

c) 8-bit code

d) 10-bit code

Chapter 3

Logic Gates & Their Symbols

3.0 Symbolic Logic

Boolean algebra derives its name from the mathematician George Boole. Symbolic Logic uses values, variables and operations:

- **True** is represented by the value **1**.
- **False** is represented by the value **0**.

Variables are represented by letters and can have one of two values, either 0 or 1. Operations are functions of one or more variables.

- **AND** is represented by A.B
- **OR** is represented by A+B
- **NOT** is represented by A’. Throughout this tutorial the B' form will be used and sometime! X will be used.

These basic operations can be combined to give expressions.

Example:

- A.B
- W.A.B + Z

Truth Tables

Truth tables are a means of representing the results of a logic function using a table. They are constructed by defining all possible combinations of the inputs to a function, and then calculating the output for each combination in turn. For the three functions we have just defined, the truth tables are as follows.

Truth tables may contain as many input variables as desired **F(X, Y, Z) = X.Y + Z**

X	Y	Z	F(X,Y,Z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3.1 LOGIC GATES

Logic gates process one or more input signals in a logical fashion. Depending on the input value or voltage, the logic gate will either output a value of '1' for ON or a value of '0' for OFF.

Logic Gates allow simplification of circuit operation. Logic gates are also called switches. With the advent of integrated circuits, switches have been replaced by TTL (Transistor Transistor Logic) circuits and CMOS circuits.

Gates Types

- AND
- OR
- NOT
- NAND
- NOR
- EX-OR
- EX-NOR

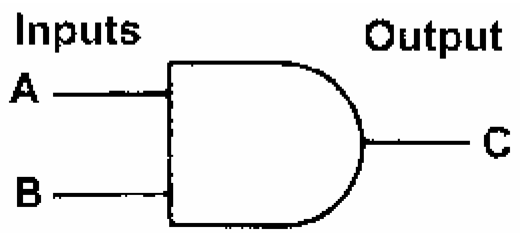
The five common logic gates used are:-
AND, OR, NOT, NAND, NOR

3.1.1 AND Gate

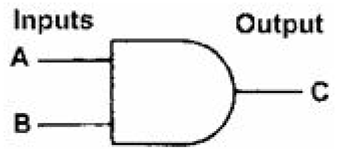
The AND gate performs logical multiplication, commonly known as AND function. The AND gate has two or more inputs and single output. The output of AND gate is HIGH only when all its inputs are HIGH (i.e. even if one input is LOW, Output will be LOW).

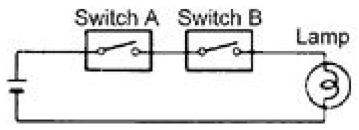
If A and B are two inputs, then output C can be represented mathematically as C = A.B, Here dot (.) denotes the AND operation. Truth table and symbol of the AND gate is shown in the figure below.

Symbol



HOW DOES THE "AND" GATE WORK?





'AND' gates are like two or more switches in series. All the switches have to be closed ('ON' or a value of '1') in order to make the lamp (output C) turn on.

If all inputs are not "ON", the output is "OFF".

Truth Table

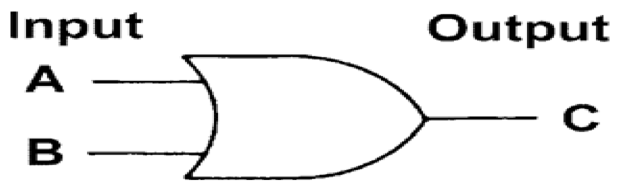
A	B	C=(X.Y)
0	0	0
0	1	0
1	0	0
1	1	1

3.1.2 OR Gate

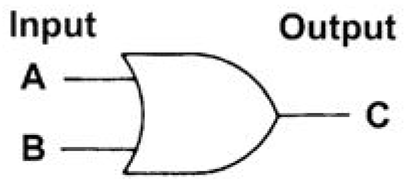
The OR gate performs logical addition, commonly known as OR function. The OR gate has two or more inputs and single output. The output of OR gate is HIGH only when any one of its inputs are HIGH (i.e. even if one input is HIGH, Output will be HIGH).

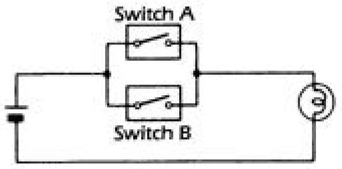
If A and B are two inputs, then output C can be represented mathematically as C = X+Y. Here plus sign (+) denotes the OR operation. Truth table and symbol of the OR gate is shown in the figure below.

Symbol



HOW DOES THE "OR" GATE WORK?





An 'OR' gate is like two or more switches in parallel. Only one switch needs to be closed ('ON' or a value of '1') in order to make the lamp (output C) turn 'ON' with a value of '1'.

Truth Table

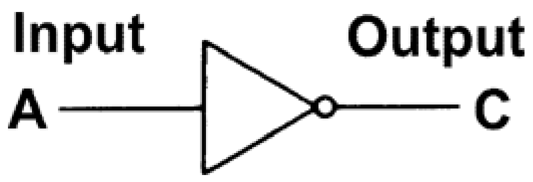
A	B	C=(X+Y)
0	0	0
0	1	1
1	0	1
1	1	1

3.1.3 NOT

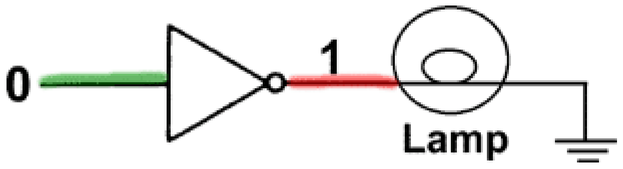
The NOT gate reverse the input signal value, commonly known as NOT function. The NOT gate has only single inputs and single output. The output of NOT gate is HIGH only when its inputs are HIGH.

If A is inputs, then output C can be represented mathematically as C = A’. Here negation sign (‘) denotes the NOT operation. Truth table and symbol of the OR gate is shown in the figure below.

Symbol



HOW DOES THE "NOT" GATE WORK?



In NOT gate, if the input value is '0', the output value will be '1'. If the input value is '1', then the output value will be '0'. NOT gates can be referred to as inverters; whatever the input signal is the output is always the opposite.

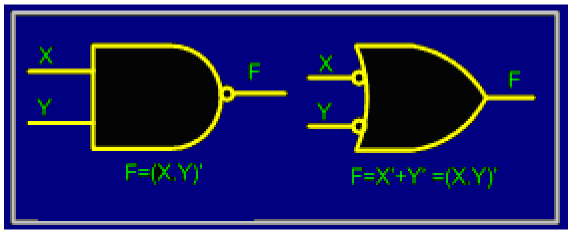
A	C
0	1
1	0

3.2 Universal Gates

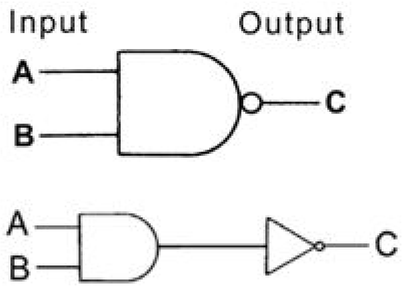
Universal gates are the ones which can be used for implementing any gate like AND, OR and NOT, or any combination of these basic gates; NAND and NOR gates are universal gates. But there are some rules that need to be followed when implementing NAND or NOR based gates.

To facilitate the conversion to NAND and NOR logic, we have two new graphic symbols for these gates.

2.2.1NAND Gate



Truth table and working of NAND gate

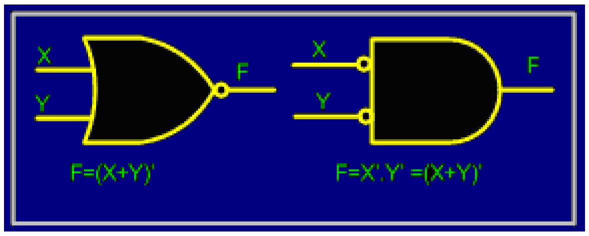


TRUTH TABLE

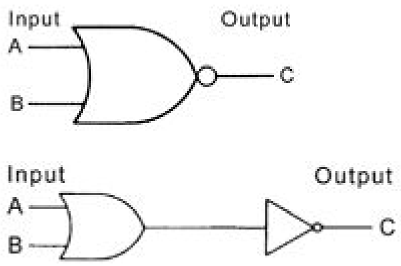
Input	Input	Output
A	**B**	**C**
0	0	1
0	1	1
1	0	1
1	1	0

A NAND gate is the combination of both an AND gate and a NOT gate. It operates the same as an AND gate but the output will be the opposite. Remember the NOT gate does not always have to be on the output leg; it could be used to invert an input signal also.

3.2.2 NOR Gate



Truth table and working of NOR gate



TRUTH TABLE

Input	Input	Output
A	**B**	**C**
0	0	1
0	1	0
1	0	0
1	1	0

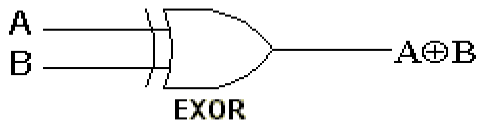
A NOR gate is the combination of both an OR gate and a NOT gate. It operates the same as an OR gate, but the output will be the opposite.

3.3 TWO ADDITIONAL GATES

EX-OR gate

This operation is widely used in digital circuits. Here when both the inputs are same the output will be “0”and when the inputs aren’t same the output is”1”.

Symbol



Truth Table

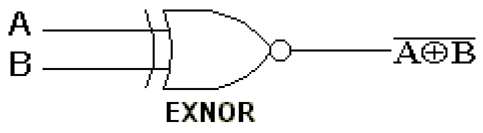
2 Input EXOR gate		
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

An encircled plus sign (⊕) is used to show the EX-OR operation.

EX-NOR gate

This operation is also used in digital circuits. Here when both the inputs are same the output will be “1”and when the inputs aren’t same the output is”0”.

Symbol



Truth Table

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Multiple Choice Questions:

Q1 which logic gate is similar to the function of two parallel switches?

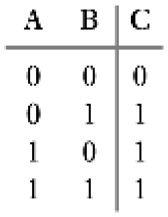
a) AND

b) NAND

c) OR

d) NOR

Q2 Which logic gate has the following truth table?



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

a) An exclusive NOR gate.

b) A two-input AND gate.

c) A two-input OR gate.

d) An exclusive OR gate

Q3 Which of the following is not a universal gate:

a) AND

b) NAND

c) NOR

d) OR

Q4 NAND gate is a combination of _______

a) AND and NOT gate

b) NOT and AND gate

c) NAND and NOT gate

d) All of the above

Q5 In which gate outputs will be true, if inputs are different:

a) OR

b) NAND

c) EX-OR

d) NOR

Q6 In which gate output is complements of its input:

a) AND

b) NOT

c) NOR

d) None of the above

Q7 Which of the following gate can be use to represent W.A.B + Z expression:

a) NOT
b) AND
c) OR
d) All the above

Q8 AND gate perform___________

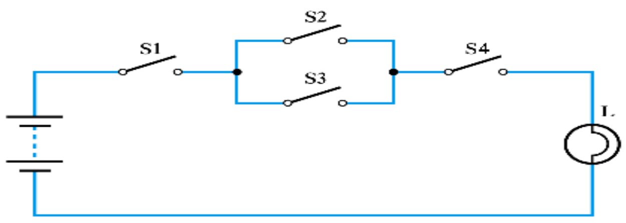
a) Logical Addition

b) Logical Multiplication

c) Both a and b

d) None of the above

Q9 What logic function corresponds to the following arrangement?



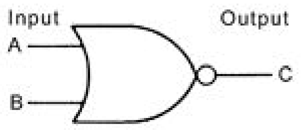
a) *L* = S1 OR (S2 AND S3) OR S4.

b) *L* = S1 AND (S2 OR S3) AND S4.

c) *L* = (S1 AND S2) OR (S3 AND S4).

d) *L* = (S1 OR S2) AND (S3 OR S4)

Q10 The circuit diagram given below represents:



a) AND gate

b) NOT gate

c) NOR gate

d) All the above

Chapter 4

Boolean Algebra & Their Simplification

4.1 De Morgan's Law

DeMorgan a logician and mathematician who was acquainted with boole, proposed two theorm that are important part of boolean algebra.

They are stated as follows in eq^n form:

1) $(A.B)' = A' + B'$
2) $(A+B)' = A'.B'$

First theorem states that the complement of a product is equal to the sum of the complement.

It says that the complement of the two or more variables AND$_{ed}$ is the same as the OR of the complements of each individual variable.

Proof of $(A.B)' = A' + B'$

A	B	A.B	(A.B)'
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

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

Second theorem states that the complement of a sum is equal to the product of the complement.

It says that the complement of the two or more variables ORed is the same as AND of the complements of each individual variable.

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

A	**B**	**A+B**	**(A+B)'**
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

A	**B**	**A'**	**B'**	**A'.B'**
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

The two truth tables are identical, and so the two expressions are identical.

Note: De Morgan's Laws are applicable for any number of variables.

4.1.1 Minterms and Maxterms

Any Boolean expression may be expressed by two ways:-

A) Minterm

B) Maxterm.

To do this we must first define the concept of a literal. A literal is a single variable within a term which may or may not be complemented. For an expression with N variables, minterm and maxterm are defined as follows:

- A min term is the product of N distinct literals where each literal occurs exactly once.
- A max term is the sum of N distinct literals where each literal occurs exactly once.

For a two-variable expression, the minterms and maxterms are as follows

A	B	Minterm	Maxterm
0	0	A'.B'	A+B
0	1	A'.B	A+B'
1	0	A.B'	A'+B
1	1	A.B	A'+B'

For a three-variable expression, the minterms and maxterms are as follows

A	B	C	Minterm	Maxterm
0	0	0	A'.B'.C'	A+B+C
0	0	1	A'.B'.C	A+B+C'
0	1	0	A'.B.C'	A+B'+C
0	1	1	A'.B.C	A+B'+C'
1	0	0	A.B'.C'	A'+B+C
1	0	1	A.B'.C	A'+B+C'
1	1	0	A.B.C'	A'+B'+C
1	1	1	A.B.C	A'+B'+C'

This allows us to represent expressions in either Sum of Products or Product of Sums forms

Sum Of Products (SOP)

The Sum of Products form represents an expression as a sum of minterms.

F (A, B,) = Sum (a_k.m_k)

where a_k is 0 or 1 and m_k is a minterm.

To derive the Sum of Products form from a truth table, OR together all of the minterms which give a value of 1.

Example - SOP

Consider the truth table

A	**B**	**F**	**Minterm**
0	0	0	A'.B'
0	1	0	A'B
1	0	1	A.B'
1	1	1	A.B

Here SOP is f (A.B) = A.B' + A.B

Product Of Sum (POS)

The Product of Sums form represents an expression as a product of maxterms.

F(A, B, ...) = Product (b_k + M_k), where b_k is 0 or 1 and M_k is a maxterm.

To derive the Product of Sums form from a truth table, AND together all of the maxterms which give a value of 0.

Example - POS

Consider the truth table from the previous example.

A	**B**	**F**	**Maxterm**
0	0	1	A+B
0	1	0	A+B'
1	0	1	A'+B
1	1	1	A'+B'

Here POS is F(A,B) = (A+B')

Conversion between POS and SOP

Conversion between the two forms is done by application of De Morgan's Laws.

Simplification

As with any other form of algebra you have encountered, simplification of expressions can be performed with Boolean algebra.

Example

Show that A.B.C' + A'.B.C' + B.C = B

A.B.C' + A'.B.C' + B.C = B.C' + B.C = B

LOGIC SYSTEM

Fixed Logic Systems

A fixed logic system has two possible choices for representing true and false.

Positive Logic

In a positive logic system, a high voltage is used to represent logical true (1), and a low voltage for a logical false (0).

Negative Logic

In a negative logic system, a low voltage is used to represent logical true (1), and a high voltage for a logical false (0).

In positive logic circuits it is normal to use +5V for true and 0V for false.

4.2 **Simplification of Boolean functions**

Simplification of Boolean functions is mainly used to reduce the gate count of a design. Less number of gates means less power consumption, sometimes the circuit works faster and also when number of gates is reduced, cost also comes down.

There are many ways to simplify a logic design; some of them are given below.

- Algebraic Simplification.
- ->Simplify symbolically using theorems/postulates.
- ->Requires good skills
- Karnaugh Maps.
- ->Diagrammatic technique using 'Venn-like diagram'.
- ->Limited to no more than 6 variables.

We have already seen how Algebraic Simplification works, so let’s concentrate on Karnaugh aps or simply k-maps.

Karnaugh Maps (K-MAP)

It is a Graphical display of minterms in which logically adjacent terms are also physically adjacent.

The Karnaugh map or K-map is a tool which is used to minimize switching functions. K-maps arrange minterms, as well as maxterms, in such a way that logically adjacent minterms becoming physically adjacent as well. Minterms which are logically adjacent can be combined into a simpler equation of equivalent value(1).

K-maps can evaluate between 2-variable and 5-variable functions. This is a compact way of representing a truth table and is a technique that is used to simplify logic expressions .K-maps can be formed from truth tables. Inputs in a truth table which produce outputs of 1 are minterms, while inputs which produce outputs of 0 are maxterms.

In a k-map which evaluates a sum of products **(SOP)** equation, the **1's** represent minterms and which evaluates a product of sums **(POS)** equation, the **0's** represent maxterms.

Every single entry in a truth table corresponds to a cell in a k-map. The top and sides of the k-map are lined with 1's and 0's in gray code, so that only one variable changes between adjacent cells along every dimension.

The K-MAP Format

The K-map is composed of an arranged of adjacent "cells", each representing one particular combination of variable in product form. Since the total number of combination of n variable and their complements is 2^n, the K-map consists of 2^n cells.

pending on what kind of equation is evaluated, SOP or POS, 1's or 0's, respectively, are grouped in rectangular shapes. Each grouping must have a number of cells which correspond to a power of 2 (groups of 1, 2, 4, or 8). Don't cares may also be placed into k-maps and can be treated as either 1's or 0's.

Minimization Technique

- Based on the Unifying Theorem: X + X' = 1
- The expression to be minimized should generally be in sum-of-product form (If necessary, the conversion process is applied to create the sum-of-product form).
- The function is mapped onto the K-map by marking a 1 in those squares corresponding to the terms in the expression to be simplified (The other squares may be filled with 0's).
- Pairs of 1's on the map which are adjacent are combined using the theorem Y(X+X') = Y where Y is any Boolean expression (If two pairs are also adjacent, then these can also be combined using the same theorem).

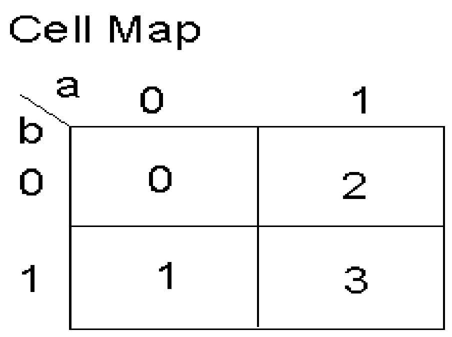
- The minimization procedure consists of recognizing those pairs and multiple pairs.
- ->These are circled indicating reduced terms.
    - Groups which can be circled are those which have two (2^1) 1's, four (2^2) 1's, eight (2^3) 1's, and so on.
- ->Note that because squares on one edge of the map are considered adjacent to those on the opposite edge, group can be formed with these squares.
- Note that because squares on one edge of the map are considered adjacent to those on the opposite edge, group can be formed with these squares.
- ->Groups are allowed to overlap.
- The objective is to cover all the 1's on the map in the fewest number of groups and to create the largest groups to do this.
- Once all possible groups have been formed, the corresponding terms are identified.
- ->A group of two 1's eliminates one variable from the original minterm.
- ->A group of four 1's eliminates two variables from the original minterm.
- ->A group of eight 1's eliminates three variables from the original minterm, and so on.
- ->The variables eliminated are those which are different in the original minterms of the group.

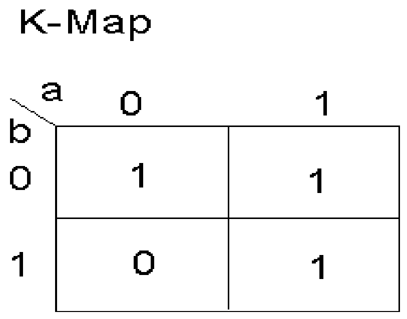
4.2.1 2-Variable K-Map

In any K-Map, each square represents a minterm. Adjacent squares always differ by just one literal (So that the unifying theorem may apply: a + a' = 1). For the 2-variable case (e.g.: variables a, b), the truth table cell map and K-map can be drawn as below. Two variable map is the one which has got only two variables as input.

TRUTH TABLE

minterm	a	b	f
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

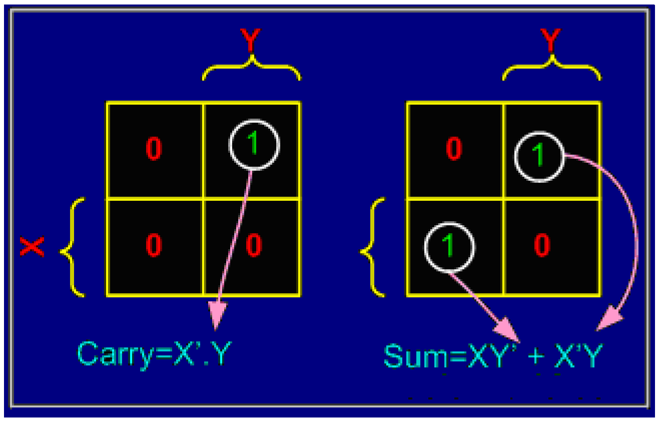




Example- Carry and Sum of a half adder

In this example we have the truth table as input, and we have two output functions. Generally we may have n output functions for m input variables. Since we have two output functions, we need to draw two k-maps (i.e. one for each function). Truth table of 1 bit adder is shown below. Draw the k-map for Carry and Sum as shown below.

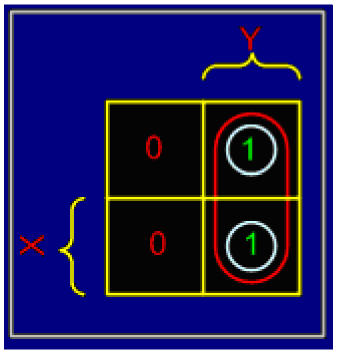
X	**Y**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Example - X'Y+XY

In this example we have the equation as input, and we have one output function. Draw the k-map for function F with marking 1 for X'Y and XY position. Now combine two 1's as shown in figure to form the single term. As you can see X and X' get canceled and only Y remains.

F = Y



4.2.2 3-Variable K-Map

There are 8 minterms for 3 variables (a, b, c). Therefore, there are 8 cells in a 3-variable K-map. One important thing to note is that K-maps follow the gray code sequence, not the binary one.

Using gray code arrangement ensures that minterms of adjacent cells differ by only ONE literal. (Other arrangements which satisfy this criterion may also be used.)

Each cell in a 3-variable K-map has 3 adjacent neighbours. In general, each cell in an n-variable K-map has n adjacent neighbors.

Truth Table

minterm	a	b	c	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Cell Map

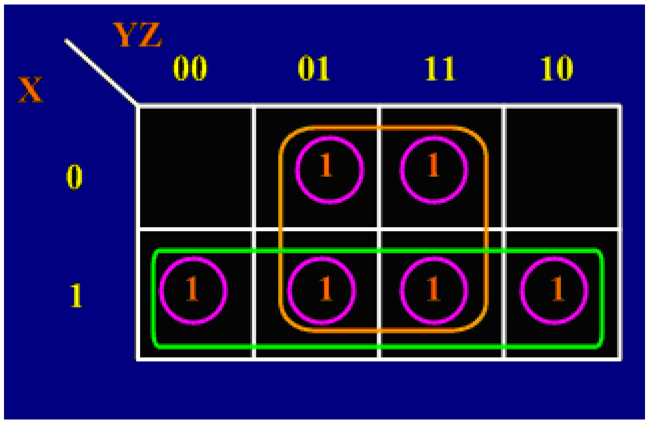
c \ ab	00	01	11	10
0	0	2	6	4
1	1	3	7	5

K-Map

c \ ab	00	01	11	10
0	1	1	0	0
1	0	1	1	1

Example

$F(X,Y,Z) = \Sigma(1,3,4,5,6,7)$



F = X + Z

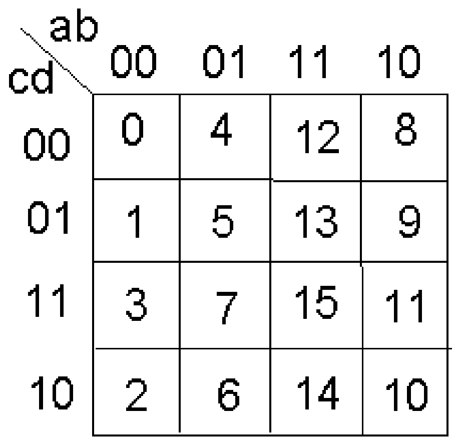
4.2.3 4-Variable K-Map

There are 16 cells in a 4-variable (a, b, c, d); K-map as shown in the figure below. There are 2 wrap-around: a horizontal wrap-around and a vertical wrap-around. Every cell thus has 4 neighbours.

Truth Table

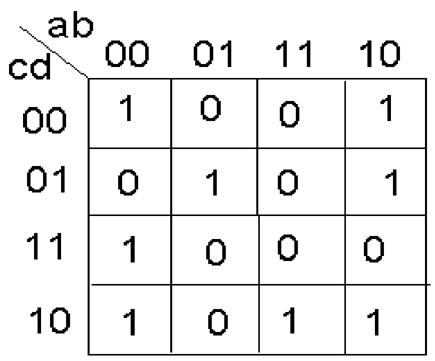
minterm	a	b	c	d	f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Cell Map



cd \ ab	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

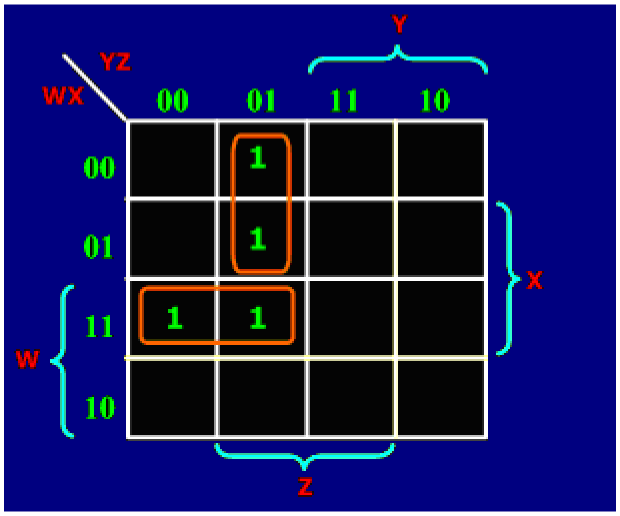
K-Map



cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	0	0	0
10	1	0	1	1

Example:

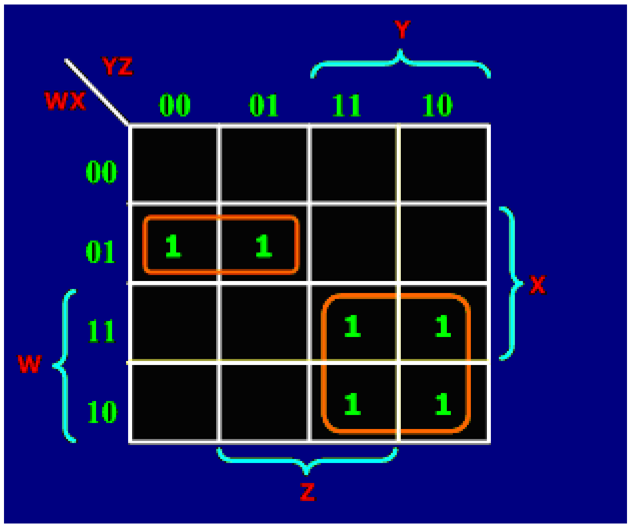
F(W,X,Y,Z) =Σ (1,5,12,13)



F = WY'Z + W'Y'Z

Example:

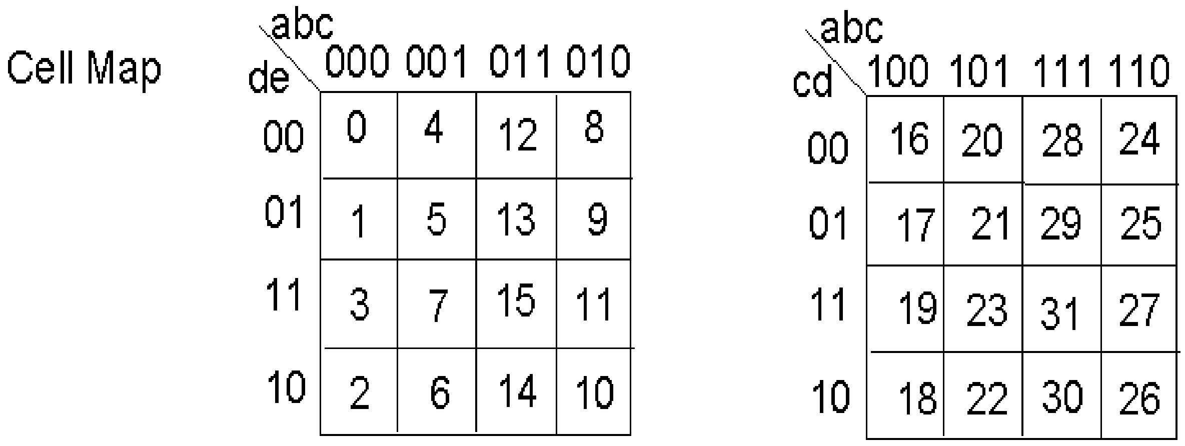
F(W,X,Y,Z) = Σ (4, 5, 10, 11, 14, 15)



F = W'XY' + WY

4.2.4 5-Variable K-Map

There are 32 cells in a 5-variable (a, b, c, d, e); K-map as shown in the figure below.

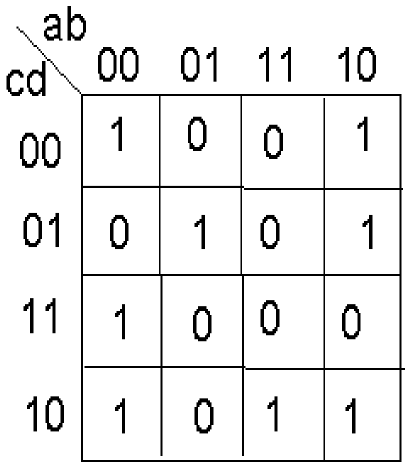


Cell Map

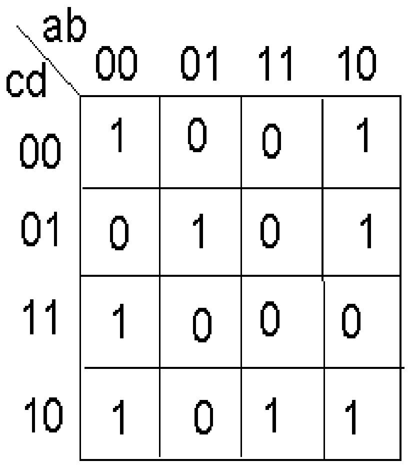
de \ abc	000	001	011	010
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

cd \ abc	100	101	111	110
00	16	20	28	24
01	17	21	29	25
11	19	23	31	27
10	18	22	30	26

K-Map



cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	0	0	0
10	1	0	1	1



cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	0	0	0
10	1	0	1	1

4.3 Inverse Function

The 0's on a K-map indicate when the function is 0.

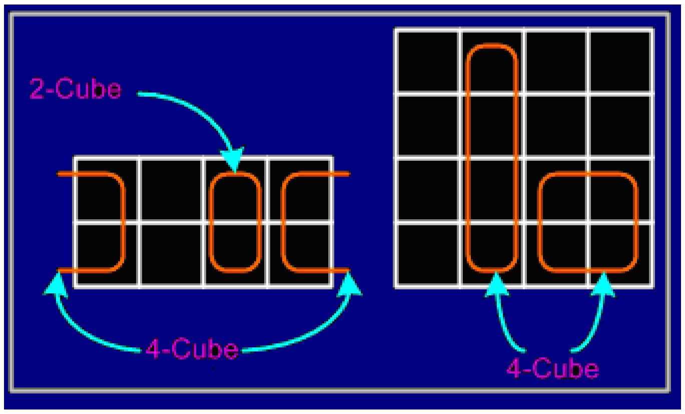
- We can minimize the inverse function by grouping the 0's (and any suitable don't cares) instead of the 1's.
- This technique leads to an expression which is not logically equivalent to that obtained by grouping the 1's (i.e., the inverse of X! = X').
- Minimizing for the inverse function may be particularly advantageous if there are many more 0's than 1's on the map.
- We can also apply De Morgan's theorem to obtain a product-of-sum expression.

4.4 Grouping/Circling K-maps

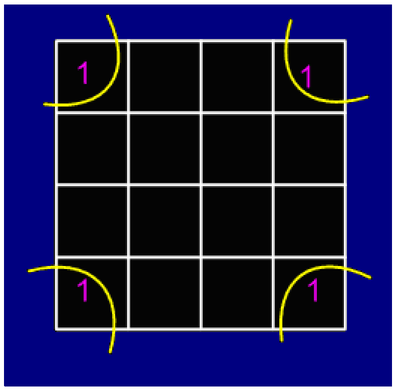
The power of K-maps is in minimizing the terms, K-maps can be minimized with the help of grouping the terms to form single terms. When forming groups of squares, observe/consider the following:

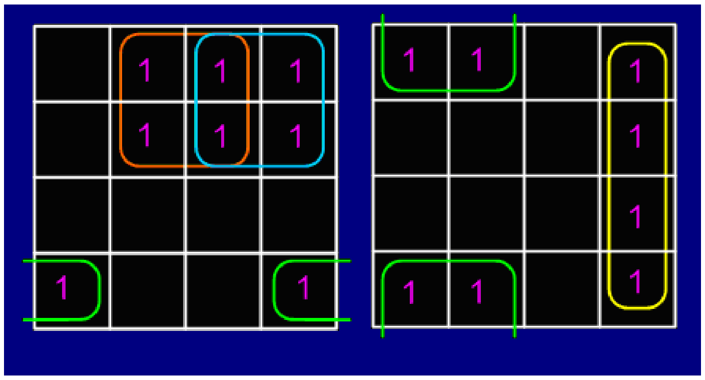
Every square containing 1 must be considered at least once.

- A square containing 1 can be included in as many groups as desired.
- A group must be as large as possible.

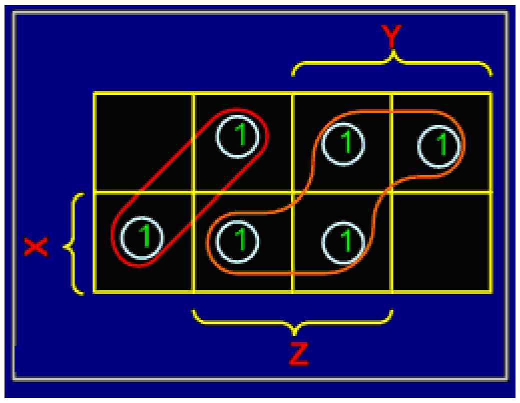


- If a square containing 1 cannot be placed in a group, then leave it out to include in final expression.
- The number of squares in a group must be equal to 2
- , i.e. 2,4,8,.
- The map is considered to be folded or spherical; therefore squares at the end of a row or column are treated as adjacent squares.
- The simplified logic expression obtained from a K-map is not always unique. Groupings can be made in different ways.
- Before drawing a K-map the logic expression must be in canonical form.





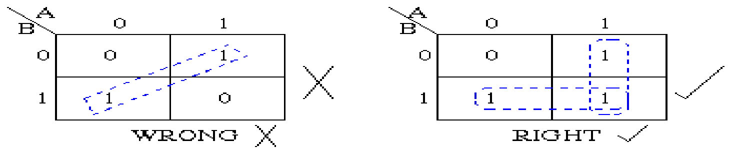
Example of invalid groups



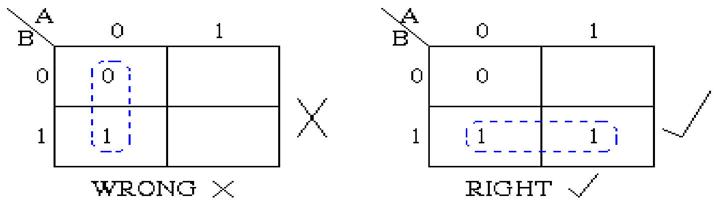
4.5 DO'S AND DON'T

The Karnaugh map uses the following rules for the simplification of expressions by *grouping* together adjacent cells containing *ones*

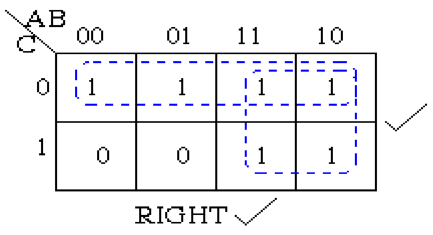
i) Groups may be horizontal or vertical, but not diagonal.

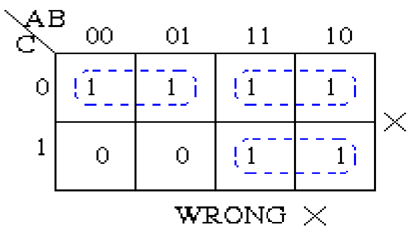


ii) Groups may not include any cell containing a zero



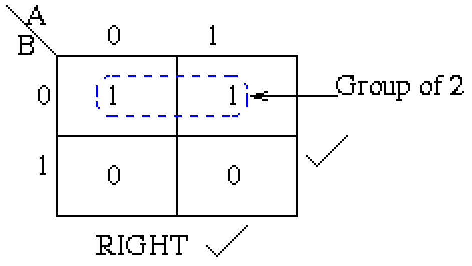
iii) Each group should be as large as possible

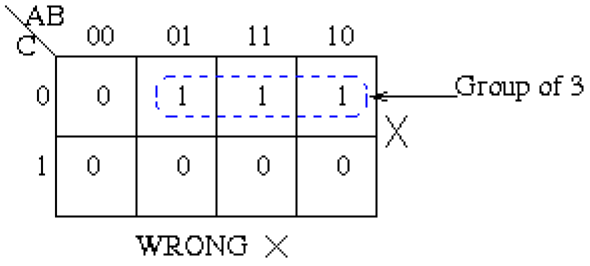


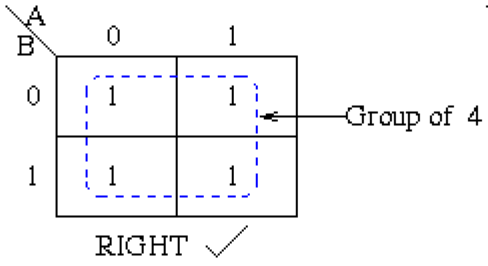


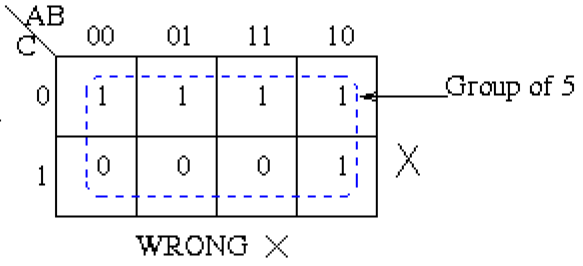
(Note that no Boolean laws broken, but not sufficiently minimal)

iv) Groups must contain 1, 2, 4, 8, or in general 2^n cells.
That is if n = 1, a group will contain two 1's since $2^1 = 2$.
If n = 2, a group will contain four 1's since $2^2 = 4$.

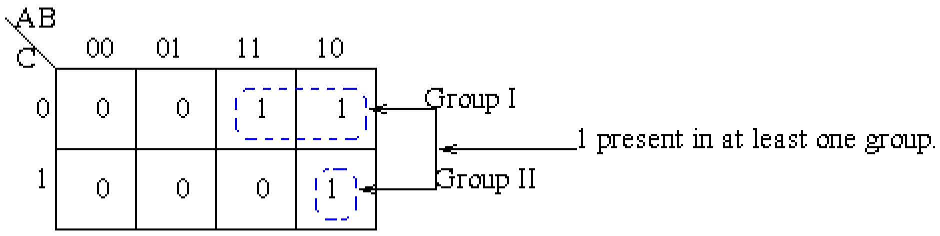




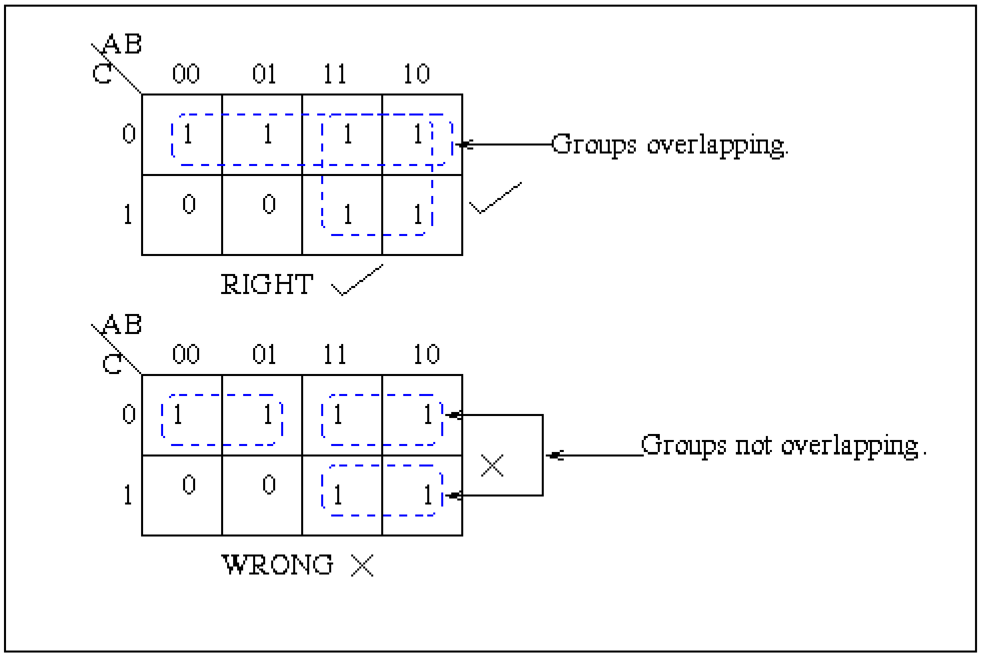




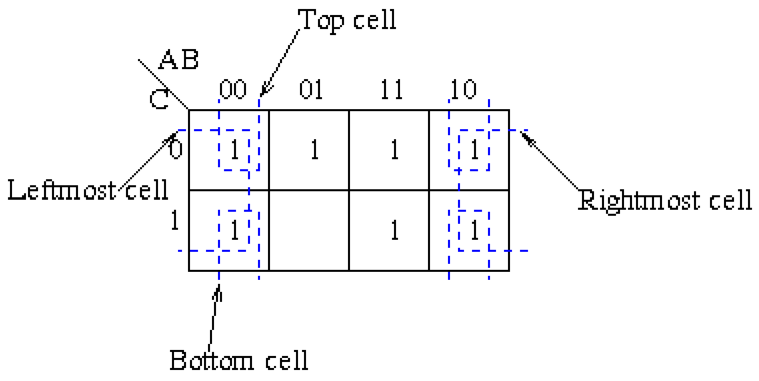
v) Each cell containing a *one* must be in at least one group.



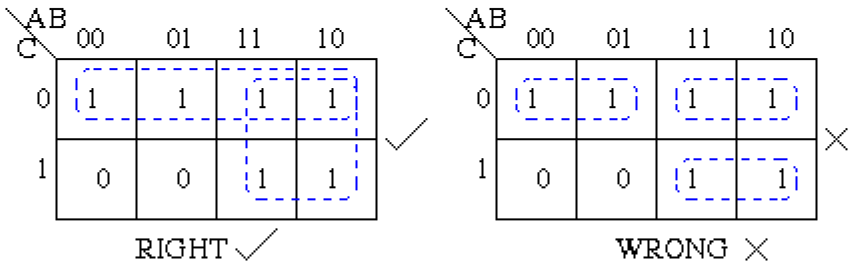
vi) Groups may overlap.



vii)Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



viii) There should be as few groups as possible, as long as this does not contradict any of the previous rules.



4.6 Don't cares

If a certain combination of variables cannot occur or it does not matter what the outputs are if the combination does occur, the combination is known as a *don't-care condition and* Such outputs are entered as X in the truth-table. The X is called as *don't-care condition*.
The X mark in a cell may be taken either as 0 or as 1, depending upon which one lead to a simpler expression.

A function with one or more don't-care conditions is called an *incompletely specified function*

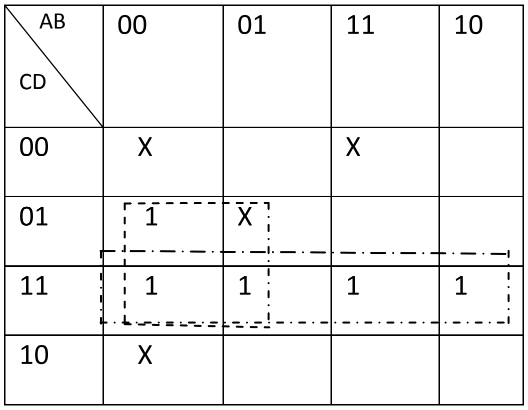
The function can be specified in one of the following ways:-

a) In terms of minterms and don't care conditions
b) In terms of maxterms and don't care conditions
c) In terms of truth table

Example:

In terms of minterms and don't care conditions

F(A,B,C,D)= Σm(1,3,7,11,15)+don't care(0,2,5,12)



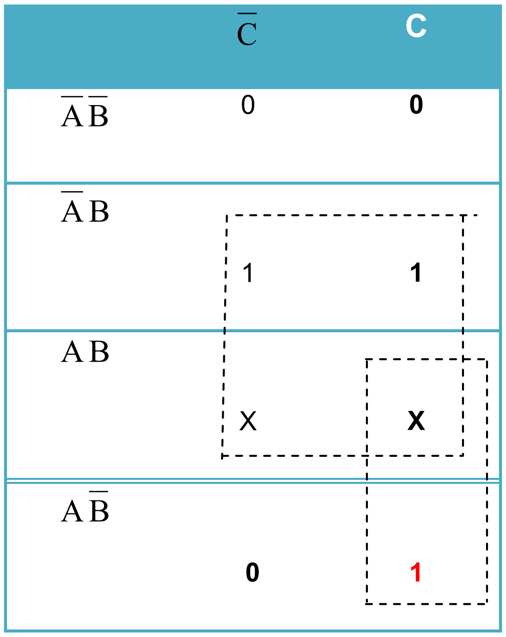
AB / CD	00	01	11	10
00	X		X	
01	1	X		
11	1	1	1	1
10	X			

Y=A'D+CD

EXAMPLE:

In terms of truth table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X



	$\bar{C}$	C
$\bar{A}\bar{B}$	0	**0**
$\bar{A}B$	1	**1**
AB	X	**X**
$A\bar{B}$	**0**	**1**

Y=B+AC

Karnaugh maps also allow easy minimizations of functions whose truth tables include "**don't care**" conditions (that is sets of inputs for which the designer doesn't care what the output is) because "don't care" conditions can be included in a ring to make it larger but do not have to be ringed.

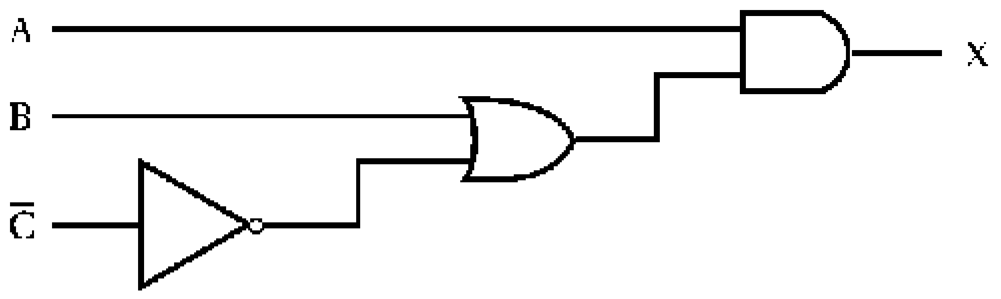
Multiple Choice Questions

Q1 In Boolean algebra the AND function is represented by the '+' sign.

a) True

b) False

Q2 In Boolean algebra the AND function is represented by the '+' sign.



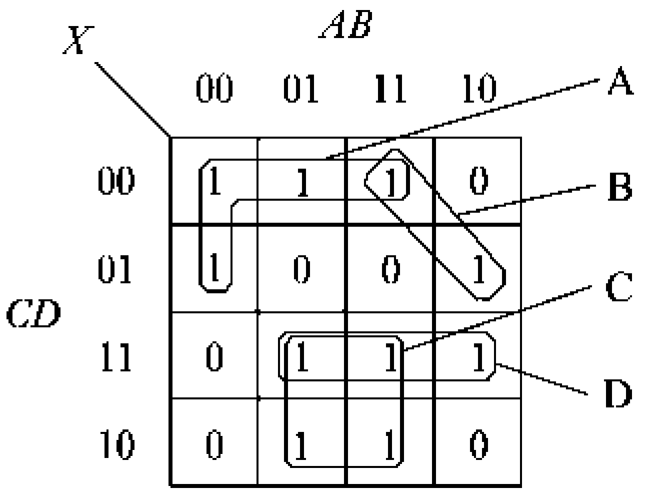
a) X = (A.B) + C

b) X = A + B + C

c) X = A + (B.C)

d) X = A.(B + C)

Q3 In the Karnaugh map shown below, which of the loops shown represents a illegal grouping?



a) A

b) B

c) C

d) D

Q4 What is the most widely used method for the automated simplification of Boolean expressions?

a) Quine-McCluskey minimisation.

b) Binary reduction.

c) Karnaugh maps

d) All the above

Q5 According to Boolean algebra which of the following relation is not valid

a) $\bar{X} = 1$

b) $X + Y = Y + X$

c) $XY = YX$

d) $X + (Y + Z) = (X + Y) + Z$

Q6 How many gate inputs are required to realize the following expression?

F=ABC+ABCD+EF+AD

a) 11
b) 15

c) 4

d) None of the above

Q7 How many gate inputs are required to realize the following expression?

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

a) 14

b) 15

c) 4

d) All the above

Q8 Is in positive logic circuits it is normal to use 0V for true and +5V for false?

a) True

b) False

Q9 How many cells present in a 5-variable format?

a) 16

b) 8

c) 32

d) None of the above

Q10 To represent X'Y+XY expression:

a) 4-variable k-map is used.

b) 3-variable k-map is used.

c) 2-variable k-map is used.

d) All of the above