MODULE 1

BINARY SYSTEMS AND BOOLEAN ALGEBRA

INTRODUCTION

Binary systems and Boolean algebra are fundamental concepts in the fields of computer science and mathematics. These principles are essential in the design and study of digital circuitry and logical operations. Binary systems are systems that exclusively utilize two different symbols or values, commonly represented as '0' & '1'.

Features of Digital systems

Digital systems possess several key characteristics that distinguish them from other types of systems. Firstly, digital systems operate using discrete values or states, as opposed to continuous values. This means that information is represented and processed in a

• Digital systems have the ability to manipulate specific components of information.

Discrete elements are defined as separate and distinct units, such as the ten decimal digits or the twenty-six letters of the alphabet.

• Digital systems employ signals, which are physical entities, to represent discrete constituents.

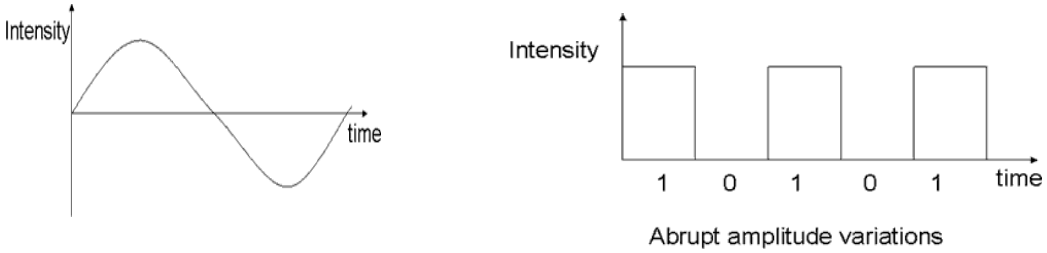
• Digital systems are defined by signals that have two unique values, making them binary in nature.

• In a digital system, a signal is employed to symbolise a single binary digit, often known as a bit. A binary digit has a numerical value of either 0 or 1.

Analog systems vs Digital systems

The comparison between analog systems and digital systems is a topic of

interest within the field of engineering and technology. Analog systems and digital systems are two distinct approaches to processing and transmitting information. Analog systems utilize continuous signals to represent and transmit data. Analog systems are capable of processing information that exhibits continuous variation. These systems are specifically engineered to process signals that vary over time and can take on any value within a continuous range of voltage, current, or other physical parameters. Digital systems utilise digital circuits that can process digital signals, represented by binary values of 0 or 1.



Digital system VS Analog system

There are several advantages associated with digital systems in comparison to analog systems.

1. **The level of ease associated with programming tasks.**

Digital systems have the capability to serve many applications by means of program modification, without necessitating any supplementary alterations in the hardware components.

2. **The decrease in expenses associated with hardware.**

The utilization of digital components has led to a decrease in the cost of hardware, which can be attributed to advancements in integrated circuit (IC) technology. The utilization of integrated circuits (ICs) enables an increased placement of components inside a specific area of Silicon, hence facilitating cost reduction.

3. **The concept of high speed refers to the ability to achieve rapid rates of movement or performance.**

The utilization of digital processing techniques facilitates rapid data processing, a capability made feasible by the advancements in Digital Signal Processing.

4. **The user's text is too short to be rewritten in an academic manner.**

The concept of high reliability refers to the ability of a system or organization to consistently perform its intended functions without failure or error. Digital systems are known for their high level of reliability, which can be attributed, in part, to the implementation of error correction codes.

5. The process of design is often perceived as being straightforward.

High reliability is the capacity of a system or organization to consistently carry out its planned functions without any failures or errors. Digital systems are known for their high level of reliability, which can be attributed, in part, to the implementation of error correction codes.

6. The outcome can be readily replicated.

Digital systems provide a better level of reproducibility in comparison to analog systems due to their independence from temperature, noise, humidity, and other component characteristics.

Drawbacks of Digital Systems

There are some drawbacks associated with digital systems.

- Analog circuits use less energy compared to their digital counterparts in order to perform equivalent functions, resulting in a lower heat output.

- Digital circuits are frequently characterized by their vulnerability, as the loss or misinterpretation of a single unit of digital data can result in a significant alteration of the overall meaning of interconnected data blocks.

- A digital computer operates by manipulating discrete units of information using a binary code.

- The occurrence of quantization error in the process of sampling analog signals.

1.2 The Concept of Binary Numbers:

The number system serves as a fundamental framework for quantifying diverse entities. Contemporary computer systems engage in communication and perform operations with binary numerals, which exclusively consist of the digits 0 and 1. The primary numerical system employed by humans is the decimal number system.

Let us examine the decimal number 18 as an illustrative example. The binary representation of this number is 10010.

It is observed that the binary number system requires a greater number of digits in order to express decimal numbers. When confronted with huge numerical values, it becomes necessary to handle binary strings of significant size. This finding led to the emergence of three novel number systems.

The octal number system is a positional numeral system that employs a base of 8. It is commonly used in computer programming and digital systems. In this system, the digits
Using a base of 16, the hexadecimal number system is a positional numeric system.

In order to define a number system, it is necessary to specify the Binary Coded Decimal (BCD) system.

The amount of distinct digits that are utilized to represent values in a number system is referred to as its base. Frequently utilised bases include of 2, 8, 10, and 16.

The radix determines the cardinality of the digit set in a given numerical system.

The initial digit in any numerical system is consistently represented by zero, while the final digit is consistently represented by the base minus one. The binary number system is a numerical representation of numbers that solely employs the digits 0 and 1.

Binary number system:

The binary number has a radix of 2. Given that the value of r is equal to 2, it is evident that just two distinct digits, namely 0 and 1, are required to represent this value. The expression of weight in the binary system is based on the concept of powers of 2.
The bit with the greatest value and the location that is farthest to the left is called the Most Significant Bit (MSB). The bit with the lowest weight, which is in the rightmost position, is frequently called the Least Significant Bit (LSB).

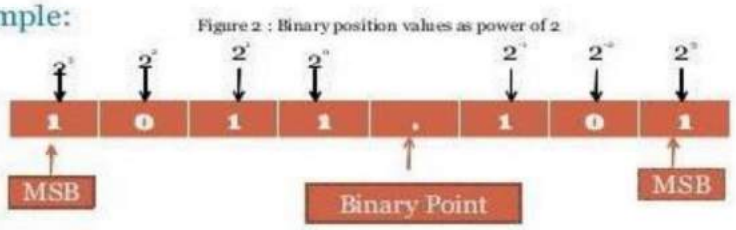
Ex.

The binary number 1001.01 can be expressed in expanded form as [(1) × 2^3] + [(0) × 2^2] + [(0) × 2^1] + [(1) × 2^0] + [(0) × 2^-1] + [(1) × 2^2].

The binary number 1001.01 can be expanded as follows: [1 × 8] + [0 × 4] + [0 × 2] + [1 × 1] + [0 × 0.5] + [1 × 0.25].

The binary number 1001.01 is equivalent to the decimal number 9.25.

- Example:



Decimal Number system

The decimal number system is a numerical system that uses the base-10, where each digit can take on values from 0 to 9. The decimal system is comprised of ten distinct symbols, namely 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. In alternative terms, the numerical system possesses a radix of 10.

Octal Number System

With a base of eight, the octal number system is a positional numeral system. It is frequently utilised in digital systems and computer programming.Digital systems exclusively function using binary numbers. The use of octal and hexadecimal notations serves as a means to portray extensive binary integers due to their frequent lengthiness. Operating with a base or radix of 8, the octal numeral system It spans the digits from 0 to 7 using the first eight digits of the decimal numeral system.

Hexa Decimal Number System

The hexadecimal number system, sometimes known as base-16, is a numerical system that uses 16 digits to represent values. The hexadecimal numeral system is characterized by a base of 16. There exists a total of 16 symbols. The initial ten digits in the decimal system range from 0 to 9, whereas the subsequent characters A, B, C, D, E, and F are employed to denote the numerical values 10, 11, 12, 13, 14, and 15, correspondingly.

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

1.3 Number Base conversions

The term "decimal" refers to a numerical system that uses a base of 10. Binary is a numerical system that uses only two digits, typically represented as 0 and 1. Octal is a numeral system that uses a base of 8. Hexadecimal is a numerical system that uses a base of 16. It is commonly used.

Human humans commonly utilize the decimal number system, but computers employ the binary number system.

Hence, it is imperative to transform the decimal number system into its corresponding binary representation.

The process of converting a binary number to an octal number involves transforming the given binary representation into its equivalent octal representation.

The process of converting a binary number to its hexadecimal equivalent is a common operation in computer science and mathematics. This conversion involves transforming a binary representation, which uses a base-2 system, into a hexadecimal representation, which uses a base-16 system.

The binary number:	001	010	011	000	100	101	110	111
The octal number:	1	2	3	0	4	5	6	7

The binary number:	0001	0010	0100	1000	1001	1010	1101	1111
The hexadecimal number:	1	2	5	8	9	A	D	F

i) The process of converting a number from octal representation to binary representation.

Each octal number converts to 3 binary digits

Code
0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111

To convert 653_8 to binary, just substitute code:

6	5	3
↓	↓	↓
110	101	011

ii) The process of converting a hexadecimal number to its binary equivalent.

iii) The process of converting a number from octal to decimal.

To convert the octal number (4057.06)8 to its decimal equivalent, we can use the positional notation system. In this system, each digit's value is determined by multiplying it with the corresponding power of 8 and summing them.

The expression can be rewritten as follows: 4(8^3) + 0.

The sum of 2048, 0, 40, 7, 0

The given numerical value, expressed in base 10, is 2095.0937

Ex: convert $(4057.06)_8$ to decimal

$$=4x8^3+0x8^2+5x8^1+7x8^0+0x8^{-1}+6x8^{-2}$$

$$=2048+0+40+7+0+0.0937$$

$$=(2095.0937)_{10}$$

iv) Decimal to Octal Conversion Ex: convert 378.9310 to octal

In this example, we will demonstrate the process of converting a decimal number, specifically 378.9310, into its octal equivalent. To convert the number (378)10 to octal, we can use the method of successive division.

$(378)_{10}$ to octal: Successive division:

The octal representation of the number (0.93)10 is requested.

The product of 0.93 multiplied by 8 is equal to 7.44

The product of 0.44 multiplied by 8 is equal to 3.52.

The product of 0.53 multiplied by 8 is equal to 4.16

The product of 0.16 multiplied by 8 equals 1.28.

The value is equal to 0.73418.

The result of converting the number 378.93 from base 10 to base 572

$(0.93)_{10}$ to octal :

0.93x8=7.44

0.44x8=3.52 ↓

0.53x8=4.16

0.16x8=1.28

=0.73418

$(378.93)_{10}=572.73418$

v) Hexadecimal to Decimal Conversion Ex: $(5C7)_{16}$ to decimal

The process of converting a hexadecimal number to its decimal equivalent can be illustrated with the example (5C7)16. The expression can be rewritten as follows: (5 x 162) + (C x 161) + (7 x 160). The sum of 1280, 192, and 7 is calculated as follows. The given value, (147)10, represents a decimal number in base-10 notation

$=(5x16^2)+(C\ x16^1)+ (7\ x16^0)$

$=1280+192+7$

$=(147)_{10}$

vi) Decimal to Hexadecimal Conversion Ex: (2598.675)10

The process of converting a decimal number to a hexadecimal number can be illustrated using the example of (2598.675)10. The number provided by the user is 16. The numerical value provided is 2598. The given numerical value, 16,162, is equivalent to -6. The value of 10 decreased by 2. The expression (A26)16 can be rewritten in an academic manner as the hexadecimal representation of the decimal number 26. The equation 0.67510 = 0.675x16 - 10.8 can be rewritten in an academic manner. The expression "=0.800x16 -- 12.8" can be simplified. The expression "0.800x16 -- 12.8" can be rewritten in an academic manner as "The product of 0.800 multiplied by 16. The expression "=0.800x16 -- 12.8" can be rewritten in an academic manner as "The result of multiplying 0.800 by 16. The given expression, "=0.ACCC16," does not have a clear academic meaning. The decimal number 2598.675 is equivalent to the hexadecimal number A26.ACCC.

16 | 2598

16 | 162 -6

10 -2

$= (A26)_{16}$

0.67510=0.675x16 -- 10.8

=0.800x16 -- 12.8 ↓

=0.800x16 -- 12.8

=0.800x16 -- 12.8

=0.ACCC16

$(2598.675)_{10} = (A26.ACCC)_{16}$

vii) Octal to hexadecimal conversion:

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

binary no. to hexadecimal.

Ex: (756.603)8

7	5	6	.	6	0	3
111	101	110	.	110	000	011
0001	1110	1110	.	1100	0001	1000
1	E	E	.	C	1	8

viii) Hexadecimal to octal conversion:

The process of converting a number from hexadecimal to octal. The provided hexadecimal number should be first converted into binary form, and then, the binary number should be converted into octal form. The hexadecimal number (B9F.AE)16 may be expressed in decimal form as 4735.68359375.
First convert the given hexadecimal no. to binary & then the binary no. to octal.

Ex: (B9F.AE)16

B	9	F	.	A	E		
1011	1001	1111	.	1010	1110		
101	110	011	111	.	101	011	100
5	6	3	7	.	5	3	4

=5637.534

1.4 Error – Detecting codes:

In the transmission and processing of binary data, there exists a vulnerability to noise which has the potential to modify or distort the data. The binary digits '1' in a digital system may be subject to alteration, resulting in a combination of '0' and '1'. This is due to the requirement of precision in digital systems, where any error can potentially present a challenge. Numerous strategies have been developed to identify

the presence of a solitary bit error in a binary word, hence enabling the correction and retransmission of the affected binary word upon detection of such an error.

Parity:

One of the simplest error detection approaches is adding a parity bit to each transmitted word. Odd and even parity bits exist. In odd parity, the transmitter sets the parity bit to '0' or '1' so that the word has an odd number of '1' bits. For even parity, the transmitter sets the parity bit to '0' or '1' to represent an even integer.

Decimal	8421 code	Odd parity	Even parity
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0100	1	0
6	0110	1	0
7	0111	0	1
8	1000	0	1
9	1001	1	0

When the digit data is received , a parity checking circuit generates an error signal if the total no of 1's is even in an odd parity system or odd in an even parity system. This parity check can always detect a single bit error but cannot detect 2 or more errors with in the same word. Odd parity is used more often than even parity does not detect the situation. Where all 0's are created by a short ckt or some other fault condition.

Ex: The concept of an even parity scheme

The given sequence of binary numbers are as follows (a) 10101010 (b) 11110110 (c)10111001

Ans:

(a) No. of 1's in the word is even is 4 so there is no error

(b) No. of 1's in the word is even is 6 so there is no error

(c) No. of 1's in the word is odd is 5 so there is error

Ex: The given sequence of binary numbers are as follows odd parity

(a)10110111 (b) 10011010 (c)11101010

Ans:

(a) No. of 1's in the word is even is 6 so word has error

(b) No. of 1's in the word is even is 4 so word has error

(c) No. of 1's in the word is odd is 5 so there is no error

Checksums:

Checksums are a method used in computer science and telecommunications to verify the integrity of data. The basic parity mechanism is incapable of detecting the occurrence of two faults inside a single word. In order to address this issue, it is recommended to implement a form of two-dimensional parity. The process involves the accumulation of each transmitted word, which is subsequently added to the sum of previously transmitted words. This cumulative sum is then stored at the transmitter's end. Upon the conclusion of the transmission, a mathematical operation known as the checksum is performed to obtain the sum. The transmission was dispatched to the recipient prior to that moment. The recipient has the ability to verify the integrity of the transmitted data by comparing it to the calculated checksum. If the two amounts are same, it can be inferred that no errors were found throughout the reception process. In teleprocessing systems, in the event of an error, the receiving site has the capability to request the retransmission of the complete set of data.

Block parity:

The concept of block parity refers to the technique used in error detection and correction codes to ensure the integrity of data transmission by dividing the

The provided data block is utilized to generate row and column parity bits using the odd parity scheme. The addition of a parity bit, either 0 or 1, is performed in a manner that ensures an odd total number of 1's in each column and row, encompassing both the data bits and the parity bit.

Data	Parity bit
10110	0
10001	1
10101	0
00010	0
11000	1
00000	1
11010	0

data
10110
10001
10101
00010
11000
00000
11010

Error –Correcting Codes:

The code word of an error-correcting code may always be deduced from an incorrect word. A single bit error correcting code must have a minimum distance of three. The code's minimum distance is the fewest bits needed to distinguish any two code phrases. A minimum distance of 3 code can correct single bit mistakes and detect (but not solve) two bit faults. Error correction relies on identifying incorrect digits. Once a mistake is found. Add the complement of the erroneous digit to fix the message using error-correcting codes like Hamming. This coding system appends K parity checking bits (P1, P2,..., Pk) to each group of m data bits. The parity bits are positioned 2^(k-1) from the left, creating a (m+k) bit code word. To fix the problem, k parity checks are performed on particular digits in each code word. Creating an error word locates the incorrect bit, which is then reversed. The k-bit error word is created. by placing either a 0 or a 1 in the location of 2k-1, based on whether the parity check involving the parity bit Pk is met or not. The places of the errors and their respective values are as follows:

Error Position	For 15 bit code C4 C3 C2 C1	For 12 bit code C4 C3 C2 C1	For 7 bit code C3 C2 C1
0	0 0 0 0	0 0 0 0	0 0 0
1	0 0 0 1	0 0 0 1	0 0 1
2	0 0 1 0	0 0 1 0	0 1 0
3	0 0 1 1	0 0 1 1	0 1 1
4	0 1 0 0	0 1 0 0	1 0 0
5	0 1 0 1	0 1 0 1	1 0 1
6	0 1 1 0	0 1 1 0	1 1 0
7	0 1 1 1	0 1 1 1	1 1 1
8	1 0 0 0	1 0 0 0	
9	1 0 0 1	1 0 0 1	
10	1 0 1 0	1 0 1 0	
11	1 0 1 1	1 0 1 1	
12	1 1 0 0	1 1 0 0	
13	1 1 0 1		
14	1 1 1 0		
15	1 1 1 1		

7- bit Hamming code:

In order to send a sequence of four data bits, a 7-bit codeword is generated by appending three parity bits at locations 20, 21, and 22 from the left. This resulting codeword is subsequently transmitted.

The word format

P1	P2	D3	P4	D5	D6	D7

D—Data bits P- Parity bits

Decimal Digit	For BCD P1P2D3P4D5D6D7	For Excess-3 P1P2D3P4D5D6D7
0	0 0 00 0 00	1 0 00 0 11
1	1 1 01 0 01	1 0 01 1 00
2	0 1 01 0 11	0 1 0 0 1 01
3	1 0 00 0 11	1 1 00 1 10
4	1 0 01 1 00	0 0 01 11 1
5	0 1 0 0 1 01	1 1 10 00 0
6	1 1 00 1 10	0 0 11 00 1
7	0 0 01 11 1	1 0 11 0 1 0
8	1 1 10 00 0	0 1 10 0 1 1
9	0 0 11 00 1	0 1 11 1 0 0

Ex: Encode the data bits 1101 into the 7 bit even parity Hamming CodeThe bit pattern is

P1P2D3P4D5D6D7

1 1 0 1

The user has provided a sequence of numbers: 1, 1, 0, 1.

The bits at positions 1, 3, 5, and 7 (P1 111) are required to possess even parity, thus resulting in P1 being assigned a value of 1. Similarly, the bits at positions 2, 3, 6, and 7 (P2 101) are also expected to exhibit even parity, leading to P2 being assigned a value of 0. Furthermore, the bits at positions 4, 5, 6, and 7 (P4 101) are mandated to have even parity, resulting in P4 being assigned a value of 0.

The ultimate code is represented by the binary sequence 1010101. For instance, the specific code word is denoted as 1001001. The bits 1, 3, 5, and 7 (represented as C1 1001) do not contain any errors. To maintain the integrity of the data, a 0 is placed in the 1's position, resulting in C1 being equal to 0. On the other hand, the bits 2, 3, 6, and 7 (represented as C2 0001) contain an error. To rectify this, a 1 is inserted in the 2's position, resulting in C2 being equal to 1. Similarly, the bits 4, 5, 6, and 7

(represented as C4 1001) do not contain any errors. To preserve the accuracy of the data, a 0 is placed in the 4's position, resulting in C3 being equal to 0.

15-bit Hamming Code:

The 15-bit Hamming Code is a coding scheme that is used to transmit a total of 11 data bits together with 4 parity bits. These parity bits are specifically situated at positions 20, 21, 22, and 23 inside the word format.

P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	D12	D13	D14	D15

12-Bit Hamming Code: The 12-bit Hamming code is designed to convey a total of 8 data bits and 4 parity bits. The parity bits are positioned at bit positions 20, 21, 22, and 23. The word format adheres to this structure.

P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	D12

1.5 Complements:

Complements refer to elements in a sentence that provide additional information or complete the meaning of the subject or object. Complements are employed in digital computers to facilitate the subtraction process and enable logical manipulation. In each radix system, two types of complements are utilized. The radix complement, sometimes known as r's complement, is a mathematical concept. The decreased radix complement, often known as the (r-1)'s complement, is a mathematical concept. The topic of discussion is to the representation of signed numbers in binary arithmetic within computer systems.

Two ways of rep signed no.s. The sign-magnitude form refers to a method of representing numbers in which the sign (positive or negative) is indicated separately from the magnitude (absolute value) of the number. The concept of "complemented

form" refers to a certain linguistic structure that involves the addition of a complement to a verb or noun.

There are two sorts of complementation.

- 1's complement form
- 2's complement form

Complementary subtraction reduces hardware.(add ckts instead of addition and subtraction.) Additionally, only adders do subtraction operations. We add the complement of the subtrahend to minuend instead of deleting one. Sign magnitude form adds a sign bit before no. A sign bit of 0 indicates +ve, while a 1 indicates _ve.Ex:

0	1	0	1	0	0	1

↓

Sign bit =+41 magnitude

↑

1	1	0	1	0	0	1

= -41

Note: manipulation is essential to add a +ve no to a –ve no

Representation of signed no.s using 2's or 1's complement method:

When the number is positive, the magnitude is represented in binary and a sign bit 0 is set before the MSB.If the no is -ve, the magnitude is returned in 2's or 1's compliment form and a sign bit 1 is placed before the MSB.Ex:

Given no.	Sign mag form	2's comp form	1's comp form
01101	+13	+13	+13
010111	+23	+23	+23
10111	-7	-7	-8

1101010	-42	-22	-21

- **Special case in 2's comp representation:**

Characteristics of 2's complement no.s:

Properties:

1. One unique zero exists.
2. Comp of 0 is 0.
3. The leftmost bit cannot convey a quantity. +ve 0 no.
4. An n-bit word with the sign bit has 2n+1 +ve integers, 2n-1 -ve integers, and one 0 for a total of 2n distinct states.

The 1s and 0s of the +ve and -ve no.s include important information.

5. To convert a -ve number to a +ve number, determine its 2's comp.

1.6 Signed binary numbers:

Decimal	Sign 2's comp form	Sign 1's comp form	Sign mag form
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0011	0011	0011
+0	0000	0000	0000

-0	--	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101

Whenever a signed no. has a 1 in the sign bit & all 0's for the magnitude bits, the decimal equivalent is -2n , where n is the amount of bits in the magnitude .

Ex: 1000= -8 & 10000=-16

Characteristics of 2's complement no.s:

Methods of obtaining 2's comp of a no:

- In 3 ways

  1. Determine the 1's proportion of the provided number by turning all 0's to 1's and 1's to 0's, then adding 1.

  2. Subtract n bit no N from 2n. 3.Start with the LSB, duplicate all bits up to the first encountered bit, and compliment the remaining bits.

  Ex: Express -45 in 8 bit 2's comp form

  +45 in 8 bit form is 00101101

  I method:

  1's comp of 00101101 & the add 1 00101101

  11010010

  +1

  — – – – – – – – – – –

  11010011 is 2's comp form

  II method:

  Subtract the given no. N from 2^n 2^n = 100000000

  Subtract 45= -00101101

  +1

— – – –

11010011 is 2's comp

III method:

Original no: 00101101

Copy up to First 1 bit 1 Compliment remaining : 1101001

_____

bits 11010011

Ex:

-73.75 in 12 bit 2's comp form I method

01001001.1100

10110110.0011

+1

_____

10110110.0100 is 2's

II method:

2^8 = 100000000.0000

Sub 73.75=-01001001.1100

_____

10110110.0100 is 2's comp

III method :

Orginal no : 01001001.1100 Copy up to 1'st bit 100

Comp the remaining bits: 10110110.0

10110110.0100

2's complement Arithmetic:

- The 2's comp system represents –ve numbers via modulus arithmetic. Computers have predetermined word lengths. A 4-bit number added to another 4-bit number yields a 4-bit number. Any fourth-bit carry will overflow, called modulus arithmetic.

- Ex:1100+1111=1011

  - Add the subtrahend's 2's comp to the minuend in the 2's comp subtraction. If there is a carry out, ignore it and check the sum term's sign bit, MSB.If MSB is 0, result is positive.It is true binary. If the MSB is a carry in or no carry, the outcome is negative.& is 2's comp. Take its 2's comp to obtain its binary magnitude.

Ex: Subtract 14 from 46 using 8 bit 2's comp arithmetic:

+14	= 00001110	
-14	= 11110010	2's comp
+46	= 00101110	
-14	=+11110010	2's comp form of -14
	(1)001000	ignore carry
-32	00	

Ignore carry , The MSB is 0 . so the result is +ve. & is in normal

binary form. So the result is +00100000=+32.

EX: Add -75 to +26 using 8 bit 2's comp arithmetic

+75	=	
-75	01001011	2's comp
	=10110101	
+26	=	
-75	00011010	2's comp form
	=+1011010	of -75
	1	
-49	11001111	No carry

No carry , MSB is a 1, result is _ve & is in 2's comp. The magnitude is 2's comp of11001111. i.e, 00110001 = 49. so result is -49

Ex: add -45.75 to +87.5 using 12 bit arithmetic

+87.5 = 01010111.1000

-45.75=+11010010.0100

-41.75 (1)00101001.1100 ignore carry MSB is 0, result is +ve. =+41.75

1's complement of n number:

- IComplementing each bit of a no yields t, and removing each bit yields 1.The initial number's value rep was enhanced. The rep of zero makes employing 1's comp harder. Both 00000000 and 1's comp 11111111 rep zero.
- The numbers 00000000 and 11111111 are called +ve and –ve zero, respectively..

Ex: -99 & -77.25 in 8 bit 1's comp

+99 = 01100011

-99 = 10011100

+77.25 = 01001101.0100

-77.25 = 10110010.1011

1's complement arithmetic:

Add 1's comp of subtrahend to minuend in 1's comp subtraction. Carryouts should be brought around and added to the LSB as end around carries. Examine the sign bit (MSB). If this is 0, the outcome is positive and true binary. If the MSB is 1 (carry or no carry), the result is –ve and comp.To calculate its binary magnitude, take its 1 comp.

Ex: Subtract 14 from 25 using 8 bit 1's EX: ADD -25 to +14

25	=	00011001	+14	=	00001110
-45	=	11110001	-25	=	+11100110
+11		(1)00001010	-11		11110100

+1

No carry MSB =1

00001011 result=-ve=-1110

MSB is a 0 so result is +ve (binary)

=+1110

1.7 Binary codes

Binary codes are modified original codes represented in binary system.

- Weighted Binary Codes
- Non-weighted codes

Weighted binary codes use positional weighting, where each integer place indicates a weight. Example: binary counting sequence.

Decimal	BCD 8421	Excess-3	84-2-1	2421	5211	Bi-Quinary 5043210			5	0	4	3	2	1	0
0	0000	0011	0000	0000	0000	0100001		0		X					X
1	0001	0100	0111	0001	0001	0100010		1		X				X	
2	0010	0101	0110	0010	0011	0100100		2		X			X		
3	0011	0110	0101	0011	0101	0101000		3		X		X			
4	0100	0111	0100	0100	0111	0110000		4		X	X				
5	0101	1000	1011	1011	1000	1000001		5	X						X
6	0110	1001	1010	1100	1010	1000010		6	X					X	
7	0111	1010	1001	1101	1100	1000100		7	X				X		
8	1000	1011	1000	1110	1110	1001000		8	X			X			
9	1001	1111	1111	1111	1111	1010000		9	X		X				

Reflective Code

When 9 complements 0, 8 and 1, 7 and 2, 6 and 3, 5 and 4 are reflecting. 2421, 5211, and excess-3 reflect, but 8421 does not.

Sequential Codes

Sequential codes are binary numbers that differ by one. It considerably aids data handling mathematically. 8421 and Excess-3 are sequential, while 2421 and 5211 are not..

Non weighted codes

Position-free codes are non-weighted. That is, each binary number location has no set value. Excess-3 code

Excess-3 Code

Excess-3 is a non-weighted decimal code. The code's name comes from each binary code being 8421 plus 0011(3).

Gray Code

Minimum change codes, like the gray code, change only one bit when switching codes. Gray coding is non-weighted because bit positions are unweighted. Gray codes are reflecting digital codes with a one-bit difference between succeeding integers. Also called a unit-distance code. In digital, gray coding is distinctive.

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

Binary to Gray Conversion

- MSB gray coding is binary.
- The XOR of binary code MSB and MSB-1 is gray code MSB-1.
- MSB-2 gray code is XOR of MSB-1 and MSB-2 binary code. MSB-N gray code is XOR of MSB-N-1 and binary code.

8421 BCD code (Natural BCD code):

Nature binary codes encode decimal digits 0-9 in 4 bits. Because of its 8,4,2,1

weights. Sequential, weighted code. It aids math. This code converts decimal to and from, which is useful. It uses more bits than pure binary and is less efficient.

Ex: 14→1110 in binary

But as 0001 0100 in 8421 ode.

The drawback of BCD is that arithmetic operations are more complicated than in binary. The 8421 BCD code system does not include these 6 prohibited combinations: 1010,1011,1100,1101,1110,1111. The drawback of 8421 coding is that binary addition rules only apply to 4 bit groups.

BCD Addition:

From the LSD, it adds each decimal digit in 4-bit binary groupings. If there is no carry and the total term is legal, no adjustment is needed.610(0100) is added to the sum term of a group that has a carry or an illegal code, and the carry is added to the following group.

Ex: Perform decimal additions in 8421 code(a)25+13

In BCD	25= 0010	0101
In BCD	+13 =+0001	0011
	38=0011	1000

No carry , no illegal code .This is the corrected sum

(b). 679.6 + 536.8

679.6	=	0110	0111	1001 .0110 in BCD
+536.8	=	+0101	0011	0010 .1000 in BCD

1216.4	1011	1010	0110	. 1110	illegal codes
	+0110 +	0011	+011	. + 0110	add 0110 to each
			0		
	(1)0001	(1)0000	(1)0101	. (1)0100	propagate carry
	/	/	/	/	
	+1	+1	+1	+1	
	0001	0010	0001	0110	. 0100
	1	2	1	6	. 4

Excess 3 (xs-3)code:

Non-weighted BCD code.Each binary coding word is 8421 plus 0011(3).Sequential codes can perform arithmetic operations. Self-complementing code.Thus, xs-3 code subtracts using compliment addition more directly than 8421 code. Xs-3 has six invalid states: 0000,0010,1101,1110,1111. Its addition and subtraction properties are intriguing..

Excess-3 Addition:

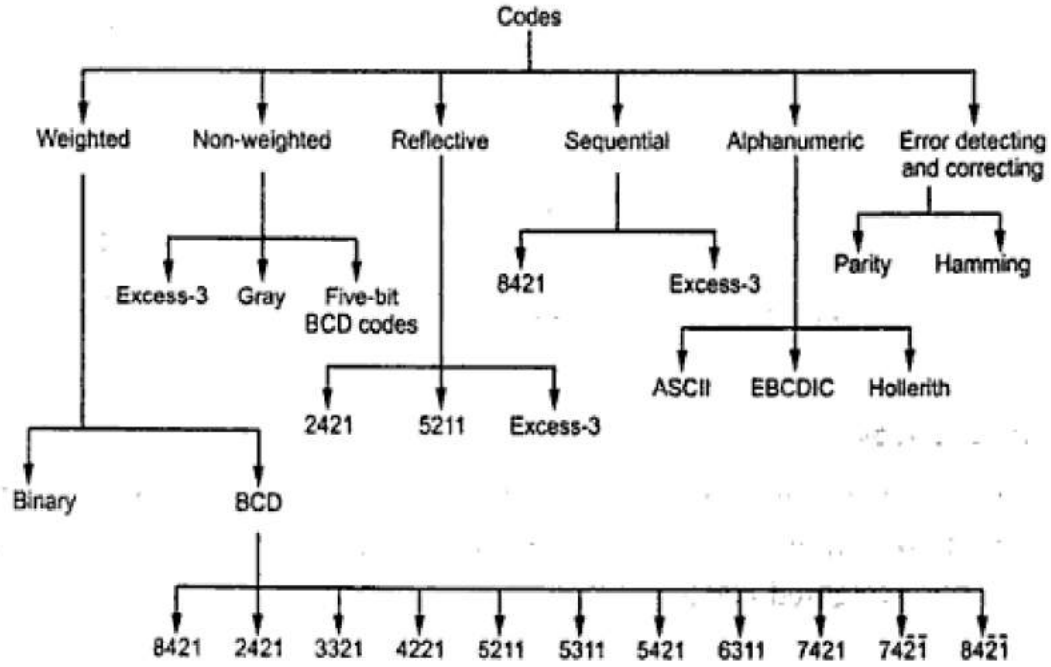
Start with the LSD and add the 4 bit groups in each column to get xs-3. If there is no carry from adding any of the 4-bit groups, subtract 0011 from their sum term (since adding 2 decimal digits in xs-3 without carry yields xs-6). Add 0011 to the sum term of those groups if there is a carry out (invalid states are skipped and the result is regular binary).

EX:	37	0110	1010
	+28	+0101	1011
	———	_ _ _ _ _	_ _ _ _ _
	65	1011	(1)0101 carry

generated

+1propagate carry

1100	0101	add 0011 to correct 0101
		&
-0011	+0011	subtract 0011 to correct 1100



Binary codes block diagram

Alphanumeric Codes:

These codes encode decimal digits and alphabet characteristics. It transfers data between computers and I/O devices like printers, keyboards, and visual displays.ASCII and EBCDIC are popular alphanumeric codes.

1.8 Boolean algebra

Boolean algebra was invented by George Boole in 1854. In 1938, Claude E. Shannon developed switching algebra, a two-valued Boolean algebra that modeled bistable electrical circuits. E. V. Huntington's 1904 postulates will help us define Boolean algebra rigorously..

Boolean algebra is a formal theory of mathematical logic. The algebraic system under consideration comprises a set of components denoted as (0, 1), along with two binary operators referred to as OR and AND, as well as a unary operator known as NOT. The analysis and synthesis of switching circuits heavily rely on a fundamental mathematical technique. Algebraic expression is employed as a means to articulate logical functions.

The axioms and laws of Boolean algebra refer to the fundamental principles and rules that govern the manipulation and properties of Boolean expressions.

Axioms and laws of Boolean algebra

The Axioms or Postulates of Boolean algebra refer to a collection of logical expressions that are accepted as true without requiring proof. These statements serve as a foundation upon which a set of valuable theorems can be constructed.

	AND Operation	OR Operation	NOT Operation
Axiom1 :	0.0=0	0+0=0	$\overline{0}=1$
Axiom2:	0.1=0	0+1=1	$\overline{1}=0$
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

Complementation law

Law1: $\overline{0}=1$ Law3: if A=0,then $\overline{A}=1$

Law2: $\overline{1}=0$ Law4: if A=1,then $\overline{A}=0$

Law5: if $\overline{\overline{A}}=A$ (double

inversion law)

AND Law	OR Law
Law1: A.0=0 (Null law)	Law1: A+0=A
Law2: A.1=A (Identity law)	Law2: A+1=1
Law3: A.A=A (Impotence law)	Law3: A+A=A (Impotence law)
Law4: $A.\overline{A}=0$	Law4: $A+\overline{A}=1$

Basic Theorems and Properties of Boolean algebra Commutative law

Law1: A+B=B+A Law2: A.B=B.A

Associative law

Law1: A + (B +C) = (A +B) +C Law2: A(B.C) = (A.B)C

Distributive law

Law1: A.(B + C) = AB+ AC Law2: A + BC = (A + B).(A +C)

Absorption law

Law1:	A +AB =A	Law2:	A(A +B) = A
Solution :	A(1+B)	Solution :	A.A+A.B
	A		A+A.B
			A(1+B)
			A

DeMorgan Theorems

Theorem1: $\overline{(A+B)} = \overline{A}.\overline{B}$ Theorem2: $\overline{(A.B)} = \overline{A} + \overline{B}$

Redundant Literal Rule

Rule1: $A+\bar{A}B=A+B$

Solution: $A+\bar{A}B$

$(A+A).(A+B)$ ∴ $A + BC = (A + B).(A+C)$ $A+B$ ∴$A+A=1$

Rule2: $A.(\bar{A}+B)=AB$

Solution: $A.(\bar{A}+B)$

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

Consensus Theorem

The Consensus Theorem is a principle in mathematics and logic that states that if two Boolean expressions are equivalent, then their consensus. The first theorem. According to Theorem 2, the equation AB + A'C + BC can be simplified to AB + A'C. The sum of A and B. The equation (A'+C).(B+C) is equivalent to (A+B).(A'+C). The phrase "BC," which stands for "Before Christ," is commonly referred to as the consensus term in academic discourse. However, it is worth noting that this term can be considered redundant due to its inherent nature. The concept of a consensus term arises from a binary expression consisting of two terms, one representing a variable (A) and the other its complement (A'). The consensus term is obtained by multiplying these two terms and excluding the selected variable and its complement.

Theorem1. $AB+ A'C + BC = AB + A'C$ Theorem2. $(A+B). (A'+C).(B+C) =(A+B).(A'+C)$

Consensus Theorem1 Proof:

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

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

$=AB(1+C)+A'C(1+B)$

$= AB+ A'C$

Principle of Duality

The principle of duality refers to a fundamental concept in various fields of study, wherein a pair of complementary elements or concepts are recognized. This principle posits that these elements

Each postulate is composed of two expressions, where one expression can be changed into the other by interchanging the operations of addition (+) and multiplication (·), as well as the identity elements 0 and 1.

These linguistic constructions are commonly referred to as duals of one another.

If a certain equivalence is proven, its dual is also immediately established.

E.g. If we prove: $(x \cdot x)+(x'+x')=1$, then we have by duality: $(x+x)\cdot(x' \cdot x')=0$

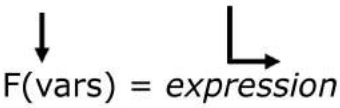
The Huntington postulates were listed in pairs and designated by part (a) and part (b) in belowtable.

Table for Postulates and Theorems of Boolean algebra

Part-A	**Part-B**
A+0=A	A.0=0
A+1=1	A.1=A
A+A=A (Impotence law)	A.A=A (Impotence law)
A+ A'=1	A. A'=0
$\overline{\overline{A}}=A$ (double inversion law)	--
Commutative law: A+B=B+A	A.B=B.A
Associative law: A + (B +C) = (A +B) +C	A(B.C) = (A.B)C
Distributive law: A (B + C) = AB+ AC	A + BC = (A + B) (A +C)
Absorption law: A +AB =A	A(A +B) = A
DeMorgan Theorem: $\overline{(AB)}=\overline{A}\,\overline{B}$	$\overline{(A+B)}=\overline{A}\,\overline{B}$
Redundant Literal Rule: $A+\overline{A}B=A+B$	A (A+B)=AB
Consensus Theorem: AB+ A'C + BC = AB + A'C	(A+B). (A'+C) (B+C) =(A+B).(A'+C)

1.9 Boolean Function

The concept of a Boolean function is a fundamental topic in computer science and mathematics. It refers to a mathematical function that operates on one or more Boolean variables and produces a Boolean. Boolean algebra is a mathematical system that focuses on binary variables and logical processes. A Boolean function, as defined by an algebraic expression, is composed of binary variables, the constants 0 and 1, and logic operation symbols. The function can take on either a value of 1 or 0, depending on the given binary variables.



Set of binary Variables Operators (+, •, ') Constants (0, 1) Groupings (parenthesis) Variables

Consider an example for the Boolean function

F1 = x + y'z

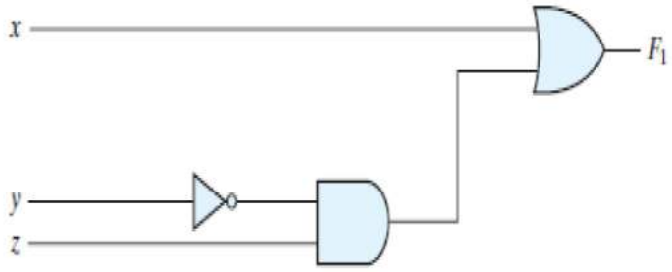
If x or y' and z are 1, F1 is 1. F1 is 0 else. The complement procedure makes y = 0 when y' = 1. Therefore, F1 = 1 if x = 1 or y = 0 and z = 1.

Boolean functions state the logical relationship between binary variables and are evaluated by calculating their binary values for all feasible values.

Truth tables represent Boolean functions. The truth table has 2n rows, where n is the function's variables. Counting from 0 to 2n-1 yields the truth table binary combinations.

Truth Table for F1

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Gate Implementation of F1 = x + y'z

Note:

Q: Let a function F() depend on n variables. How many rows are there in the truth table of F() ? A: 2^n rows, since there are 2^n possible binary patterns/combinations for the n variables.

Truth Tables

- Enumerates all possible combinations of variable values and the corresponding function value
- Truth tables for some arbitrary functions
  F1(x,y,z), F2(x,y,z), and F3(x,y,z) are shown to the below.

x	y	z	F1	F2	F3
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	1

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

- Truth table: a unique representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- Truth tables can be used to prove equality theorems.
- However, the size of a truth table grows exponentially with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

x	y	z	F	G
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Boolean expressions-NOT unique Unlike truth tables, expressions representing a Boolean function are NOT unique.

- Example:
  - $F(x,y,z) = x' \bullet y' \bullet z' + x' \bullet y \bullet z' + x \bullet y \bullet z'$
  - $G(x,y,z) = x' \bullet y' \bullet z' + y \bullet z'$

- The corresponding truth tables for F() and G() are to the right. They are identical.
- Thus, F() = G()

Algebraic Manipulation (Minimization of Boolean function)

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- Example: Simplify F = x′yz + x′yz′ + xz.F= x′yz + x′yz′ + xz

  = x′y(z+z′) + xz

  = x′y•1 + xz

  = x′y + xz

- Example: Prove

  x′y′z′ + x′yz′ + xyz′ = x′z′ + yz′

- **Proof:**

  x′y′z′+ x′yz′+ xyz′

  = x′y′z′ + x′yz′ + x′yz′ + xyz′

  = x′z′(y′+y) + yz′(x′+x)

  = x′z′•1 + yz′•1

  = x′z′ + yz′

Complement of a Function

To find the complement of a function, swap (• and +) and (1 and 0) and complement each variable.

• Change 1s to 0s in the truth table column for F.

• The complement and dual of a function are different. Example: • Determine

G(x,y,z) as the complement of F(x,y,z) = xy'z' + x'yz. G = F' = (xy'z' + x'yz)' / (xy'z')'De Morgan = (x'+y+z) / (x+y+z') DeMorgan again

A function's complement can also be found by identifying its dual and complementing all literals..

1.10 Canonical and Standard Forms

We must investigate formal methods for simplifying Boolean functions. The canonical form of identical functions is identical.

• Minimum and maximum terms • Sum and product of minimum and maximum terms

• Sum and Product terms • SOP and POS

Definitions

Literal: A variable or its complement **Product term:** literals connected by

Sum term: literals connected by +

Minterm: a product term in which all the variables appear exactly once, either complemented or uncomplemented.

Maxterm: a sum term in which all the variables appear exactly once, either complemented or uncomplemented.

Canonical form: Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

Minterm

- ☐ Represents exactly one combination in the truth table.
- ☐ Denoted by mj, where j is the decimal equivalent of the minterm's corresponding binary combination (bj).
- ☐ A variable in mj is complemented if its value in bj is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, bj = 011 and its corresponding minterm is denoted by mj = A'BC

Maxterm

- Represents one truth table combination.
- Mj represents the decimal equivalent of the maxterm's binary combination (bj).
- Mj variables are complemented if bj = 1. Otherwise, they are uncomplemented.

Consider j=3 with 3 variables (A, B, C). Bj = 011 and Mj = A+B'+C' is its maxterm.

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.

Example: Assume 3 variables x,y,z (order is fixed)

x	y	z	Minterm	Maxterm
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

Canonical Forms

- Every function F() has two canonical forms:

- Canonical Sum-Of-Products (sum of minterms)
- Canonical Product-Of-Sums (product of maxterms) Canonical Sum-Of-Products:

The minterms included are those m_j such that F( ) = 1 in row j of the truth table for F( ). Canonical Product-Of-Sums:

The maxterms included are those M_j such that F( ) = 0 in row j of the truth table for F( ).

a	b	c	f_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Example

Consider a Truth table for f1(a,b,c) at right The canonical sum-of-products form for f1 is $f1(a,b,c) = m1 + m2 + m4 + m6$

$= a'b'c + a'bc' + ab'c' + abc'$

The canonical product-of-sums form for f1 is $f1(a,b,c) = M0 \bullet M3 \bullet M5 \bullet M7$

$= (a+b+c)\bullet(a+b'+c')\bullet (a'+b+c')\bullet(a'+b'+c')$.

- Observe that: $m_j = M_j'$
-

Conversion between Canonical Forms

- Replace ∑ with ∏ (or *vice versa*) and replace those *j*'s that appeared in the original form with those that do not.
    - Example:

$$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$$

$$= m_1 + m_2 + m_4 + m_6$$

$$= \Sigma(1,2,4,6)$$

$$= \Pi(0,3,5,7)$$

$$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c')$$

Standard Forms

Boolean functions can also be expressed standardly. In this configuration, function terms can have one, two, or more literals.

Sum of products and products of sums are conventional forms.

Boolean expressions with AND terms, called product terms, and literals make up the total of products. Sum represents ORing these phrases. Example of a sum of products function: F1 = y' + xy + x'yz'

It has three product terms with one, two, and three literals. The sum is an OR operation. Sum products are Boolean expressions with OR terms, termed sum terms. Terms can have any number of literals. The product means ANDing these terms. An example of a sum product function is F2 = x(y' + z)(x' + y + z').

One, two, and three literals make up this expression's sum terms. The product is AND.

1.11 Digital Logic Gates

Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR		$F = x + y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inverter		$F = x'$	$x \mid F$ $0 \mid 1$ $1 \mid 0$
Buffer		$F = x$	$x \mid F$ $0 \mid 0$ $1 \mid 1$
NAND		$F = (xy)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR		$F = (x + y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

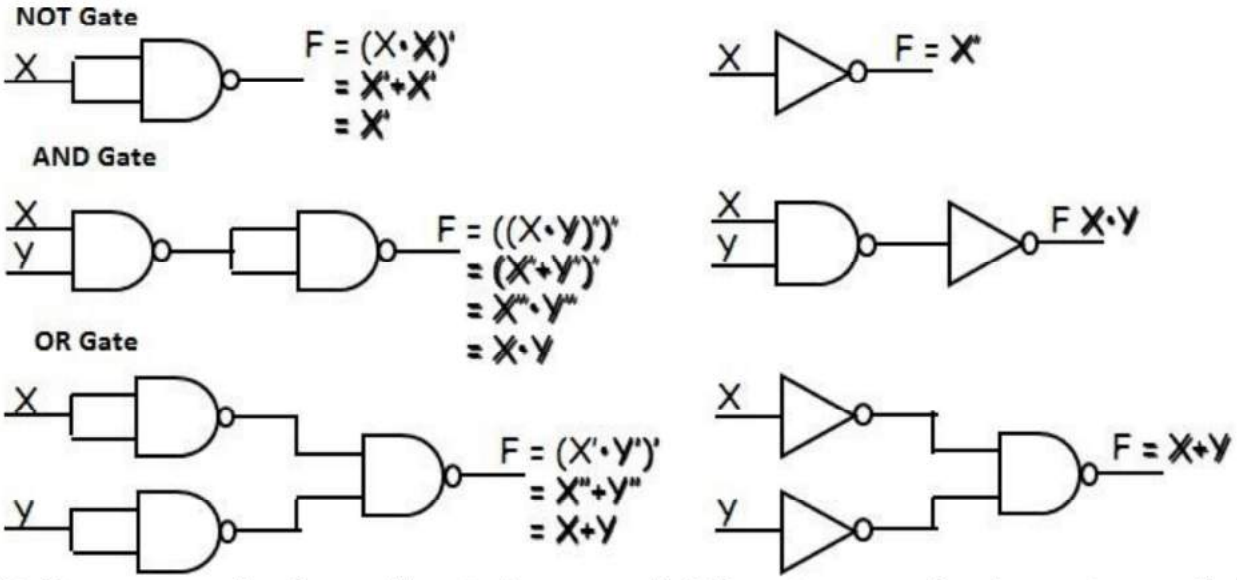
Properties of XOR Gates

1. XOR is commutative.
    - This means, a^b = b^a.
2. XOR is associative.
    - This means, a^(b^c) = (a^b)^c = (a^c)^b.
3. XOR is self-inverse.
    - This means, any number XOR'ed with itself evaluates to 0.
    - a^a = 0.
4. 0 is the identity element for XOR.
    - This means, any number XOR'ed with 0 remains unchanged.
    - a^0 = a.

Universal Logic Gates

NAND and NOR are universal gates. All fundamental gates (NOT, AND, OR) can be implemented with NAND or NOR gates. A universal gate gives logic designers flexibility and huge benefits.

NAND as a Universal Gate



NAND Known as a "universal" gate because NAND gates may implement any digital circuit.Proving that only NAND gates can implement AND, OR, and NOT proves the above.