UNIT - 1
NUMBER SYSTEMS & BOOLEAN ALGEBRA

- Introduction about digital system
- Philosophy of number systems
- Complement representation of negative numbers
- Binary arithmetic
- Binary codes
- Error detecting & error correcting codes
- Hamming codes

INTRODUCTION ABOUT DIGITAL SYSTEM

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

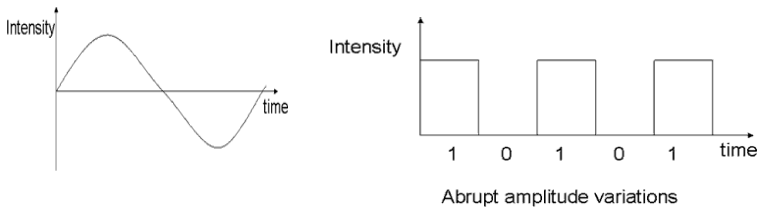
Characteristics of Digital systems

- Digital systems manipulate discrete elements of information.
- Discrete elements are nothing but the digits such as 10 decimal digits or 26 letters of alphabetsand so on.
- Digital systems use physical quantities called signals to represent discrete elements.
- In digital systems, the signals have two discrete values and are therefore said to be binary.
- A signal in digital system represents one binary digit called a bit. The bit has a value either 0 or 1.

Analog systems vs Digital systems

Analog system process information that varies continuously i.e; they process time varying signals that can take on any values across a continuous range of voltage, current or any physical parameter.

Digital systems use digital circuits that can process digital signals which can take either 0 or 1 for binary system.



Abrupt amplitude variations

Advantages of Digital system over Analog system

1. Ease of programmability

The digital systems can be used for different applications by simply changing the program without additional changes in hardware.

2. Reduction in cost of hardware

The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.

3. gh speed

Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.

4. High Reliability

Digital systems are highly reliable one of the reasons for that is use of error correction codes.

5. Design is easy

The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.

6. Result can be reproduced easily

Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

Disadvantages of Digital Systems

- Use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well.
- Digital circuits are often fragile, in that if a single piece of digital data is lost or misinterpretedthe meaning of large blocks of related data can completely change.
- Digital computer manipulates discrete elements of information by means of a binary code.
- Quantization error during analog signal sampling.

NUMBER SYSTEM

Number system is a basis for counting varies items. Modern computers communicate and operate with binary numbers which use only the digits 0 &1. Basic number system used by humans is Decimal number system.

For Ex: Let us consider decimal number 18. This number is represented in binary as 10010.

We observe that binary number system take more digits to represent the decimal number. For large numbers we have to deal with very large binary strings. So this fact gave rise to three new number systems.

i) Octal number systems

ii) Hexa Decimal number system

iii) Binary Coded Decimal number(BCD) system

To define any number system we have to specify

- Base of the number system such as 2,8,10 or 16.
- The base decides the total number of digits available in that number system.
- First digit in the number system is always zero and last digit in the number system is always base-1.

Binary number system:

The binary number has a radix of 2. As r = 2, only two digits are needed, and these are 0 and 1. In binary system weight is expressed as power of 2.

- Example:

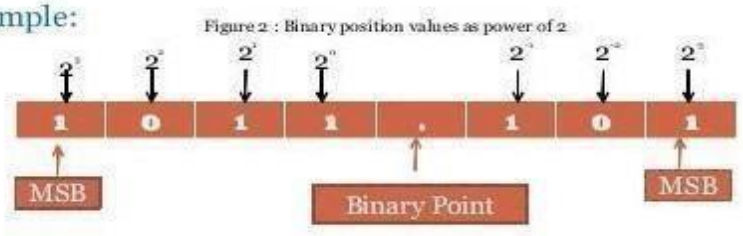


Figure 2 : Binary position values as power of 2

The left most bit, which has the greatest weight is called the Most Significant Bit (MSB). And the right most bit which has the least weight is called Least Significant Bit (LSB).

For Ex: $1001.01_2 = [(1) \times 2^3] + [(0) \times 2^2] + [(0) \times 2^1] + [(1) \times 2^0] + [(0) \times 2^{-1}] + [(1) \times 2^2]$

$1001.01_2 = [1 \times 8] + [0 \times 4] + [0 \times 2] + [1 \times 1] + [0 \times 0.5] + [1 \times 0.25]$

$1001.01_2 = 9.25_{10}$

Decimal Number system

The decimal system has ten symbols: 0,1,2,3,4,5,6,7,8,9. In other words, it has a base of 10.

Octal Number System

Digital systems operate only on binary numbers. Since binary numbers are often very long, two shorthand notations, octal and hexadecimal, are used for representing large binary numbers. Octal systems use a base or radix of 8. It uses first eight digits of decimal number system. Thus it has digits from 0 to 7.

Hexa Decimal Number System

The hexadecimal numbering system has a base of 16. There are 16 symbols. The decimal digits 0 to 9 are used as the first ten digits as in the decimal system, followed by the letters A, B, C, D, E and F, which represent the values 10, 11,12,13,14 and 15 respectively.

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

Number Base conversions

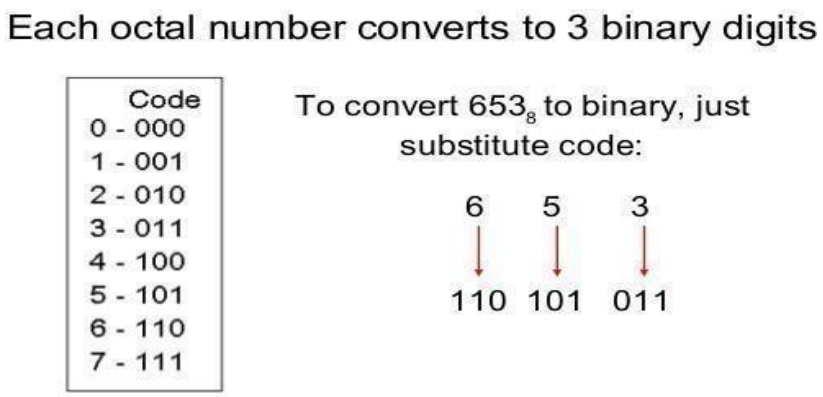
The human beings use decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number system into its equivalent binary.

i) Binary to octal number conversion
ii) Binary to hexa decimal number conversion

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

iii) Octal to binary Conversion



iv) Hexa to binary conversion

v) Octal to Decimal conversion

Ex: convert 4057.06_8 to octal

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

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

$=2095.0937_{10}$

vi) Decimal to Octal Conversion

Ex: convert 378.93_{10} tooctal

378_{10} to octal: Successive division:

8 | 378
|_____
8 |47 --- 2
|_____
8 |5 --- 7 ↑
|_____
0 --- 5

$=572_8$

0.93_{10} to octal :

0.93x8=7.44
0.44x8=3.52 ↓
0.53x8=4.16
0.16x8=1.28

$=0.7341_8$

$378.93_{10}=572.7341_8$

vii) Hexadecimal to Decimal Conversion

Ex: $5C7_{16}$ to decimal

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

$=1280+192+7$

$=147_{10}$

viii) Decimal to Hexadecimal Conversion

Ex: 2598.67510

1 6 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

$=0.ACCC_{16}$

$2598.675_{10} = A26.ACCC_{16}$

ix) 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

x) Hexadecimal to octal conversion:

First convert the given hexadecimal no. to binary & then the binary no. to octal.

Ex: B9F.AE16

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

Complements:

In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.

i) The radix complement or r's complement
ii) The diminished radix complement or (r-1)'s complement

Representation of signed no.s binary arithmetic in computers:

- Two ways of rep signed no.s
    1. Sign Magnitude form
    2. Complemented form
- Two complimented forms
    1. 1‘s compliment form
    2. 2‘s compliment form

Advantage of performing subtraction by the compliment method is reduction in the hardware.(instead of addition & subtraction only adding ckt‘s are needed.)
i. e, subtraction is also performed by adders only.

Instead of subtracting one no. from other the compliment of the subtrahend is added to minuend. In sign magnitude form, an additional bit called the sign bit is placed in front of the no. If the sign bit is 0, the no. is +ve, If it is a 1, the no is _ve.

Ex:

0	1	0	1	0	0	1

↓

Sign bit =+41 magnitude

↑

1	1	0	1	0	0	1

= -41

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

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

If the no. is +ve, the magnitude is rep in its true binary form & a sign bit 0 is placed in front of the MSB.I f the no is _ve , the magnitude is rep in its 2‘s or 1‘s compliment form &a sign bit 1 is placed in front of 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:

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

Ex: 1000= -8 & 10000=-16

Characteristics of 2's compliment no.s:

Properties:

1. There is one unique zero
2. 2's comp of 0 is 0
3. The leftmost bit can't be used to express a quantity . it is a 0 no. is +ve.
4. For an n-bit word which includes the sign bit there are $(2^{n-1}-1)$ +ve integers, 2^{n-1} –ve integers & one 0 , for a total of 2^nuniquestates.
5. Significant information is containd in the 1's of the +ve no.s & 0's of the _ve no.s
6. A _ve no. may be converted into a +ve no. by finding its 2's comp.

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
-6	1010	1001	1110
-7	1001	1000	1111
8	1000	--	--

Methods of obtaining 2's comp of a no:

- In 3 ways
  1. Byobtaining the 1's comp of the given no. (by changing all 0's to 1's & 1's to 0's) & then adding 1.
  2. By subtracting the given n bit no N from 2^n
  3. Starting at the LSB , copying down each bit upto & including the first 1 bit encountered , and complimenting 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

Copyup to First 1 bit 1

Compliment remaining : 1101001

———————

bits 11010011

Ex:

-73.75 in 12 bit 2‘compform

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 :

Orginalno : 01001001.1100

Copy up to 1‘st bit 100

Comp the remaining bits: 10110110.0

_____

10110110.0100

2’s compliment Arithmetic:

- The 2‘s comp system is used to rep –ve no.s using modulus arithmetic . The word length of a computer is fixed. i.e, if a 4 bit no. is added to another 4 bit no . the result will be only of 4 bits. Carry if any , from the fourth bit will overflow called the Modulus arithmetic. Ex:1100+1111=1011
- In the 2‘s compl subtraction, add the 2‘s comp of the subtrahend to the minuend . If there is a carry out , ignore it , look at the sign bit I,e, MSB of the sum term .If the MSB is a 0, the result is positive.& it is in true binary form. If the MSB is a ` (carry in or no carry at all) the result is negative.& is in its 2‘s comp form. Take its 2‘s comp to find its magnitude in binary.

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

___ _____

-32 (1)00100000 ignore carry

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 = 01001011
-75 =10110101 2‘s comp

+26 = 00011010
-75 =+10110101 2‘s comp form of -75

-49 11001111 No carry

No carry , MSB is a 1, result is _ve & is in 2‘s comp. The magnitude is 2‘s comp of 11001111. 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 compliment of n number:

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

Ex: -99 & -77.25 in 8 bit 1‘s comp

+99 = 01100011
-99 = 10011100

+77.25 = 01001101.0100
-77.25 = 10110010.1011

1’s compliment arithmetic:

In 1‘s comp subtraction, add the 1‘s comp of the subtrahend to the minuend. If there is a carryout , bring the carry around & add it to the LSB called the **end around carry.** Look at the sign bit (MSB) . If this is a 0, the result is +ve & is in true binary. If the MSB is a 1 (carry or no carry), the result is –ve & is in its is comp form .Take its 1‘s comp to get the magnitude inn binary.

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=-11_{10}

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

=+11_{10}

Binary codes

Binary codes are codes which are represented in binary system with modification from the original ones.

- Weighted Binary codes
- Non Weighted Codes

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	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

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.

Non weighted codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: Excess-3 code

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).

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	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

- 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.

8421 BCD code (Natural BCD code):

Each decimal digit 0 through 9 is coded by a 4 bit binary no. called natural binary codes. Because of the 8,4,2,1 weights attached to it. It is a weighted code & also sequential . it is useful for mathematical operations. The advantage of this code is its case of conversion to & from decimal. It is less efficient than the pure binary, it require more bits.

Ex: 14→1110 in binary

But as 0001 0100 in 8421 ode.

The disadvantage of the BCD code is that , arithmetic operations are more complex than they are in pure binary . There are 6 illegal combinations 1010,1011,1100,1101,1110,1111 in these codes, they are not part of the 8421 BCD code system . The disadvantage of 8421 code is, the rules of binary addition 8421 no, but only to the individual 4 bit groups.

BCD Addition:

It is individually adding the corresponding digits of the decimal no,s expressed in 4 bit binary groups starting from the LSD . If there is no carry & the sum term is not an illegal code , no correction is needed .If there is a carry out of one group to the next group or if the sum term is an illegal code then 6_{10}(0100) is added to the sum term of that group & the resulting carry is added to the next group.

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 = 011(0111 1001 .0110 in BCD
+536.8 = +0101 0011 0010 .1000 in BCD

1216.4 1011 1010 0110 . 1110 illegal codes
+0110 + 0011 +0110 . + 0110 add 0110 to each

(1)0001 (1)0000 (1)0101 . (1)0100 propagate carry
/ / / /
+1 +1 +1 +1

0001 0010 0001 0110 . 0100

1 2 1 6 . 4

BCD Subtraction:

Performed by subtracting the digits of each 4 bit group of the subtrahend the digits from the corresponding 4- bit group of the minuend in binary starting from the LSD . if there is no borrow from the next group , then 6_{10}(0110)is subtracted from the difference term of this group.

(a)38-15

In BCD 38= 0011 1000
In BCD -15 = -0001 0101

23 0010 0011

No borrow, so correct difference.

.(b) 206.7-147.8

206.7 = 0010 0000 0110 . 0111 in BCD
-147.8 = -0001 0100 0111 . 0110 in BCD

58.9 0000 1011 1110 . 1111 borrows are present
-0110 -0110 . -0110 subtract 0110

0101 1000 . 1001

BCD Subtraction using 9's & 10's compliment methods:

Form the 9's & 10's compliment of the decimal subtrahend & encode that no. in the 8421 code . the resulting BCD no.s are then added.

EX: 305.5 – 168.8

305.5 = 305.5
-168.8= +83.1 9's comp of -168.8

(1)136.6
+1 end around carry
136.7 corrected difference

305.5_{10} = 0011 0000 0101 . 0101
$+831.1_{10}$ = +1000 0011 0001 . 0001 9's comp of $168._{10}$ in BCD
+1011 0011 0110 . 0110 1011 is illegal code
+0110 add 0110

(1)0001 0011 0110 . 0110
+1 End around carry

0001 0011 0110 . 0111
= 136.7

Excess three(xs-3)code:

It is a non-weighted BCD code .Each binary codeword is the corresponding 8421 codeword plus 0011(3).It is a sequential code & therefore , can be used for arithmetic operations..It is a self-complementing code.s o the subtraction by the method of compliment addition is more direct in xs-3 code than that in 8421 code. The xs-3 code has six invalid states 0000,0010,1101,1110,1111.. It has interesting properties when used in addition & subtraction.

Excess-3 Addition:

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

EX: 37 0110 1010
+28 +0101 1011

65 1011 (1)0101 carry generated
+1 propagate carry

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

1001 1000 $=65_{10}$

Excess -3 (XS-3) Subtraction:

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

Ex: 267-175

267 = 0101 1001 1010
-175= -0100 1010 1000

0000 1111 0010
+0011 -0011 +0011

0011 1100 +0011 $=92_{10}$

Xs-3 subtraction using 9's & 10's compliment methods:

Subtraction is performed by the 9's compliment or 10's compliment

Ex:687-348 The subtrahend (348) xs -3 code & its compliment are:

9's comp of 348 = 651

Xs-3 code of 348 = 0110 0111 1011

1's comp of 348 in xs-3 = 1001 1000 0100

Xs=3 code of 348 in xs=3 = 1001 1000 0100

687 687

-348 → +651 9's compl of 348

339 (1)338

+1 end around carry

339 corrected difference in decimal

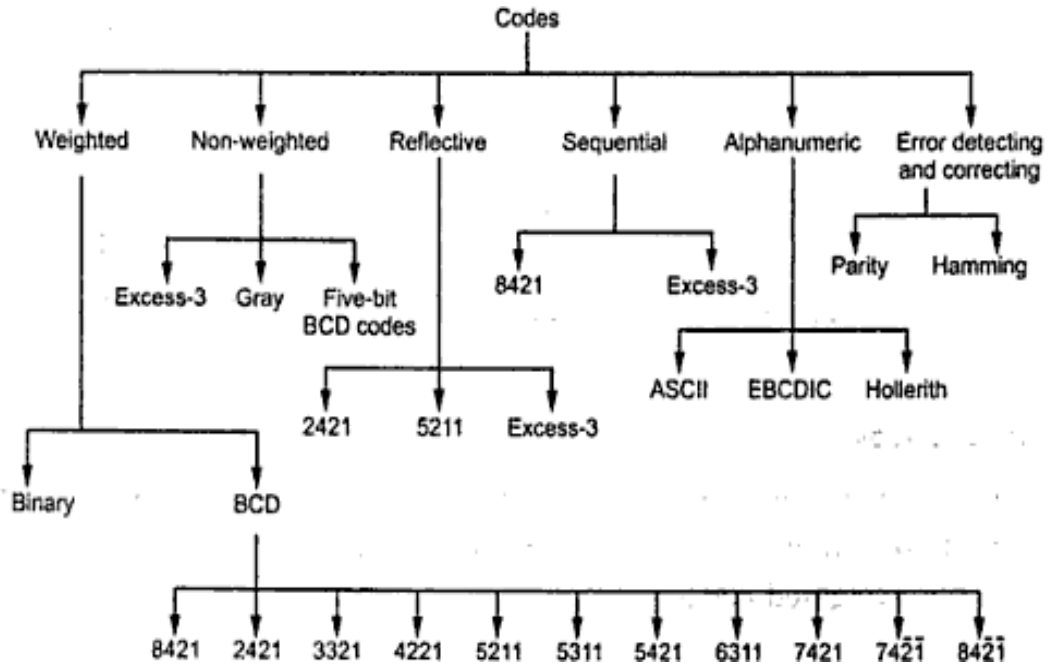
1001	1011	1010	687 in xs-3
+1001	1000	0100	1's comp 348 in xs-3
(1)0010	(1)0011	1110	carry generated
+1	+1		propagate carry
(1)0011	0010	1110	
		+1	end around carry
0011	0011	1111	(correct 1111 by sub0011 and
+0011	+0011	+0011	correct both groups of 0011 by
			adding 0011)
0110	0110	1100	corrected diff in xs-3 = 330_{10}

The Gray code (reflective –code):

Gray code is a non-weighted code & is not suitable for arithmetic operations. It is not a BCD code . It is a cyclic code because successive code words in this code differ in one bit position only i.e, it is a unit distance code.Popular of the unit distance code.It is also a reflective code i.e,both reflective & unit distance. The n least significant bits for 2^n through 2^{n+1}-1 are the mirror images of thosr for 0 through 2^n-1.An N bit gray code can be obtained by reflecting an N- 1 bit code about an axis at the end of the code, & putting the MSB of 0 above the axis & the MSB of 1 below the axis.

Reflection of graycodes:

Gray Code				Decimal	4 bit binary
1 bit	2 bit	3 bit	4 bit		
0	00	000	0000	0	0000
1	01	001	0001	1	0001
	11	011	0011	2	0010
	10	010	0010	3	0011
		110	0110	4	0100
		111	0111	5	0101
		101	0101	6	0110
		110	0100	7	0111
			1100	8	1000
			1101	9	1001
			1111	10	1010
			1110	11	1011
			1010	12	1100
			1011	13	1101
			1001	14	1110
			1000	15	1111



Binary codes block diagram

Error – Detecting codes: When binary data is transmitted & processed,it is susceptible to noise that can alter or distort its contents. The 1‘s may get changed to 0‘s & 1‘s .because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

Parity: The simplest techniques for detecting errors is that of adding an extra bit known as parity bit to each word being transmitted.Two types of parity**:** Oddparity**,** evenparity forodd parity, the parity bit is set to a _0‘ or a _1‘ at the transmitter such that the total no. of 1 bit in the word including the parity bit is an odd no.For even parity, the parity bit is set to a _0‘ or a _1‘ at the transmitter such that the parity bit is an even no.

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: Even parity scheme

(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: 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:

Simple parity can‘t detect two errors within the same word. To overcome this, use a sort of 2 dimensional parity. As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end. At the end of transmission, the sum called the check sum. Up to that time sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same, then no errors were detected at the receiver end. If there is an error, the receiving location can ask for retransmission of the entire data, used in teleprocessing systems.

Block parity:

Block of data shown is create the row & column parity bits for the data using odd parity. The parity bit 0 or 1 is added column wise & row wise such that the total no. of 1‘s in each column & row including the data bits & parity bit is odd as

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:

A code is said to be an error –correcting code, if the code word can always be deduced from an erroneous word. For a code to be a single bit error correcting code, the minimum distance of that code must be three. The minimum distance of that code is the smallest no. of bits by which any two code words must differ. A code with minimum distance of 3 can‘t onlycorrect single bit errors but also detect (can‘t correct) two bit errors, The key to error correction is that it must be possible to detect & locate erroneous that it must be possible to detect & locate erroneous digits. If the location of an error has been determined. Then by complementing the erroneous digit, the message can be corrected , error correcting , code is the Hamming code , In this , to each group of m information or message or data bits, K parity checking bits denoted by P1,P2, ---------pk located at positions 2^{k-1} from left are added to form an (m+k) bit code word. To correct the error, k parity checks are performed on selected digits of each code word, & the position of the error bit is located by forming an error word, & the error bit is then complemented. The k bit error word is generated by putting a 0 or a 1 in the 2^{k-1}th position depending upon whether the check for parity involving the parity bit P_k is satisfied or not.Error positions & their corresponding values :

Error Position	For 15 bit code $C_4 C_3 C_2 C_1$	For 12 bit code $C_4 C_3 C_2 C_1$	For 7 bit code $C_3 C_2 C_1$
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:

To transmit four data bits, 3 parity bits located at positions 2^0 21&2^2 from left are added to make a 7 bit codeword which is then transmitted.

The word format

P_1	P_2	D_3	P_4	D_5	D_6	D_7

D—Data bits P-
Parity bits

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

Ex: Encode the data bits 1101 into the 7 bit even parity Hamming Code

The bit pattern is

$P_1P_2D_3P_4D_5D_6D_7$

1 1 0 1

Bits 1,3,5,7 (P_1 111) must have even parity, so P_1 =1

Bits 2, 3, 6, 7(P_2 101) must have even parity, so P_2 =0

Bits 4,5,6,7 (P_4 101)must have even parity, so P_4 =0

The final code is 1010101

EX: Code word is 1001001

Bits 1,3,5,7 (C_1 1001) →no error →put a 0 in the 1's position→C1=0

Bits 2, 3, 6, 7(C_2 0001)) → error →put a 1 in the 2's position→C2=1

Bits 4,5,6,7 (C_4 1001)) →no error →put a 0 in the 4's position→C3=0

15-bit Hamming Code: It transmit 11 data bits, 4 parity bits located 2^0 2^1 2^2 2^3

Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12	D13	D14	D15

12-Bit Hamming Code: It transmit 8 data bits, 4 parity bits located at position 2^0 2^1 2^2 2^3

Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12

Alphanumeric Codes:

These codes are used to encode the characteristics of alphabet in addition to the decimal digits. It is used for transmitting data between computers & its I/O device such as printers, keyboards & video display terminals. Popular modern alphanumeric codes are ASCII code & EBCDIC code.

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

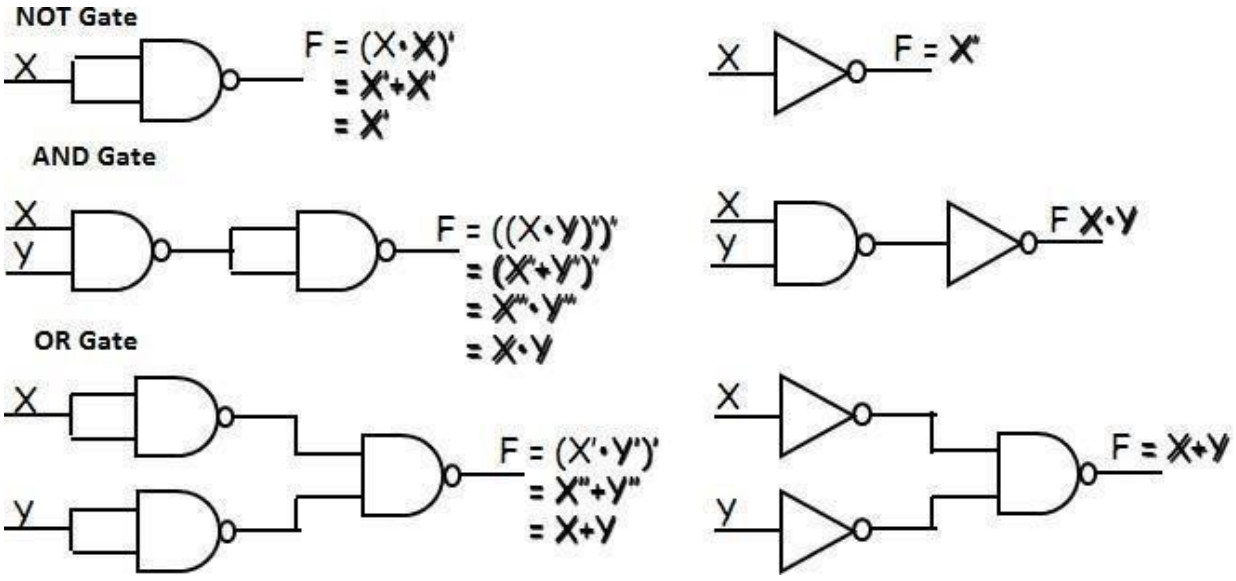
- XOR (also ⊕) : the "not-equal" function
- XOR(X,Y) = X ⊕ Y = X'Y + XY'
- Identities:
  - X ⊕ 0 = X
  - X ⊕ 1 = X'
  - X ⊕ X = 0
  - X ⊕ X' = 1
- Properties:
  - X ⊕ Y = Y ⊕ X
  - (X ⊕ Y) ⊕ W = X ⊕ (Y ⊕W)

Universal Logic Gates

NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

NAND as a Universal Gate

NAND Known as a "universal" gate because ANY digital circuit can be implemented with NAND gates alone.
To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.



Boolean Algebra: In 1854, George Boole developed an algebraic system now called Boolean algebra. In 1938, Claude E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0, 1), two binary operators called OR, AND, and one unary operator NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits. It is a way to express logic functions algebraically.

Boolean algebra, like any other deductive mathematical system, may be defined with aset of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is any collection of objects having a common property. If **S** is a set and ***x*** and ***y*** are certain objects, then **x Î Sdenotes** that ***x*** is a member of the set **S**, and ***y*** Ï**S** denotes that ***y*** is not an element of **S**. A set with adenumerable number of elements is specified by braces: **A** = {1,2,3,4}, *i.e.* the elements of set **A** are thenumbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair ofelements from S a unique element from S._ Example: In *a***b*=*c*, we say that * is a binary operator if it specifies a rule for finding *c* from the pair (*a*,*b*)and also if *a*, *b*, *c* Î S.

Axioms and laws of Boolean algebra

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

	AND Operation	OR Operation	NOT Operation
Axiom1 :	0.0=0	0+0=0	0=1
Axiom2:	0.1=0	0+1=1	1=0
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

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)

CLOSURE: The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values,it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

_ A set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.

_ For example, the set of natural numbers N = {1, 2, 3, 4, … 9} is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any *a*, *b* Î N we obtain a unique *c* Î N by the operation *a* + *b* = c.

ASSOCIATIVE LAW:

A binary operator * on a set *S* is said to be associative whenever $(x * y) * z = x * (y * z)$ for all $x, y, z \hat{I}$ S, forall Boolean values x, y and z.

COMMUTATIVE LAW:

A binary operator * on a set *S* is said to be commutative whenever $x * y = y * x$ for all $x, y, z \epsilon$ S

IDENTITY ELEMENT:

A set ***S*** is said to have an identity element with respect to a binary operation * on S if there exists an element $e \epsilon$ S with the property $e * x = x * e = x$ for every $x \epsilon$ S

BASIC IDENTITIES OF BOOLEAN ALGEBRA

- *Postulate 1(Definition)*: A Boolean algebra is a closed algebraic system containing a set *K* of two or more elements and the two operators · and + which refer to logical AND and logical OR •$x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = 1$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = x$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x (yz) = (xy) z$
- $x (y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x' y'$
- $(xy)' = x' + y'$

- *(x')' = x*

DeMorgan's Theorem

(a) $(a + b)' = a'b'$

(b) $(ab)' = a' + b'$

Generalized DeMorgan's Theorem

(a) $(a + b + \ldots z)' = a'b' \ldots z'$

(b) $(a.b \ldots z)' = a' + b' + \ldots z'$

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

Solution: A(1+B)
A

Law2: A(A +B) = A

Solution: A.A+A.B
A+A.B
A(1+B)
A

Consensus Theorem

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

The BC term is called the consensus term and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (A) and its complement (A') are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement

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

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations (+) and (·) as well as the identity elements 0 and 1. Such expressions are known as duals of each other.

If some equivalence is proved, then its dual is also immediately true.

If we prove: (x.x)+(x'+x')=1, then we have by duality: (x+x)·(x'.x')=0

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

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)
$A+\bar{A}=1$	$A.\bar{A}=0$
$\bar{\bar{A}}=A$ (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: $\overline{(A+B)}=\bar{A}\bar{B}$	$\overline{AB}=\bar{A}+\bar{B}$
Redundant Literal Rule: $A+\bar{A}B=A+B$	$A.(\bar{A}+B)=AB$
Consensus Theorem: AB+ A'C + BC = AB + A'C	(A+B). (A'+C).(B+C) =(A+B).(A'+C)

Boolean Function

Boolean algebra is an algebra that deals with binary variables and logic operations.

A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.

For a given value of the binary variables, the function can be equal to either 1 or 0.

F(vars) = *expression*

Set of binary Variables

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

Consider an example for the Boolean function

F1 = x + y'z

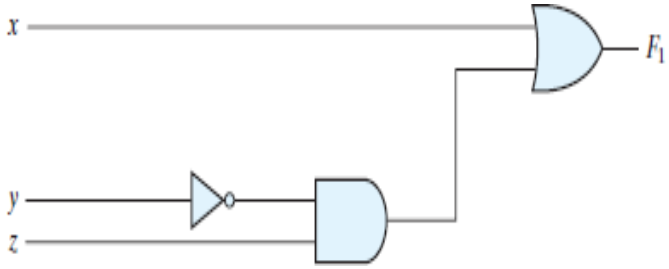
The function F1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F1 is equal to 0 otherwise. The complement operation dictates that when y' = 1, y = 0. Therefore, F1 = 1 if x = 1 or if y = 0 and z = 1.

A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n, where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

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	F_1	F_2	F_3
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 ofvariables involved, hence unwieldy. This motivates the use of Boolean Algebra.

Boolean expressions-NOT unique

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

- Example:
  - F(x,y,z) = x'•y'•z' + x'•y•z' + x•y•z'
  - G(x,y,z) = x'•y'•z' + y•z'
- The corresponding truth tables for F() and G() are to the right. Theyare identical.
- Thus, F() = G()

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

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

- The complement of a function is derived by interchanging (• and +), and (1 and 0),and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The *complement* of a function IS NOT THE SAME as the *dual* of afunction.

Example

- Find G(x,y,z), the complement of F(x,y,z) = xy'z' +x'yz
  Ans: G = F' = (xy'z' + x'yz)'
  = (xy'z')' • (x'yz)' *DeMorgan*
  = (x'+y+z) • (x+y'+z') *DeMorgan* again

Note: The complement of a function can also be derived by finding the function's *dual,* and then complementing all of the literals

Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.
Identical functions will have exactly the same canonical form.

- Minterms and Maxterms
- Sum-of-Minterms and Product-of- Maxterms
- Product and Sum terms
- Sum-of-Products (SOP) and Product-of-Sums (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 m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.

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

Maxterm

- Represents exactly one combination in the truth table.
- Denoted by M_j, where *j* is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and *j*=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truthtable.

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

x	y	z	Minterm	Maxterm
0	0	0	x'y'z' = m_0	x+y+z = M_0
0	0	1	x'y'z = m_1	x+y+z' = M_1
0	1	0	x'yz' = m_2	x+y'+z = M_2
0	1	1	x'yz = m_3	x+y'+z'= M_3
1	0	0	xy'z' = m_4	x'+y+z = M_4
1	0	1	xy'z = m_5	x'+y+z' = M_5
1	1	0	xyz' = m_6	x'+y'+z = M_6
1	1	1	xyz = m_7	x'+y'+z' = M_7

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( ).

Example

Consider a Truth table for $f_1(a,b,c)$ at right

The canonical sum-of-products form for f_1 is

$f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

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

The canonical product-of-sums form for f_1 is

$f_1(a,b,c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$

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

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

- Observe that: $m_j = M_j'$

Shorthand: ∑ and ∏

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where ∑ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where ∏ indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$
-

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:

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

$= m_1 + m_2 + m_4 + m_6$

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

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

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

Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

F1 = y' + xy + x'yz'

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

F2 = x(y' + z)(x' + y + z')

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

Conversion of SOP from standard to canonical form

Example-1.

Express the Boolean function F = A + B'C as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

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

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

The second term B'C is missing one variable; hence,

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

Combining all terms, we have

F = A + B'C

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

But AB'C appears twice, and according to theorem (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

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

= m1 + m4 + m5 + m6 + m7

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

F(A, B, C) = ∑m (1, 4, 5, 6, 7)

Example-2.

Express the Boolean function F = xy + x'z as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

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

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

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

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

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

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

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

Combining all the terms and removing those which appear more than once, we finally obtain

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

F= M0M2M4M5

A convenient way to express this function is as follows: F(x, y, z) = πM(0, 2, 4, 5)

The product symbol, π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.