Engineering Science Courses (ESC)-I/II

Introduction to Electronics and Communication

Sub Code: 22ESC143/243

Shalini Hanok (SH)
Assistant Professor

Navya N (NN)
Assistant Professor

Dept of ECE
ATME, Mysuru

- The term digital refers to any process that is accomplished using discrete units.

Number Systems

There are four number systems of arithmetic that are used in the digital systems.

- Decimal
- Binary
- Hexadecimal
- Octal.

Decimal Number System

It has ten symbols and any number of any magnitude can be expressed by using the system of positional weighting.

Eg:- $3628 = 3000 + 600 + 20 + 8$

$= 3 \times 10^3 + 6 \times 10^2 + 2 \times 10^1 + 8 \times 10^0$

In general,

$$N = d_n \times r^n + d_{n-1} \times r^{n-1} + \cdots + d_1 \times r^1 + d_0 \times$$

where, N – Value of the entire number

d_n – value of the n^{th} digit from decimal po

r – radix or base

The ten symbols used in decimal number system are,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

The base for decimal number is 10.

Binary Number System

It has 2 symbols, 0 and 1. These are called bits. or binary digits. A group of 8 bits is called a byte and a group of 4 bits is called a nibble.

The base for binary number is 2.

Octal Number System

It has 8 symbols. They are,

0, 1, 2, 3, 4, 5, 6, 7.

$8 = 2^3$. 3 bit binary numbers can be used to represent octal numbers.

The base for octal number is 8.

Hexadecimal Number system

Base is 16.

Hexadecimal means 16. There are 16 symbols and those are,

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

$16 = 2^4$. ∴ 4 bit binary numbers can be used to represent hexadecimal numbers. Since both number & alphabets are used, it is also called as alphanumeric number system.

Number Conversion

- Binary to Decimal conversion

$( )_2 = ( )_{10}$

The procedure for converting a binary number to decimal is similar to that of breaking a decimal number into its weighted values.

We know, $N = d_n * r^n + d_{n-1} * r^{n-1} + \ldots + d_1 * r^1 + d_0 * r^0$

d's will be 0s or 1s and $r = 2$ &

n's will be various powers of 2, depending on the position of the digit with reference to the binary point.

Eg: Convert $(110101)_2$ to $( )_{10}$

$= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 32 + 16 + 0 + 4 + 0 + 1$

$= 53$

$\therefore (110101)_2 = (53)_{10}$

Note: In a binary integer, the right most bit which has the least significance is called LSB (Least Significant Bit). and the left most bit which has the highest or most significan is called MSB (Most significant Bit)

Eg:- 1 1 0 1 0 1

MSB LSB

For a fractional number, the powers of 2 towards the left of the decimal point starts from 0 and the powers of 2 towards the right of the decimal point starts from -1 and goes negative.

Eg:- $(11011.1011)_2 = (\ )_{10}$.

$1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 . 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$

$16 + 8 + 0 + 2 + 1 . 0.5 + 0 + 0.125 + 0.0625$

$= (27.6875)_{10}$

$\therefore (11011.1011)_2 = (27.6875)_{10}$

- Octal to Decimal conversion

$(\ )_8 = (\ )_{10}$

The procedure is similar to that of converting binary to decimal except that the base value 'r' is taken as 8. & ni will be various powers of 8 depending on the position of the digit with reference to the decimal point.

Eg:- $(1053)_8 = (\ )_{10}$

$= 1\times8^3 + 0\times8^2 + 5\times8^1 + 3\times8^0$

$= 512 + 0 + 40 + 3$

$= 555$

$\therefore (1053)_8 = (555)_{10}$

For a fractional number,

Eg:- $(347.216)_8 = (\quad)_{10}$

$= 3\times8^2 + 4\times8^1 + 7\times8^0 . \; 2\times8^{-1} + 1\times8^{-2} + 6\times8^{-3}$

$= 192 + 32 + 7 . \; 0.25 + 0.0156 + 0.0117$

$= (231.2773)_{10}$

$\therefore (347.216)_8 = (231.2773)_{10}$

Hexadecimal to Decimal conversion

$(\quad)_{16} = (\quad)_{10}$

The procedure is similar to that of converting binary to decimal except that the base value r is taken as 16 and the n's will be various powers of 16 depending on the position of the digit with reference to the decimal point. If the digit is an alphabet, the decimal equivalent value is taken.

Hex	Dec
A	10
B	11
C	12
D	13
E	14
F	15

Eg:- $(52B)_{16} = (\quad)_{10}$

$= 5\times16^2 + 2\times16^1 + 11\times16^0$

$= 1280 + 32 + 11$

$= (1323)_{10}$

$\therefore (52B)_{16} = (1323)_{10}$

For a fractional numbes,

Eg:- $(F8E.28C)_{16} = (\quad)_{10}$

$= 15 \times 16^2 + 8 \times 16^1 + 14 \times 16^0 . 2 \times 16^{-1} + 8 \times 16^{-2} + 12 \times 16^{-3}$

$= 3840 + 128 + 14 . 0.125 + 0.03125 + 0.00292$

$= (3982.15917)_{10}$

$\therefore (F8E.28C)_{16} = (3982.15917)_{10}$

- Decimal to binary conversion

To convert a decimal number to its binary equivalent, progressively divide the decimal number by 2, noting the remainders. The remainders taken in the reverse order form the binary equivalent

To convert a decimal fraction to its binary equivalent, progressively multiply the fraction by 2, removing and noting the carries. The carries taken in forward order form the binary equivalent.

This method is known as double-dabble method.

Eg:- Convert $(33)_{10} = (\ )_2$

2	33	
2	16 – 1	← Remainders
2	8 – 0	
2	4 – 0	
2	2 – 0	
	1 – 0	

$(100001)_2$

$\therefore (33)_{10} = (100001)_2$

Eg:- $(25.375)_{10} = (\ )_2$

2	25	
2	12 – 1	
2	6 – 0	
2	3 – 0	
	1 – 1	

$25 = 11001$

$0.375 \times 2 = 0.75 \rightarrow 0$

$0.75 \times 2 = 1.5 \rightarrow 1$

$0.5 \times 2 = 1.0 \rightarrow 1$

$0.375 = 011$

$\therefore (25.375)_{10} = (11001.011)_2$

Decimal to Octal Conversion

The procedure is similar to decimal to binary conversion. Instead of dividing by 2, divide progressively by 8. For fractional part, instead of multiplying by 2, multiply progressively by 8.

Eg:- $(294.6875)_{10} = (\ )_8$

Consider integer part that is 294

8 | 294		
8 | 36	– 6	
4	– 4	↑

$294 = (446)_8$

Fractional part

$0.6875 \times 8 = 5.5 \longrightarrow 5$

$0.5 \times 8 = 4.0 \longrightarrow 4$ ↓

$(0.6875)_{10} = (0.54)_8$

$\therefore (294.6875)_{10} = (446.54)_8$

Eg:- $(123.456)_{10} = (\ )_8$

8 | 123		
8 | 15	– 3	
1	– 7	↑

$0.456 \times 8 = 3.648 \longrightarrow 3$

$0.648 \times 8 = 5.184 \longrightarrow 5$ ↓

$0.184 \times 8 = 1.472 \longrightarrow 1$

$0.472 \times 8 = 3.776 \longrightarrow 3$

$\therefore (123.456)_{10} = (173.3513)_8$

Decimal to Hexadecimal Conversion

The procedure is similar to decimal to binary conversion. Instead of diriding by 2, divide progressively by 16. The remainder is taken and if it is between 10-15, take equivalent alphabet for hexa ie A-F. For fractional part, instead of multiplying by 2, multiply progressively by 16.

Eg:- $(458.341)_{10} = (\quad)_{16}$

16 | 458
16 | 28 — 10 → A
1 — 12 → C ↑

= 1CA

$0.341 \times 16 = 5.456 \longrightarrow 5$
$0.456 \times 16 = 7.296 \longrightarrow 7$
$0.296 \times 16 = 4.736 \longrightarrow 4$
$0.736 \times 16 = 11.776 \longrightarrow 11 \rightarrow B$ ↓

= 0.574B

$\therefore (458.341)_{10} = (1CA.574B)_{16}$

— Octal to Binary conversion

Each octal digit can be represented by 3 binary bits. To convert octal number to binary, write the binary equivalent of each octal digit from the table.

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

Eg:- $(642.731)_8 = (\quad)_2$

110 100 010 . 111 011 001

$\therefore (642.731)_8 = (110100010.111011001)_2$

- Binary to Octal Conversion

Make groups of 3 bits starting from LSB and move towards MSB for the integer part. For the fractional part, start grouping from the bit next to the binary point and move towards the right.

Eg:- $(1011001.100101)_2 = (\quad)_8$

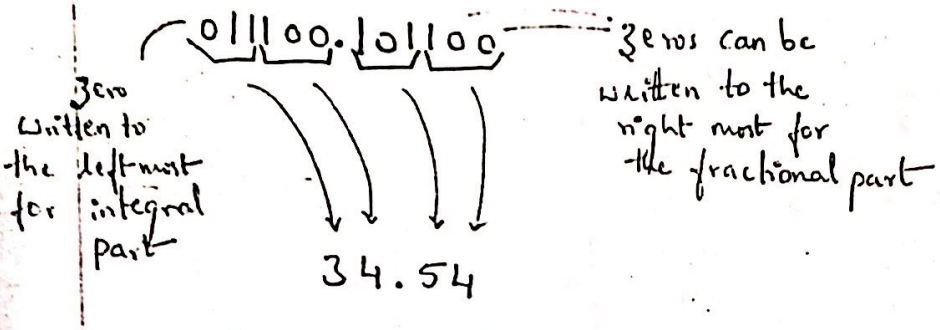
001 011 001 . 100 101

extra zeros can be added towards left most

131.45

$\therefore (1011001.100101)_2 = (131.45)_8$

Eg:- $(11100.1011)_2 = (\quad)_8$



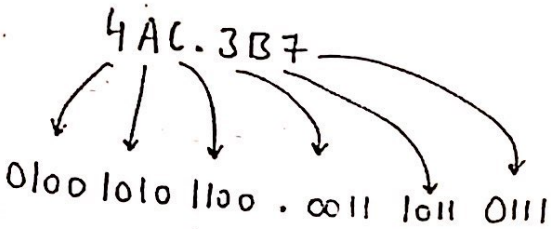
$\therefore (11100.1011)_2 = (34.54)_8$

- Hexadecimal to Binary conversion.

Each hexadecimal digit can be represented by 4 binary bits. To convert hexadecimal number to binary, write the binary equivalent of each hexadecimal digit from the table.

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

Eg:- $(4AC.3B7)_{16} = (\quad)_2$



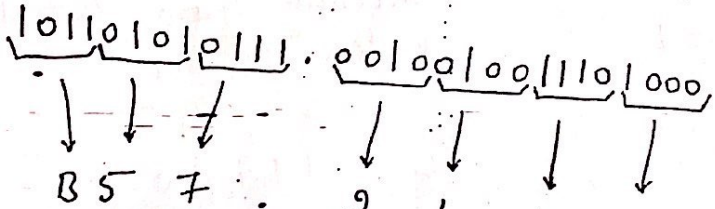
$\therefore (4AC.3B7)_{16} = (010010101100.001110110111)_2$

- Binary to Hexadecimal conversion

Make groups of 4 bits starting from LSB and move towards MSB, for the integer part. To form a group of 4, extra zeros can be written near the MSB if necessary.

For the fractional part, start grouping from the bit next to the binary point and move towards the right.

Eg:- $(101101010111.001001001101)_2 = (\quad)_{16}$



$\therefore (101101010111.001001001101)_2 = (B57.24E8)_{16}$

Binary Arithmetic

Binary Addition

Rules:

$0+0=0$
$0+1=1$
$1+0=1$
$1+1=(1)0$

↓ Carry to the next bit

$1+1+1 = 10+1 = 11$

Eg:- $(1110)_2$ and $(1011)_2$

```
 1 1   ← carry
  1110
+ 1011
-------
 11001
```

Binary Subtraction

Rules:

$0-0=0$
$1-0=1$
$1-1=0$
$10-1=1$

Subtract column by column and take borrow whenever necessary from the higher position columns.

Eg:-

```
  1101
- 1001
------
  0100
```

Eg:-

```
    0 10      ← Borrow
  1 1 ✗ ø 0
- 0 1 0 1 0
----------
  1 0 0 1 0
```

Binary Multiplication

Rules:- $0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Eg:- Multiply 1101 and 101

$$
\begin{array}{r}
1101 \times 101 \\
\hline
1101 \\
0000+ \\
1101++ \\
\hline
1000001
\end{array}
$$

Eg:- Multiply 1111 and 111

$$
\begin{array}{r}
1111 \times 111 \\
\hline
1111 \\
1111+ \\
1111++ \\
\hline
1101001
\end{array}
$$

Binary Division

Eg:- $1011 \div 10$

$= 101 \rightarrow$ Quotient
$1 \rightarrow$ Remainder

$$
\begin{array}{r|l}
10 & 1011 \\
\cline{2-2}
 & 10 \\
\cline{2-2}
 & 011 \\
 & 10 \\
\cline{2-2}
 & 1
\end{array}
\quad (101
$$

Eg:- $1011011 \div 111$

```
      1101
111 ) 1011011 (
      111
      ----
       1000
        111
       ----
        0001 11
           111
          ----
           000
```

$= 1101$

Signed Numbers

In decimal number system, we use '+' sign for denoting positive numbers and '−' sign for denoting negative numbers. As these signs are not available in the binary system, there are 3 binary signed number systems.

1. Sign-magnitude representation
2. One's complement representation
3. Two's complement representation

1. Sign-magnitude

In sign-magnitude representation, the most significant bit (MSB) is used to represent the sign (0 for positive and 1 for negative) and the remaining bits are used to represent the magnitude of the number.

2. One's complement notation

One's complement of a binary number is obtained by replacing each 0 by 1 and each 1 by 0.

Eg:- Complement of 101110010 is - 010001101

Using one's complement to represent positive and negative number, for a 4-bit number.

→ Eg: $(0111)_2 = (+7)_{10}$

one's Complement = $(1000)_2 = (-7)_{10}$

Note that MSB denotes the sign of the number. (0 for positive and 1 for negative)

The magnitude of the negative numbers is obtained by taking the one's compliment of the number.

5-bit number

Eg:- $(00110)_2 = (+6)_{10}$

One's complement = $(11001)_2 = (-6)_{10}$

Magnitude of the -ve number, take 1's complement of 11001, ⇒ 00110 = |6|

3. Two's complement notation

To get a two's complement of a binary number, add 1 to the one's complement.

Eg:- 101101

2's complement ⇒ 1's complement = 010010

$$\begin{array}{r} 010010 \\ + \quad\quad 1 \\ \hline 010011 \end{array}$$

2's complement of 101101 is 010011.

Using 2's complement to represent negative numbers,

Eg:- 0101 $= (+5)_{10}$

To represent -5, take 2's complement of 0101 which is 1011 (1's complement +1) represents -5.

MSB is used to denote the sign of the number.

0 → positive
1 → negative.

Here 1011 is a negative number. To get the magnitude, take 2's complement of it,

$$\begin{array}{r} 0100 \\ \quad\quad 1 \\ \hline 0101 \end{array} = |5|$$

Data types

There are two types of data
→ Signed : Both the & -ve numbers.
→ Unsigned : Only the numbers.

example +6 ⟹ 0110
-6 ⟹ 1110 In Signed numbers MSB (Most significant) is used to represent sign of number

-8 ⇒ 10001000

Range of integer data values that can be represented as bytes and words are as follows:

Data types	Bits	Range of Values
Unsigned byte	8	0 to 255
Signed byte	8	-128 to +127
Unsigned word	16	0 to 65,535
Signed word	16	-32768 to +32767

Logic gates : There are the electronic circuits which are the basic building blocks of all digital systems

Logic gate is a physical device implementing a Boolean function i.e., it performs a logical operation on one or more binary inputs & produces a single output.

Logic gate requires a power supply. Gate inputs are driven by voltages having 2 nominal values, 0V & 5V representing logic 0 & logic 1 respectively.

Truth table: It shows the behaviour of the logic gate

Basic Gates:

① NOT Gate (Inverter): It is an electronic circuit that produces the inverted version of the input at its output

Symbol

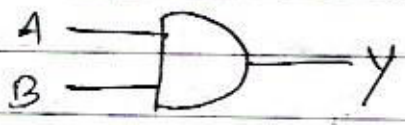
A	Y
0	1
1	0

Truth table

output: $Y = \overline{A}$

② AND Gate: It is equivalent to mathematical multiplication operation

The output of AND gate is high only if all its inputs are high.



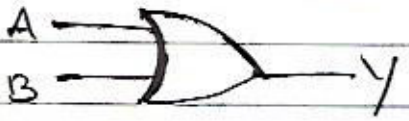
$Y = A \cdot B$

dot

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

③ OR Gate :- It is equivalent to mathematical Addition operation.

The output of OR Gate is high if one or more of its inputs are high.



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

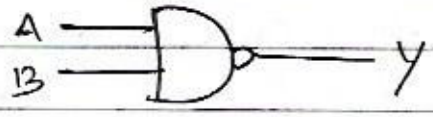
$Y = A + B$

↑
plus

Universal Gates

① NAND Gate : It is equivalent to AND gate followed by NOT gate.

The output of NAND Gate is high if any of its inputs are low.

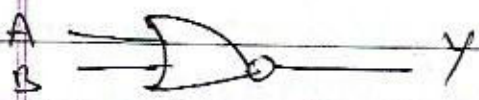


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

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

② NOR Gate : It is equivalent to OR gate followed by NOT gate.

The output of NOR Gate is low if any of its inputs are high



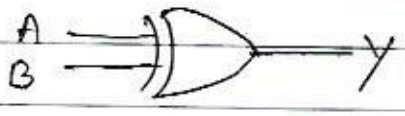
$Y = \overline{A+B}$

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

Exclusive gates

① XOR Gate:

Output of XOR gate is high if either, but not both, of its two inputs are high.



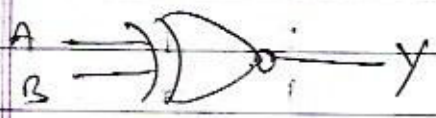
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = A \oplus B = \bar{A}B + A\bar{B}$

↑ encircled plus sign

② XNOR Gate:

Output of XNOR is low, if either or not both, of its two inputs are high



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

$Y = A \odot B$

$= \overline{A \oplus B}$

$Y = AB + \bar{A}\bar{B}$

Buffer (Non-inverter)

The output of Buffer is same as its input.

A	Y
0	0
1	1

$Y = A$

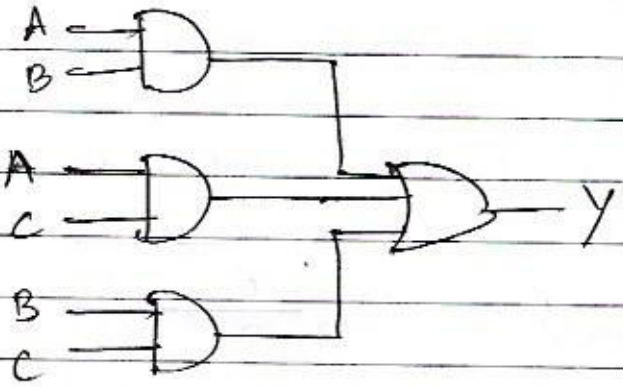
Note: Buffer is used to increase the fan-out of a given logic gate.
Fan-out refers to the maximum no. of gates (inputs) that the output of a single logic gate can feed.

Problems:

① Design a logic circuit that will produce a logic 1 output whenever two or more of its three inputs are at logic 1.

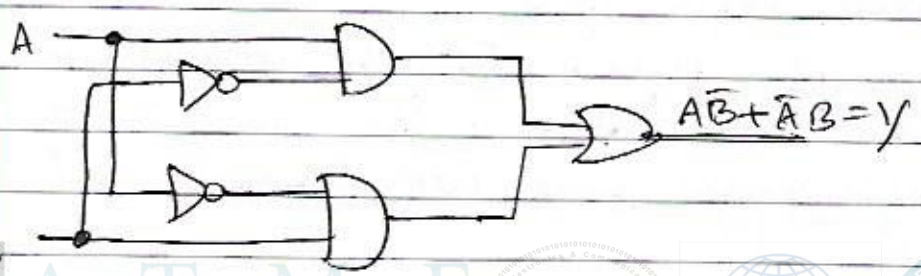
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$
$= BC(A+\bar{A}) + A\bar{B}C + AB\bar{C}$
$= BC + A\bar{B}C + AB\bar{C}$
$=$
$= B(C + A\bar{C}) + A\bar{B}C$
$= B(C+A) + A\bar{B}C$
$= BC + AB + A\bar{B}C$
$= BC + A(B + \bar{B}C)$
$= BC + A(B+\bar{B})$
$Y = BC + AB + AC$



(2) Realize XOR function using logic gates AND, OR, NOT

XOR ⟹ $\bar{A}B + A\bar{B}$



Half Adder

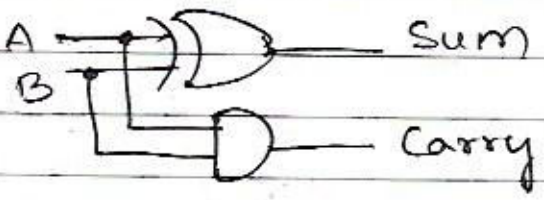
It adds 2 1-bit binary numbers A & B and produces output as Sum (S) and Carry (C).

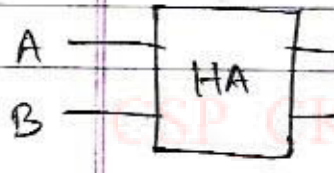
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum = $\bar{A}B + A\bar{B}$
= $A \oplus B$ (XOR)

Carry = AB (AND)

Schematic





Full Adder

It adds 3 1-bit binary numbers A, B and Cin. and produces output as Sum (S) and carry (C_o)

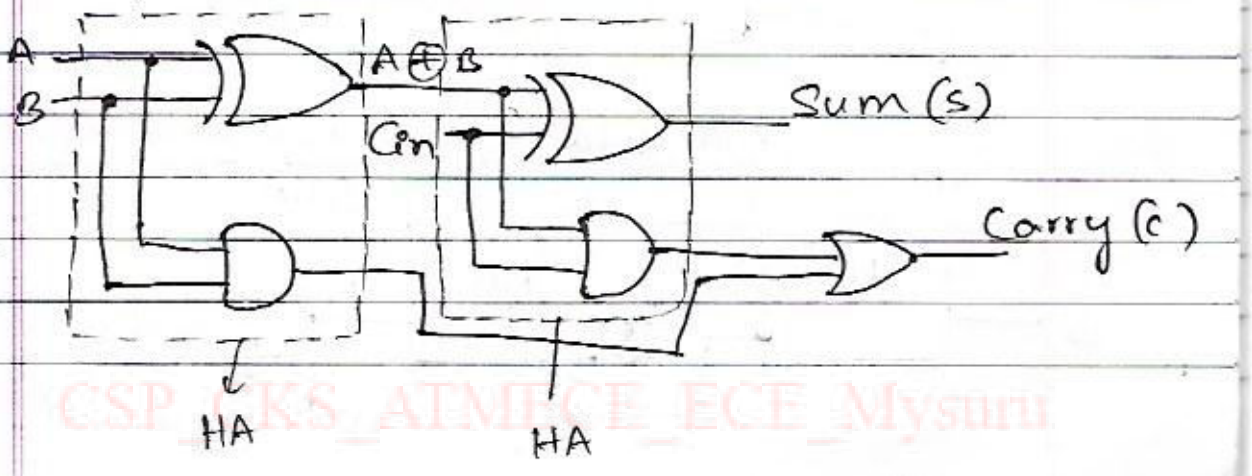
A	B	Cin	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum } (S) = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$
$$= (\bar{A}\bar{B} + AB)\,C_{in} + (\bar{A}B + A\bar{B})\,\bar{C}_{in}$$
$$= A \oplus B \oplus C_{in} \quad \text{---} (1)$$

$$\text{Carry } (C) = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$= (\bar{A}B + A\bar{B})\,C_{in} + AB(\bar{C}_{in} + C_{in})$$
$$= (A \oplus B)\,C_{in} + AB \quad \text{---} (2)$$

Full adder using 2 Half adder

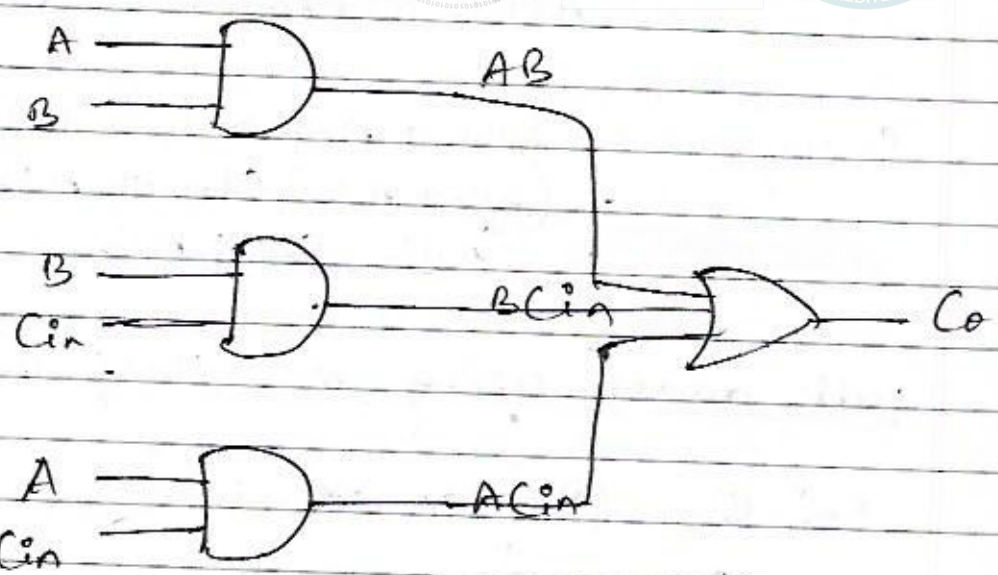
eq^n (1) & (2) can be used as below:



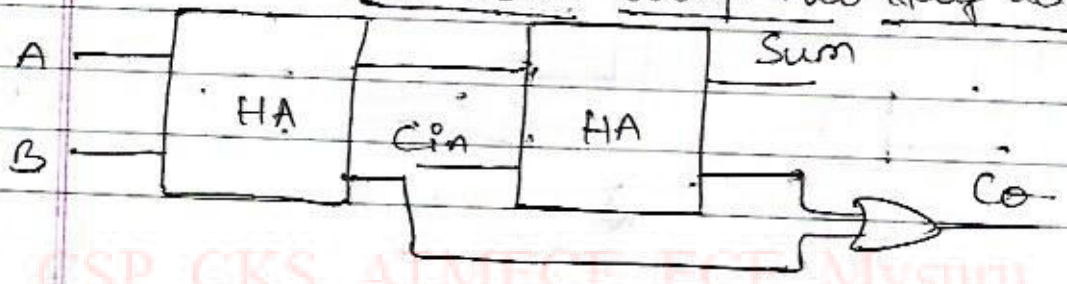
eqn ② can also be simplified as

$$C_o = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$
$$= \bar{A}BC_{in} + A\bar{B}C_{in} + AB(\bar{C}_{in} + C_{in})$$
$$= \underline{\bar{A}BC_{in} + A\bar{B}C_{in} + AB}$$
$$= B[\bar{A}C_{in} + A] + A\bar{B}C_{in}$$
$$= B[(A+\bar{A})(A+C_{in})] + A\bar{B}C_{in}$$
$$= BA + BC_{in} + A\bar{B}C_{in}$$
$$= BA + C_{in}[B + A\bar{B}]$$
$$= BA + C_{in}[(B+A)(B+\bar{B})]$$
$$C_o = BA + BC_{in} + AC_{in} \quad \text{— ③}$$

Implementing ① & ③



full adder using two Half adder



Boolean algebra and Logic Gates

BOOLEAN OPERATIONS AND EXPRESSIONS

Variable, complement, and literal are terms used in Boolean algebra. A variable is a symbol used to represent a logical quantity. Any single variable can have a 1 or a 0 value. The complement is the inverse of a variable and is indicated by a bar over variable (overbar). For example, the complement of the variable A is A. If A = 1, then A = 0. If A = 0, then A = 1. The complement of the variable A is read as "not A" or "A bar." Sometimes a prime symbol rather than an overbar is used to denote the complement of a variable; for example, B' indicates the complement of B. A literal is avariable or the complement of a variable.

LAWS AND RULES OF BOOLEAN ALGEBRA OR POSTULATES AND THEOREMS

■ *Laws of Boolean Algebra*

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

Commutative Laws

►The commutative law of addition for two variables is written as

A+B = B+A

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. Fig.(4-1) illustrates the commutative law as applied to the OR gate and shows that it doesn't matterto which input each variable is applied. (The symbol ≡ means "equivalent to.").

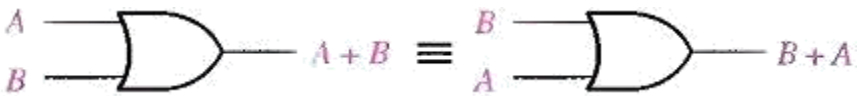


Fig.(4-1) Application of commutative law of addition.

► The commutative law of multiplication for two variables

is A.B = B.A

This law states that the order in which the variables are ANDed makes no difference. Fig.(4-2), il1ustrates this law as applied to the AND gate.

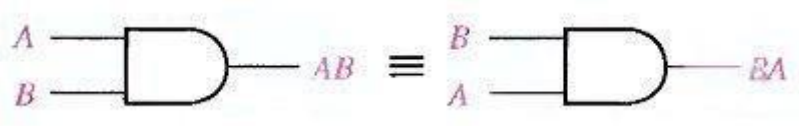


Fig.(4-2) Application of commutative law of multiplication.

Associative Laws :

►The associative law of addition is written as follows for three variables:

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

This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Fig.(4-3), illustrates this law as applied to 2-input OR gates.

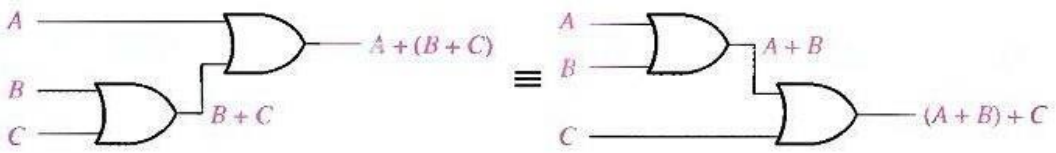


Fig.(4-3) Application of associative law of addition.

► The associative law of multiplication is written as follows for three variables:

A(BC) = (AB)C

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Fig.(4-4) illustrates this law as applied to 2-input AND gates.

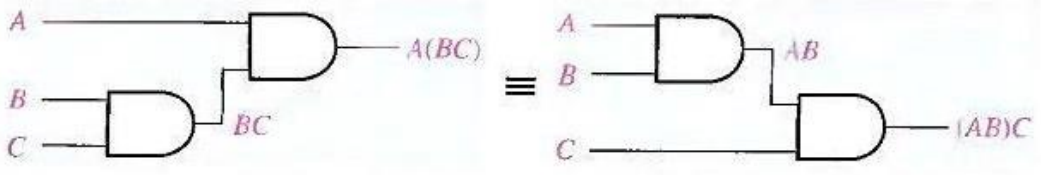


Fig.(4-4) Application of associative law of multiplication.

Distributive Law:

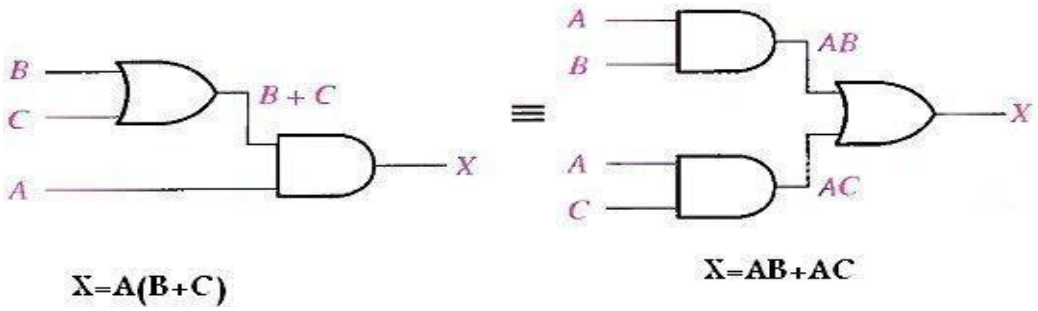
► The distributive law is written for three variables as follows:

$$A(B + C) = AB + AC$$

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of factoring in which the common variable A is factored out of the product terms, for example,

$$AB + AC = A(B + C).$$

Fig.(4-5) illustrates the distributive law in terms of gate implementation.



Rules of Boolean Algebra

Table 4-1 lists 12 basic rules that are useful in manipulating and simplifying Boolean expressions. Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.

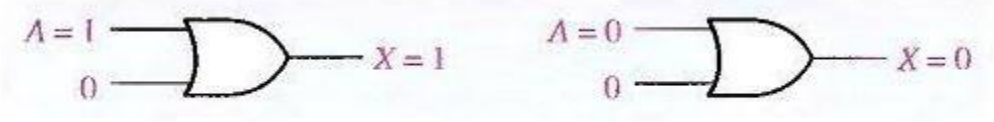
Table 4-1 Basic rules of Boolean algebra.

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

A, B, or *C* can represent a single variable or a combination of variables.

Rule 1. A + 0 = A

A variable ORed with 0 is always equal to the variable. If the input variable A is 1, the output variable X is 1, which is equal to A. If A is 0, the output is 0, which is also equal to A. This rule is illustrated in Fig.(4-6), where the lower input is fixed at 0.



A+0=A

Fig.(4-6)

Rule 2. A + 1 = 1

A variable ORed with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input. This rule is illustrated in Fig.(4-7), where the lower input is fixed at 1.

X=A+1=1

Fig.(4-7)

Rule 3. A . 0 = 0

A variable ANDed with 0 is always equal to 0. Any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Fig.(4-8), where the lower input is fixed at 0.

X= A.0 = 0

Fig.(4-8)

Rule 4. A . 1 = A

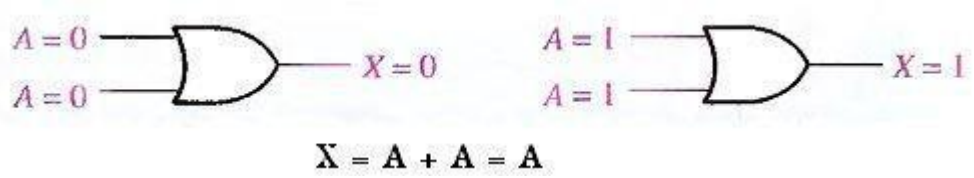
A variable ANDed with 1 is always equal to the variable. If A is 0 the output of the AND gate is 0. If A is 1, the output of the AND gate is 1 because both inputs are now 1s. This rule is shown in Fig.(4-9), where the lower input is fixed at 1.

X = A . 1 = A

Fig.(4-9)

Rule 5. A + A = A

A variable ORed with itself is always equal to the variable. If A is 0, then 0 + 0 = 0; and if A is 1, then 1 + 1 = 1. This is shown in Fig.(4-10), where both inputs are the same variable.

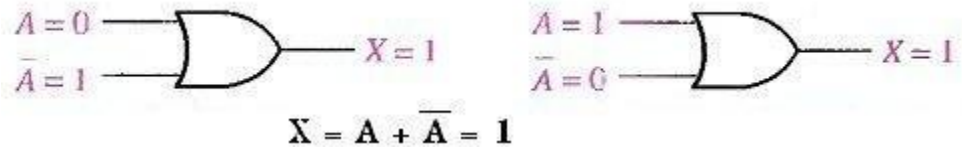


$X = A + A = A$

Fig.(4-10)

Rule 6. $A + \overline{A} = 1$

A variable ORed with its complement is always equal to 1. If A is 0, then 0 + $\overline{0}$ = 0 + 1 = 1. If A is l, then 1 + 1 = 1+ 0 = 1. See Fig.(4-11), where one input is the complement of the other.

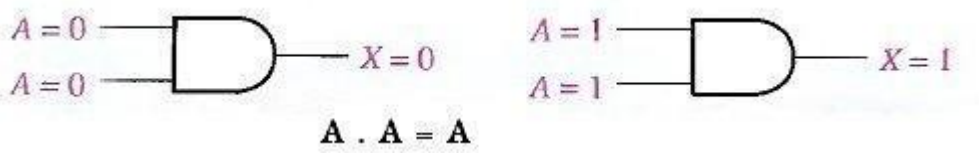


$X = A + \overline{A} = 1$

Fig.(4-11)

Rule 7. A . A = A

A variable ANDed with itself is always equal to the variable. If A = 0, then 0.0 = 0; and if A = 1. then 1.1 = 1. Fig.(4-12) illustrates this rule.



$A \cdot A = A$

Fig.(4-12)

Rule 8. $A \cdot \overline{A} = 0$

A variable ANDed with its complement is always equal to 0. Either A or $\overline{A}$ will always be 0: and when a 0 is applied to the input of an AND gate. the output will be 0 also. Fig.(4-13) illustrates this rule.

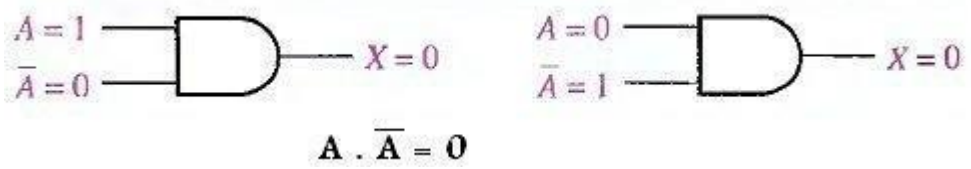


Fig.(4-13)

Rule 9 $A = \overline{\overline{A}}$

The double complement of a variable is always equal to the variable. If you start with the variable A and complement (invert) it once, you get ~~A.~~ If you then take ~~A~~ and complement (invert) it, you get A, which is the original variable. This rule is shown in Fig.(4-14) using inverters.

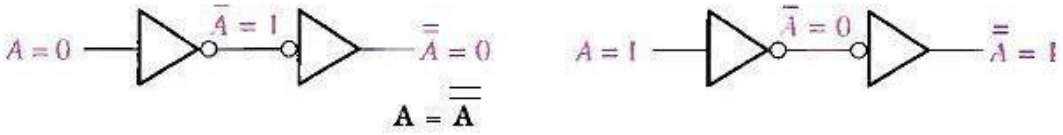


Fig.(4-14)

Rule 10. $A + AB = A$

This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

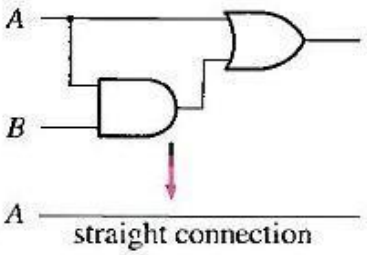
$A + AB = A(1 + B)$	Factoring (distributive law)
$= A \cdot 1$	Rule 2: $(1 + B) = 1$
$= A$	Rule 4: $A \cdot 1 = A$

The proof is shown in Table 4-2, which shows the truth table and the resulting logic circuit simplification.

Table 4-2

A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal



Rule 11. $A + A\overline{B} = A + B$

This rule can be proved as follows:

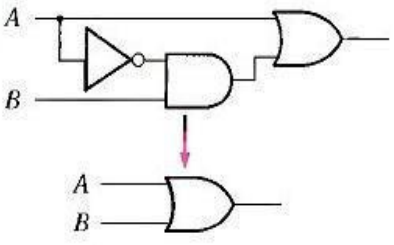
$A + \overline{A}B = (A + AB) + \overline{A}B$	Rule 10: A = A + AB
$= (AA + AB) + \overline{A}B$	Rule 7: A = AA
$=AA +AB +A\overline{A} +\overline{A}B$	Rule 8: adding $A\overline{A} = 0$
$= (A + \overline{A})(A + B)$	Factoring
$= 1. (A + B)$	Rule 6: $A + A\overline{} = 1$
$=A + B$	Rule 4: drop the 1

The proof is shown in Table 4-3, which shows the truth table and the resulting logic circuit simplification.

Table 4-3

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

equal



Rule 12. $(A + B)(A + C) = A + BC$

This rule can be proved as follows:

$(A + B)(A + C) = AA + AC + AB + BC$ Distributive law

$= A + AC + AB + BC$ Rule 7: $AA = A$

$= A(1 + C) + AB + BC$ Rule 2: $1 + C = 1$

$= A. 1 + AB + BC$ Factoring (distributive law)

$= A(1 + B) + BC$ Rule 2: $1 + B = 1$

$= A. 1 + BC$ Rule 4: $A . 1 = A$

$= A + BC$

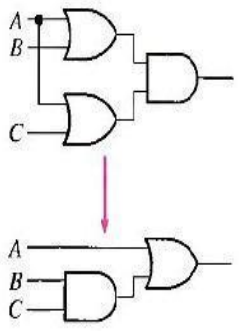
The proof is shown in Table 4-4, which shows the truth table and the resulting logic circuit

simplification.

Table 4-4

A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



DEMORGAN'S THEOREMS

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms. DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates, which were discussed in part 3.

One of DeMorgan's theorems is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables,

Stated another way,

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is

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

DeMorgan's second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

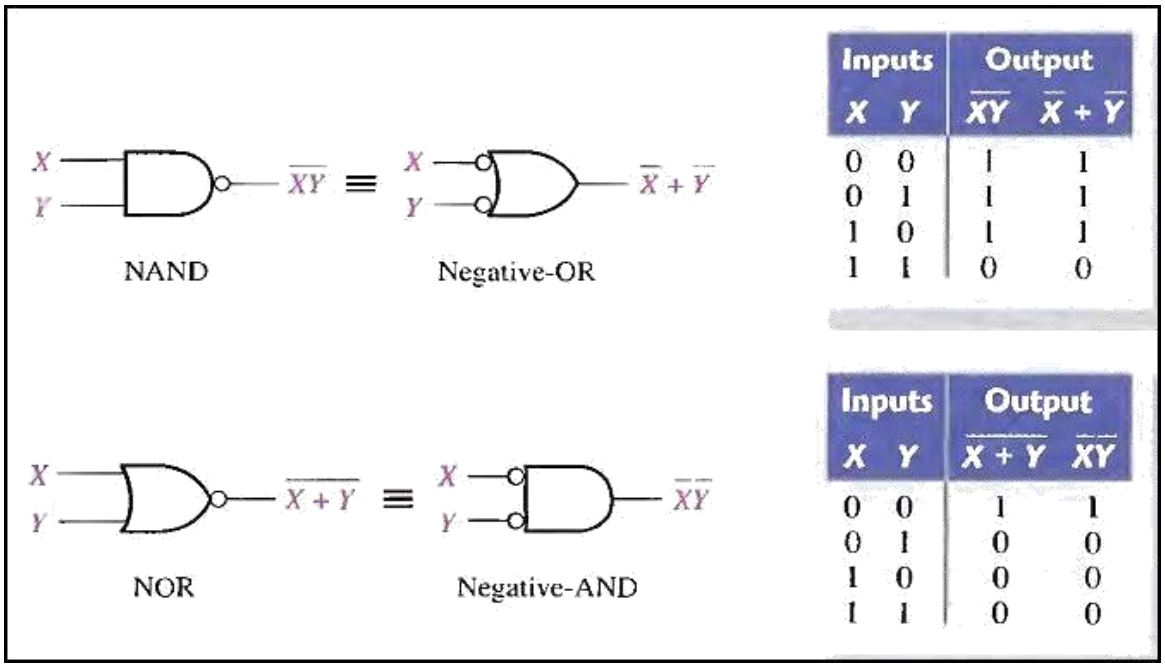
Stated another way,

The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables,

The formula for expressing this theorem for two variables is

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

Fig.(4-15) shows the gate equivalencies and truth tables for the two equations above.



Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Inputs		Output	
X	Y	$\overline{X+Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Fig.(4-15) Gate equivalencies and the corresponding truth tables that illustrate DeMorgan's theorems.

As stated, DeMorgan's theorems also apply to expressions in which there are more than two variables. The following examples illustrate the application of DeMorgan's theorems to 3-variable and 4-variable expressions.

Example

Apply DeMorgan's theorems to the expressions XYZ and X + Y + z.

$XYZ = X + Y + Z$

$\overline{X + y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$

Example

Apply DeMorgan's theorems to the expressions WXYZ and W + X + y + z.

$WXYZ = \overline{W} + \overline{X} + \overline{y} + \overline{Z}$

$\overline{W + X + y + Z} = W\,X\,Y\,Z$

SIMPLIFICATION USING BOOLEAN ALGEBRA

A simplified Boolean expression uses the fewest gates possible to implement a given expression.

Example

Using Boolean algebra techniques, simplify this expression:

AB + A(B + C) + B(B + C)

Solution

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

AB + AB + AC + BB + BC

Step 2: Apply rule 7 (BB = B) to the fourth term.

AB + AB + AC + B + BC

Step 3: Apply rule 5 (AB + AB = AB) to the first two terms.

AB + AC + B + BC

Step 4: Apply rule 10 (B + BC = B) to the last two terms.

AB + AC + B

Step 5: Apply rule 10 (AB + B = B) to the first and third terms.

B+AC

At this point the expression is simplified as much as possible.

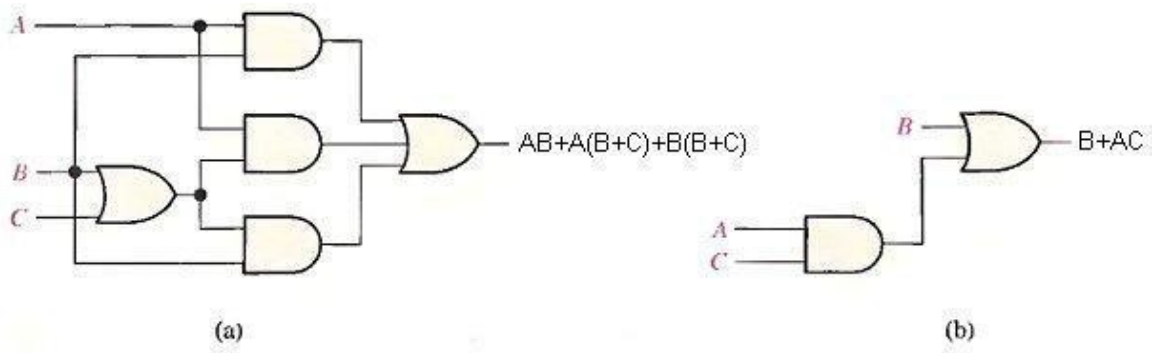


Fig.(4-17) Gate circuits for example above.

Example

Simplify the Boolean expressions:

1- $A\overline{B} + A(B + C) + B(B + C)$.

2- $[A\overline{B}(C + BD) + \overline{A}\,\overline{B}]C$

3- $\overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + ABC$

Standard and Canonical Forms:

STANDARD FORMS OF BOOLEAN EXPRESSIONS

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of- sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

The Sum-of-Products (SOP) Form

When two or more product terms are summed by Boolean addition, the resulting expression is a sum-of-products (SOP). Some examples are:

AB + ABC

ABC + CDE + BCDAB

+ BCD + AC

Also, an SOP expression can contain a single-variable term, as in

A + ABC + BCD.

In an SOP expression a single overbar cannot extend over more than one variable.

Example

Convert each of the following Boolean expressions to SOP form:

(a) AB + B(CD + EF)

(b) (A + B)(B + C + D)

(c) (A + B) + C

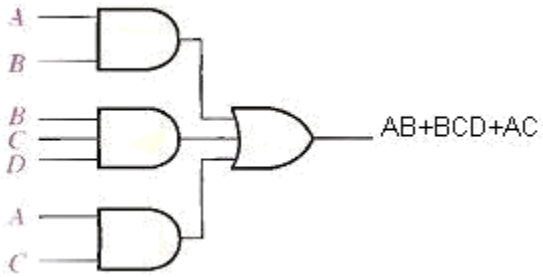


Fig.(4-18) Implementation of the SOP expression AB + BCD + AC.

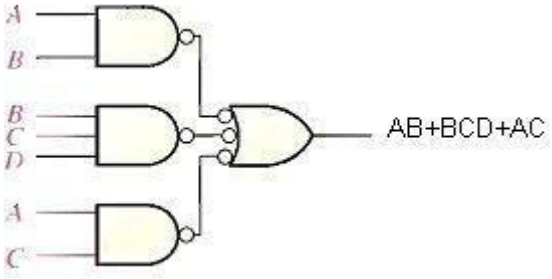


Fig.(4-19) This NAND/NAND implementation is equivalent to the AND/OR in figure above.

The Standard SOP Form

So far, you have seen SOP expressions in which some of the product terms do not contain all of the variables in the domain of the expression. For example, the expression ABC + ABD + ABCD has a domain made up of the variables A, B, C. and D. However, notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or D is missing from the first term and C or C is missing from the second term.

A standard SOP expression is one in which all the variables in the domain appear in each product term in the expression. For example, ABCD + ABCD + ABCD is a standard SOP expression.

Converting Product Terms to Standard SOP:

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard SOP to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard form using Boolean algebra rule 6 (A + A = 1) from Table 4-1: A variable added to its complement equals 1.

Step 1. Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As you know, you can multiply anything by 1 without changing its value.

Step 2. Repeat Step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable.

Example

Convert the following Boolean expression into standard SOP form: ABC + AB + ABCD

Solution

The domain of this SOP expression A, B, C, D. Take one term at a time. The first term, ABC, is missing variable D or D, so multiply the first term by (D + D) as follows:

ABC = ABC(D + D) = ABCD + ABCD

In this case, two standard product terms are the result.

The second term, AB, is missing variables C or C and D or D, so first multiply the second term by C + C as follows:

AB = AB(C + C) = ABC + ABC

The two resulting terms are missing variable D or D, so multiply both terms by (D + D) as follows:

ABC(D + D) + ABC(D + D)

= A BCD + ABCD + ABCD + ABCD

In this case, four standard product terms are the result.

The third term, ABCD, is already in standard form. The complete standard SOP form of the original expression is as follows:

ABC + AB + ABCD = ABCD + ABCD + A BCD + ABCD + ABCD + ABCD + ABCD

The Product-of-Sums (POS) Form

A sum term was defined before as a term consisting of the sum(Boolean addition) of literals (variables or their complements). When two or more sum terms are multiplied, the resulting expression is a product-of-sums(POS). Some examples are

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

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

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

A POS expression can contain a single-variable term, as in

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

In a POS expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar. For example, a POS expression can have the term A + B + C but not A + B + C.

Implementation of a POS Expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation and the product of two or more sum terms is produced by an AND operation. Fig.(4-

20) shows for the expression (A + B)(B + C + D)(A + C). The output X of the AND gate equals the POS expression.

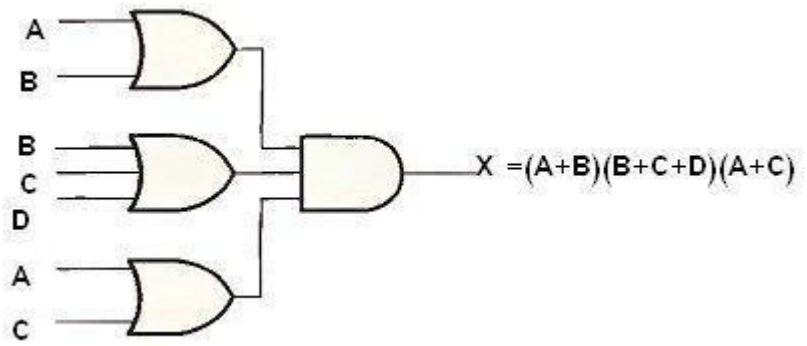


Fig.(4-20)

The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression. For example,the expression

(A + B + C) (A + B + D) (A + B + C + D)

has a domain made up of the variables A, B, C, and D. Notice that the complete set of variables in the domain is not represented in e first two termsof the expression; that is, D or D is missing from the first term and C or C is missing from the second term.

A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression. For example,

(A + B + C + D)(A + B + C + D)(A + B + C + D)

is a standard POS expression. Any nonstandard POS expression (referred to simply as POS) can be converted to the standard form using Boolean algebra.

Converting a Sum Term to Standard POS

Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a

nonstandard POS expression is converted into standard form using Boolean algebra rule 8 (A A = 0) from Table 4-1:

Step 1. Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms. As you know, you can add 0 to anything without changing its value.

Step 2. Apply rule 12 from Table 4-1: A + BC = (A + B)(A + C)

Step 3. Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or noncomplemented form.

Example

Convert the following Boolean expression into standard POS form: (A + B + C)(B + C + D)(A + B + C + D)

Solution

The domain of this POS expression is A, B, C, D. Take one term at a time. The first term, A + B + C, is missing variable D or D, so add DD and apply rule 12 as follows:

A + B + C = A + B + C + DD = (A + B + C + D)(A + B + C + D)

The second term, B + C + D, is missing variable A or A, so add AA and apply rule 12 as follows:

B + C + D = B + C + D + AA = (A + B + C + D)(A + B + C + D)

The third term, A + B + C + D, is already in standard form. The standard POS form of the original expression is as follows:

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

Examples:-

1. Identify each of the following expressions as SOP, standard SOP, POS, or standard POS:
   (a) $AB + \overline{A}BD + \overline{A}C\overline{D}$ (b) $(A + \overline{B} + C)(A + B + \overline{C})$
   (c) $\overline{A}BC + AB\overline{C}$ (d) $A(A + \overline{C})(A + B)$
2. Convert each SOP expression in Question 1 to standard form.
3. Convert each POS expression in Question 1 to standard form.

CANONICAL FORMS OF BOOLEAN EXPRESSIONS

n variables can be combined to form 2^n minterms.

Note that each maxterm is the complement of its corresponding minterm and vice versa.

Minterms and maxterms are related

- Any minterm m_i is the *complement* of the corresponding maxterm M_i

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

- For example, $m_4' = M_4$ because $(xy'z')' = x' + y + z$

46

For example the function F

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

$F = \bar{x}\,\bar{y}\,z + x\,\bar{y}\,\bar{z} + x\,y\,z$

$F = m_1 + m_4 + m_7$

Any Boolean function can be expressed as a sum of minterms (sum of products **SOP**) or product of maxterms (product of sums **POS**).

$F = x\,y\,z + \bar{x}\,y\,z + x\,y\,z + x\,y\,z + x\,y\,\bar{z}$

The complement of F = F = F

$F = (x + y + z)\,(x + y + z)\,(x + y + \bar{z})\,(x + y + z)\,(x + y + z)$ $F = M_0\,M_2\,M_3\,M_5\,M_6$

Example

Express the Boolean function F = A + BC in a sum of minterms (SOP).

Solution

The term A is missing two variables because the domain of F is (A, B, C)

$A = A(B + B) = AB + AB$ because $B + B = 1$

$\overline{B}C$ missing A, so

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

$AB(C + \overline{C}) = ABC + AB\overline{C}$

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

$F = ABC + AB\overline{C} + \underline{A\overline{B}C} + A\overline{B}\overline{C} + \underline{A\overline{B}C} + \overline{A}\overline{B}C$

Because A + A = A

$F = ABC + ABC + ABC + ABC + ABC$

$F = m_7 + m_6 + m_5 + m_4 + m_1$

In short notation

$F(A, B, C) = \sum(1, 4, 5, 6, 7)$

$\overline{F}(A, B, C) = \sum(0, 2, 3)$

The complement of a function expressed as the sum of minterms equal to the sum of minterms missing from the original function.

Truth table for $F = A + \overline{B}C$

	A	**B**	**C**	$\overline{\mathbf{B}}$	**BC**	**F**
0	0	0	0	1	0	0
1	0	0	1	1	1	1
2	0	1	0	0	0	0
3	0	1	1	0	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	1
6	1	1	0	0	0	1
7	1	1	1	0	0	1

Example

Express F = xy + xz in a product of maxterms form.

Solution

F = xy + xz = (xy + x)(xy + z) = (x + x)(y + x)(x + z)(y + z)

remember x + x = 1

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

F = (x + y + zz)(x + yy + z)(xx + y + z)

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

========== ---------- ---------- ==========

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

F = $M_4 M_5 M_0 M_2$

F(x, y, z) = ∏(0, 2, 4, 5)

F(x, y, z) = ∏(1, 3, 6, 7)

The complement of a function expressed as the product of maxterms equal to the product of maxterms missing from the original function.

To convert from one canonical form to another, interchange the symbols ∑, ∏ and list those numbers missing from the original form.

F = $M_4 M_5 M_0 M_2 = m_1 + m_3 + m_6 + m_7$

F(x, y, z) = ∏(0, 2, 4, 5) = ∑(1, 3, 6, 7)

Example

Develop a truth table for the standard SOP expression ABC + ABC + ABC.

INPUTS A	B	C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Converting POS Expressions to Truth Table Format

Reca11 that a POS expression is equal to 0 only if at least one of the sum terms is equal to 0. To construct a truth table from a POS expression, list all the possible combinations of binary values of the variables just as was done for the SOP expression. Next, convert the POS expression to standard form if it is not already. Finally, place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values. This procedure is illustrated in Example below:

Example

Determine the truth table for the following standard POS expression:

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

Solution

There are three variables in the domain and the eight possible binary values are listed in the left three columns of. The binary values that make thesum terms in the expression equal to 0 are A+ B + C: 000; A + B + C: 010:A + B + C: 011; A + B + C: 10l; and A + B + C: 110. For each of these binary values, place a 0 in the output column as shown in the table. For each of the remaining binary combinations, place a 1 in the output column.

INPUTS			OUTPUT	
A	B	C	X	SUM TERM
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \overline{B} + C)$
0	1	1	0	$(A + \overline{B} + \overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A} + B + \overline{C})$
1	1	0	0	$(\overline{A} + \overline{B} + C)$
1	1	1	1	

Boolean Functions

The binary variables and logic operations are used in Boolean algebra. The algebraic expression is known as **Boolean Expression**, is used to describe the **Boolean Function**. The Boolean expression consists of the constant value 1 and 0, logical operation symbols, and binary variables.

Example 1: F=xy' z+p

We defined the Boolean function **F=xy' z+p** in terms of four binary variables x, y, z, and p. This function will be equal to 1 when x=1, y=0, z=1 or z=1.

Example 2:

$$F(A,B,C,D) = A + \overline{BC} + ACD \qquad Equation\ No.1$$

Boolean Function *Boolean Expression*

The output Y is represented on the left side of the equation. So,

$$Y = A + BC + ACD$$

Apart from the algebraic expression, the Boolean function can also be described in terms of the truth table. We can represent a function using multiple algebraic expressions. They are their logically equivalents. But for every function, we have only one unique truth table.

In truth table representation, we represent all the possible combinations of inputs and their result. We can convert the switching equations into truth tables.

Every boolean function can be expressed by an algebraic expression, such as one mentioned above, or in terms of a Truth Table. A function may be expressed through several algebraic expressions, on account of them being logically equivalent, but there is only one unique truth table for every function.

Example: F(A, B ,C,D)=A+BC'+D

The output will be high when A=1 or BC'=1 or D=1 or all are set to 1. The truth table of the above example is given below. The 2^n is the number of rows in the truth table. The n defines the number of input variables. So the possible input combinations are 2^3=8.

Inputs				Output
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1