

Switching Theory And Logic Design

UNIT-I

NUMBER SYSTEM & BOOLEAN ALGEBRA

The Decimal Number system:

The Decimal number system contains ten unique symbols. 0,1,2,3,4,5,6,7,8,9. Since Counting in decimal involves ten symbols its base or radix is ten. There is no symbol for its base.i.e, for ten .It is a positional weighted system i.e,the value attached to a symbol depends on its location w.r.t. the decimalpoint.In this system, any no.(integer, fraction or mixed) of any magnitude can be rep. by the use of these ten symbols only.Each symbol in the no. is called a Digit. The leftmost digit in any no.rep ,which has the greatest positional weight out of all the digits present in that no.is called the MSD (Most Significant Digit) and the right most digit which has the least positional weight out of all the digits present in that no. is called the LSD(Least Significant Digit).The digits on the left side of the decimal pt. form the integer part of a decimal no. & those on the right side form the fractional part.The digits to the right of the decimal pt have weights which are negative powers of 10 and the digits to the left of the decimal pt have weights are positive powers of 10. The value of a decimal no.is the sum of the products of the digit of that no. with their respective column weights. The weights of each column is 10 times greater than the weight of unity or 10^{10} .The first digit to the right of the decimal pt. has a weight of $1/10$ or 10^{-1} .for the second $1/100$ & for third $1/1000$.In general the value of any mixed decimal no. is

$$d_n d_{n-1} d_{n-2} \dots\dots\dots d_1 d_0.d_{-1} d_{-2} d_{-3} \dots\dots\dots d_{-k} \quad \text{is given by}$$
$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + \dots\dots\dots (d_1 \times 10^1) + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \dots\dots\dots$$

9's & 10's Complements:

It is the Subtraction of decimal no.s can be accomplished by the 9's & 10's compliment methods similar to the 1's & 2's compliment methods of binary . the 9's compliment of a decimal no. is obtained by subtracting each digit of that decimal no. from 9. The 10's compliment of a decimal no is obtained by adding a 1 to its 9's compliment.

Example: 9's compliment of 3465 and 782.54 is

| | |
|-------|---------|
| 9999 | 999.99 |
| -3465 | -782.54 |
| ----- | ----- |
| 6534 | 217.45 |
| ----- | ----- |

$$\begin{array}{r}
10\text{'s complement of } 4069 \text{ is} \\
9999 \\
- 4069 \\
\hline
5930 \\
+ 1 \\
\hline
5931 \\
\hline
\end{array}$$

9's compliment method of subtraction:

To perform this, obtain the 9's compliment of the subtrahend and add it to the minuend now call this no. the intermediate result .if there is a carry to the LSD of this result to get the answer called **end around carry**.If there is no carry , it indicates that the answer is negative & the intermediate result is its 9's compliment.

Example: Subtract using 9's comp

(1) 745.81 - 436.62

$$\begin{array}{r}
745.81 \\
- 436.62 \\
\hline
309.19 \\
\hline
745.81 \\
+ 563.37 \quad \text{9's compliment of } 436.62 \\
\hline
1309.18 \quad \text{Intermediate result} \\
+ 1 \quad \text{end around carry} \\
\hline
309.19 \\
\hline
\end{array}$$

(2) 436.62 - 745.82

$$\begin{array}{r}
436.62 \\
- 745.81 \\
\hline
-309.19 \\
\hline
436.62 \\
+ 254.18 \\
\hline
690.80 \\
\hline
\end{array}$$

If there is no carry indicating that answer is negative . so take 9's complement of intermediate result & put minus sign (-) result should be -309.19
If carry indicates that the answer is positive +309.19

10's compliment method of subtraction:

To perform this, obtain the 10's compliment of the subtrahend & add it to the minuend. If there is a carry ignore it. The presence of the carry indicates that the answer is positive, the result is the answer. If there is no carry, it indicates that the answer is negative & the result is its 10's compliment. Obtain the 10's compliment of the result & place negative sign in front to get the answer.

Example: (a) 2928.54-41673

(b) 416.73-2928.54

```

2928.54
-0416.73
-----
2511.81
-----
2928.54
+9583.27  10's compliment of 436.62
-----
12511.81      ignore the carry
    
```

```

0416.73
-2928.54
-----
-2511.81
-----
0416.73
+7071.46
-----
7488.19
    
```

The Binary Number System:

It is a positional weighted system. The base or radix of this no. system is 2 Hence it has two independent symbols. The basic itself can't be a symbol. The symbol used are 0 and 1. The binary digit is called a bit. A binary no. consist of a sequence of bits each of which is either a 0 or 1. The binary point separates the integer and fraction parts. Each digit (bit) carries a weight based on its position relative to the binary point. The weight of each bit position is on power of 2 greater than the weight of the position to its immediate right. The first bit to the left of the binary point has a weight of 2^0 & that column is called the **Units Column**. The second bit to the left has a weight of 2^1 & it is in the 2's column & the third has weight of 2^2 & so on. The first bit to the right of the binary point has a weight of 2^{-1} & it is said to be in the $\frac{1}{2}$'s column , next right bit with a weight of 2^{-2} is in $\frac{1}{4}$'s column so on..The decimal value of the binary no. is the sum of the products of all its bits multiplied by the weight of their respective positions. In general , binary no. wioth an integer part of (n+1) bits & a fraction parts of k bits can be $d_n d_{n-1} d_{n-2} \dots\dots d_1 d_0.d_{-1} d_{-2} d_{-3} \dots\dots d_{-k}$

In decimal equivalent is

$$(d_n \times 2^n) + (d_{n-1} \times 2^{n-1}) + \dots\dots (d_1 \times 2^1) + (d_0 \times 2^0) + (d_{-1} \times 2^{-1}) + (d_{-2} \times 2^{-2}) \dots\dots$$

The decimal equivalent of the no. system

$d_n d_{n-1} d_{n-2} \dots\dots d_1 d_0.d_{-1} d_{-2} d_{-3} \dots\dots d_{-k}$ in any system with base b is

$$(d_n \times b^n) + (d_{n-1} \times b^{n-1}) + \dots\dots (d_1 \times b^1) + (d_0 \times b^0) + (d_{-1} \times b^{-1}) + (d_{-2} \times b^{-2}) \dots\dots$$

The binary no. system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors , diodes etc. A transistor can be OFF or ON a switch can be OPEN or CLOSED , a diode can be OFF or ON etc(two possible states). These two states represented by the symbols 0 & 1 respectively.

Counting in binary:

Easy way to remember to write a binary sequence of n bits is

- The rightmost column in the binary number begins with a 0 & alternates between 0 & 1.
- Second column begins with 2(=2¹) zeros & alternates between the groups of 2 zeros & 2 ones. So on

| Decimal no. | Binary no. | Decimal no. | Binary no. |
|-------------|------------|-------------|------------|
| 0 | 0 | 20 | 10100 |
| 1 | 1 | 21 | 10101 |
| 2 | 10 | 22 | 10110 |
| 3 | 11 | 23 | 10111 |
| 4 | 100 | 24 | 11000 |
| 5 | 101 | 25 | 11001 |
| 6 | 110 | 26 | 11010 |
| 7 | 111 | 27 | 11011 |
| 8 | 1000 | | |
| 9 | 1001 | | |
| 10 | 1010 | | |
| 11 | 1011 | | |
| 12 | 1100 | | |
| 13 | 1101 | | |
| 14 | 1110 | | |
| 15 | 1111 | | |
| 16 | 10000 | | |
| 17 | 10001 | | |
| 18 | 10010 | | |
| 19 | 10011 | 39 | 100111 |

Binary to Decimal Conversion:

It is by the positional weights method. In this method, each binary digit of the no. is multiplied by its position weight. The product terms are added to obtain the decimal no.

Example: convert 10101_2 to decimal

$$\begin{aligned} \text{Positional weights } & 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \\ \text{Binary no. } 10101_2 &= (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= 16 + 0 + 4 + 0 + 1 \\ &= 21_{10} \end{aligned}$$

Example: convert 11011.101_2 to decimal

$$\begin{aligned} \text{Positional weights } & 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \quad 2^{-1} \quad 2^{-2} \quad 2^{-3} \\ &= 16 + 8 + 0 + 2 + 1 + .5 + 0 + .125 \\ &= 27.625_{10} \end{aligned}$$

- An integer binary no. can also be converted to an integer decimal no as follows

- * Left bit MSB , multiply this bit by 2 & add the provided to next bit to the right
- * Multiply the result obtained in the previous step by 2 & add the product to the next bit to the right.

Exaple: 1001011_2

| | | | | | | |
|---|----------------|----------------|----------------|----------------|-----------------|-----------------|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| | 1×2^0 | 2×2^0 | 4×2^1 | 9×2^0 | 18×2^1 | 37×2^1 |
| | =2 | =4 | =9 | =18 | =37 | =75 |

Result= 75_{10}

Decimal to Binary conversion:

Two methods

There are reverse processes of the two methods used to convert a binary no. to a decimal no.

I method: is for small no.s The values of various powers of 2 need to be remembered. . for conversion of larger no.s have a table of powers of 2 known as the sum of weights method. The set of binary weight values whose sum is equal to the decimal no. is determined.

- To convert a given decimal integer no. to binary,
 - (1). Obtain largest decimal no. which is power of 2 not exceeding the remainder & record it
 - (2). Subtract this no. from the given no & obtain the remainder
 - (3). Once again obtain largest decimal no. which is power of 2 not exceeding this remainder & record it.
 - (4). Subtract through no. from the remainder to obtain the next remainder.
 - (5). Repeat till you get a "0" remainder

The sumof these powers of 2 expressed in binary is the binary equivalent of the original decimal no. similarly to convert fractions to binary.

II method: It converts decimal integer no. to binary integer no by successive division by 2 & the decimal fraction is converted to binary fraction by **double –dabble method**

Example: 163.875^{10} binary

Given decimal no. is mixed no.

So convert its integer & fraction parts separately.

Integer part is 163_{10}

The largest no. which is a power of 2, not exceeding 163 is 128.

$$128 = 2^7 = 10000000_2$$

remainder is $163 - 128 = 35$

The largest no., apower of 2 , not exceeding 35 is 32.

$$32=2^5=100000_2$$

remainder is $35-32=3$

The largest no., a power of 2, not exceeding 35 is 2.

$$2=2^1=10_2$$

Remainder is

$$3-2=1$$

$$1=2^0=1_2$$

$$163_{10}=10000000_2+100000_2+10_2+1_2=10100011_2$$

The fraction part is 0.875_{10}

1. The largest fraction, which is a power of 2, not exceeding 0.875 is 0.5

$$0.5=2^{-1}=0.100_2$$

Remainder is $0.875-.5=0.375_2$.

2. 0.375 is 0.25

$$0.25=2^{-2}=0.01_2$$

Remainder is $0.375-.25=0.125_2$.

3. 0.125 is 0.125 itself

$$0.125=2^{-3}=0.001_2$$

$$0.875_{10}=0.100_2+0.01_2+0.001_2=0.111_2$$

final result is

$$163.875_{10}=10100011.111_2$$

Example: convert 52_{10} to binary using double-dabble method

Divide the given decimal no successively by 2 & read the remainders upwards to get the equivalent binary no.

| Successive division | remainder | |
|---------------------|-----------|-----------------------|
| 2 52 | | |
| 2 26 --- | 0 | |
| 2 13 --- | 0 | |
| 2 6 --- | 1 | |
| 2 3 --- | 0 | ↓ |
| 2 1 --- | 1 | ↓ |
| 2 0 --- | 1 | ↓ |
| | | = 110100 ₂ |

Example: 0.75_{10} using double – dabble method

Multiply give fraction by 2
 Keep the integer in the product as it is & multiply the new fraction in the product
 by 2

| | | | |
|--------------------|------|---|--------------------|
| | 0.75 | | |
| Multiply 0.75 by 2 | 1.50 | ↓ | |
| Multiply 0.50 by 2 | 1.00 | ↓ | =0.11 ₂ |

Binary Addition:

Rules:

- 0+0=0
- 0+1=1
- 1+0=1
- 1+1=10 i.e, 0 with a carry of 1.

Example: add binary no.s 1101.101 & 111.011

| | | | |
|-----------|----------|----------|----------|
| 8421 | 2^{-1} | 2^{-2} | 2^{-3} |
| 1101.101 | | | |
| 111.011 | | | |
| 10101.000 | | | |

| | | |
|--------------------|---------|------------------------------------------|
| In 2^{-3} column | 1+1=0 | with a carry of 1 to the 2^{-2} column |
| In 2^{-2} column | 0+1+1=0 | 2^{-1} |
| 1 | 1+0+1=0 | 1's |
| 2 | 1+1+1=1 | 2's |
| 4 | 0+1+1=0 | 4's |
| 8 | 1+1+1=1 | 8's |
| 16 | 1+1 =0 | 16's |

Binary Subtraction:

- Rules:
- 0-0=0
 - 1-1=0
 - 1-0=1
 - 0-1=1 with a borrow of 1

Example: subtract binary no.s 111.1₂& 1010.01₂

| | | | |
|----------|----------|----------|----------|
| 8421 | 2^{-1} | 2^{-2} | 2^{-3} |
| 1010.010 | | | |
| 111.111 | | | |
| 0010.011 | | | |

In 2^{-3} column 10-1=1

| | | | |
|----------|---------|----------|------------------------|
| 2^{-2} | | $10-1=1$ | |
| 2^{-1} | | $1-1=0$ | |
| 1's | $1-1=0$ | | |
| 2's | | $10-1=1$ | |
| 4's | | $1-1=0$ | |
| 8's | $0-0=0$ | | result is 0010.011_2 |

Binary multiplication:

Two methods:

1. paper method
2. computer method

Rules:

- $0 \times 0 = 0$
- $1 \times 1 = 1$
- $1 \times 0 = 0$
- $0 \times 1 = 0$

Paper method:

1101_2 by 110_2

1011.101_2 by 101.01_2

```

      1101
      X110
      ----
      0000
      1101
      1101
      ----
    1001110
  
```

```

      1011.101
      x101.01
      ----
      1011101
      0000000
      1011101
      0000000
      1011101
      ----
    111101.00001
  
```

Computer method:

1100_2 by 1001_2

```

      MQ reg      10010000
      Shifted MQ left 10010000
      Add M      1100
      ----
      Partial sum in MQ 00101100
      Shift MQ left 001011000
      Add 0      0000
  
```

A1 shifted out so add
M to MQ

A 0 shifted out so add
0 to MQ

| | | |
|-------------------|-----------|-----------------------------------|
| Partial sum in MQ | 01011000 | |
| Shift MQ left | 010110000 | A 0 shifted out so add 0 to MQ |
| Add 0 | 0000 | |
| Partial sum in MQ | 101100000 | |
| Shift MQ left | 101100000 | A 1 shifted out so add M to MQ |
| Add M | 1100 | |
| Final sum in MQ | 01101100 | |

Binary Division:

Two methods:

1. paper method
2. computer method

Example : 101101_2 by 110

110) 101101 (111.1

| |
|------|
| 110 |
| 1010 |
| 110 |
| 1001 |
| 110 |
| 110 |
| 110 |
| 000 |

Ans: 111.1

Representation of signed no.s binary arithmetic in computers:

- Two ways of rep signed no.s
 1. Sign Magnitude form
 2. Complement 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.

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^n unique states.
5. Significant information is contained 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. By obtaining 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
Copy up to First 1 bit 1
Compliment remaining : 1101001

bits 11010011

Ex:

-73.75 in 12 bit 2's comp form

I method

01001001.1100
10110110.0011
+1

10110110.0100 is 2's

II method:

$2^8 = 100000000.0000$
Sub 73.75 = -01001001.1100

10110110.0100 is 2's comp

III method :

Originalno : 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

$$+87.5 = 01010111.1000$$

$$-45.75 = +11010010.0100$$

$$\begin{array}{r} \hline -41.75 \quad (1)00101001.1100 \text{ ignore carry} \\ \hline \text{MSB is 0, result is +ve. } = +41.75 \end{array}$$

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. from 1. This complemented value rep the -ve of the original no. One of the difficulties of using 1's comp is its rep of 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

$$\begin{array}{rcl} +99 & = & 01100011 \\ -99 & = & 10011100 \end{array}$$

$$\begin{array}{rcl} +77.25 & = & 01001101.0100 \\ -77.25 & = & 10110010.1011 \end{array}$$

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 1's comp form. Take its 1's comp to get the magnitude in binary.

Ex: Subtract 14 from 25 using 8 bit 1's

$$\begin{array}{rcl} 25 & = & 00011001 \\ -14 & = & 11110001 \\ \hline +11 & (1)00001010 & \\ & & +1 \\ \hline & & 00001011 \end{array}$$

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

$$= +11_{10}$$

EX: ADD -25 to +14

$$\begin{array}{rcl} +14 & = & 00001110 \\ -25 & = & +11100110 \\ \hline -11 & 11110100 & \end{array}$$

No carry MSB = 1
result = -ve = -11₁₀

Double precision no.s:

For any computer the word length is fixed. In a 16 bit computer, i.e., with a 16 bit word length, only no.s from $+2^{16-1} (+32,767)$ to $-2^{16-1} (-32,768)$ can be expressed in each register.

If no. is greater than this, two storage locations need to be used. i.e, each such no. has to be stored in two registers called Double Precision.

Leaving the MSB which is the sign bit, allows a 31 bit no. length with two 16 bit registers. If still larger no.s are to be expressed, there registers are used to store each no. called Triple Precision.

Floating Point NO.s:

In decimal system, very large & very small no.s expressed in scientific notation by stating a no. (mantissa) & an exponent of 10.

Binary no.s can be expressed in same notation by an exponent of 2.

| Mantissa | Exponent |
|------------|----------|
| 0110000000 | 100101 |

16 bit word contains two parts:10 bit mantissa , 6 bit exponent.i.e, in 2’s comp form & in that MSB is sign bit.

| | |
|-----------------|------------------------------------------------------------------------------|
| Mantissa = | +0.110000000 |
| Exponent= | 100101 |
| Actual exponent | = 100101- |
| | 100000=000101 |
| Entire no. =N= | +0.1100 ₂ x 2 ⁵ = 11000 ₂ =24 ₁₀ |

Many formats of floating pt.no.s.Someuse 2 words for mantissa, one for exponent .other use 2 & half words for mantissa & half for exponent.

Depending on the accuracy desired. some use excess n notation for the exponent, some use 2’s comp notation for mantissa &some use sign magnitude for both mantissa & exponent.

The Octal Number System:

It is used by early minicomputers. It is also a positional weights system. Its base or radix is 8.It has 8 independent symbols 0, 1,2,3,4,5,6,7. Since its base 8=2³, every 3-bit group of binary can be rep by an octal digit. An octal no. is, 1/3 rd the length of the corresponding binary no.

Octal to Binary conversion:

Just replace each octal digit by its 3 bit binary equivalent.

Ex:

367.52₈ to binary

Given octal no is 367.52

| | | | | | |
|-----|-----|-----|---|-----|-----|
| 3 | 6 | 7 | . | 5 | 2 |
| 011 | 110 | 111 | | 101 | 010 |

= 011110111.101010₂

Binary to Octal conversion:

Starting from the binary pt. make groups of 3 bits each, on either side of the binary pt, & replace each 3 bit binary group by the equivalent octal digit.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-----|-----|-----|-----|-----|--|---|---|---|---|---|----|-----|-----|-----|---|-----|---|-----|-----|-----|-----|---|-----|-----|---|---|---|---|---|---|---|
| <p>Ex:</p> <p style="text-align: center;">Convert 110101.101010₂ to octal</p> <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 10px;">Group of 3</td> <td style="padding: 0 10px;">110</td> <td style="padding: 0 10px;">101</td> <td style="padding: 0 10px;">.</td> <td style="padding: 0 10px;">101</td> <td style="padding: 0 10px;">010</td> </tr> <tr> <td></td> <td style="padding: 0 10px;">6</td> <td style="padding: 0 10px;">5</td> <td style="padding: 0 10px;">.</td> <td style="padding: 0 10px;">5</td> <td style="padding: 0 10px;">2</td> </tr> </table> <p style="text-align: center;">=65.52₈</p> <p>Ex:</p> <p style="text-align: center;">10101111001.0111₂</p> <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 10px;">10</td> <td style="padding: 0 10px;">101</td> <td style="padding: 0 10px;">111</td> <td style="padding: 0 10px;">001</td> <td style="padding: 0 10px;">.</td> <td style="padding: 0 10px;">011</td> <td style="padding: 0 10px;">1</td> </tr> <tr> <td style="padding: 0 10px;">010</td> <td style="padding: 0 10px;">101</td> <td style="padding: 0 10px;">111</td> <td style="padding: 0 10px;">001</td> <td style="padding: 0 10px;">.</td> <td style="padding: 0 10px;">011</td> <td style="padding: 0 10px;">100</td> </tr> <tr> <td style="padding: 0 10px;">2</td> <td style="padding: 0 10px;">5</td> <td style="padding: 0 10px;">7</td> <td style="padding: 0 10px;">1</td> <td style="padding: 0 10px;">.</td> <td style="padding: 0 10px;">3</td> <td style="padding: 0 10px;">4</td> </tr> </table> <p style="text-align: center;">=2571.34₈</p> | Group of 3 | 110 | 101 | . | 101 | 010 | | 6 | 5 | . | 5 | 2 | 10 | 101 | 111 | 001 | . | 011 | 1 | 010 | 101 | 111 | 001 | . | 011 | 100 | 2 | 5 | 7 | 1 | . | 3 | 4 |
| Group of 3 | 110 | 101 | . | 101 | 010 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 6 | 5 | . | 5 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 101 | 111 | 001 | . | 011 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 010 | 101 | 111 | 001 | . | 011 | 100 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 5 | 7 | 1 | . | 3 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Octal to decimal Conversion:

Multiply each digit in the octal no by the weight of its position & add all the product terms
Decimal value of the octal no.

$$d_n d_{n-1} d_{n-2} \dots\dots\dots d_1 d_0.d_{-1} d_{-2} d_{-3} \dots\dots\dots d_{-k} \text{ is } (d_n \times 8^n) + (d_{n-1} \times 8^{n-1}) + \dots\dots\dots (d_1 \times 8^1) + (d_0 \times 8^0) + (d_{-1} \times 8^{-1}) + (d_{-2} \times 8^{-2}) \dots\dots\dots$$

| |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Ex: convert 4057.06₈ to octal</p> $=4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$ $=2048 + 0 + 40 + 7 + 0 + 0.0937$ $=2095.0937_{10}$ |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Decimal to Octal Conversion:

To convert a mixed decimal no. To a mixed octal no. convert the integer and fraction parts separately. To convert decimal integer no. to octal, successively divide the given no by 8 till the quotient is 0. The last remainder is the MSD .The remainder read upwards give the equivalent octal integer no. To convert the given decimal fraction to octal, successively multiply the decimal fraction&the subsequent decimal fractions by 8 till the product is 0 or till the required accuracy is the MSD. The integers to the left of the octal pt read downwards give the octal fraction.

Ex: convert 378.93_{10} to octal

378_{10} to octal: Successive division:

$$\begin{array}{r}
 8 \mid 378 \\
 \hline
 8 \mid 47 \text{ --- } 2 \\
 \hline
 8 \mid 5 \text{ --- } 7 \quad \uparrow \\
 \hline
 0 \text{ --- } 5
 \end{array}$$

$$=572_8$$

0.93_{10} to octal :

$$\begin{array}{r}
 0.93 \times 8 = 7.44 \\
 0.44 \times 8 = 3.52 \quad \downarrow \\
 0.53 \times 8 = 4.16 \\
 0.16 \times 8 = 1.28 \\
 \hline
 =0.7341_8
 \end{array}$$

$$378.93_{10} = 572.7341_8$$

EX: 5497_{10} to binary

$$8 \mid 5497$$

$$\begin{array}{r}
 8 \mid 687 \text{ --- } 1 \\
 \hline
 8 \mid 85 \text{ --- } 7 \quad \uparrow \\
 \hline
 8 \mid 10 \text{ --- } 5 \\
 \hline
 8 \mid 1 \text{ --- } 2 \\
 \hline
 0 \text{ --- } 1 \quad \uparrow \quad =12571_8 = 001010101111001_2
 \end{array}$$

Conversion of large decimal no.s to binary & large binary no.s to decimal can be conveniently & quickly performed via octal

EX: 101111010001_2 to decimal

$$\begin{aligned}
 101111010001_2 &= 5721_8 = 5 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 \\
 &= 2560 + 448 + 16 + 1 = 3025_{10}
 \end{aligned}$$

Octal Arithmetic:

The rules are similar to the decimal or binary arithmetic. This no. system used to enter long strings of binary data in a digital system like a microcomputer. Arithmetic operations can be performed by converting the octal no.s to binary no.s & then using the rules of binary arithmetic. Octal subtraction can be performed using 1's complement method or 2's complement method & can also be performed directly by 7's & 8's complement methods of decimal system.

Ex: $5C7_{16}$ to decimal

$$\begin{aligned} & (5 \times 16^2) + (C \times 16^1) + (7 \times 16^0) \\ & = 1280 + 192 + 7 \\ & = 147_{10} \end{aligned}$$

Decimal to Hexadecimal conversion:

It is successively divide the given decimal no. by 16 till the quotient is zero. The last remainder is the MSB. The remainder read from bottom to top gives the equivalent hexadecimal integer. To convert a decimal fraction to hexadecimal successively multiply the given decimal fraction & subsequent decimal fractions by 16, till the product is zero. Or till the required accuracy is obtained, and collect all the integers to the left of decimal pt. The first integer is MSB & the integer read from top to bottom give the hexadecimal fraction known as **the hexadabble method**.

Ex: 2598.67510

$$\begin{array}{r} 16 \overline{) 2598} \\ 16 \overline{) 162} \quad -6 \\ 10 \quad -2 \end{array}$$

= $A26_{(16)}$

$$\begin{aligned} 0.67510 &= 0.675 \times 16 \quad \text{--} \quad 10.8 \\ &= 0.800 \times 16 \quad \text{--} \quad 12.8 \downarrow \\ &= 0.800 \times 16 \quad \text{--} \quad 12.8 \\ &= 0.800 \times 16 \quad \text{--} \quad 12.8 \end{aligned} \quad = 0.ACCC_{16}$$

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

Ex: 49056_{10}

| | decimal | hexa | binary |
|-----------------------------------------|---------------|------|--------|
| $16 \overline{) 49056}$ | | | |
| $16 \overline{) 3066} \quad \text{---}$ | 0 | 0 | 000 |
| $16 \overline{) 191} \quad \text{---}$ | 10 | A | 1010 |
| $16 \overline{) 11} \quad \text{---}$ | 15 \uparrow | F | 1111 |
| $0 \quad \text{---}$ | 11 | B | 1011 |

= $BFA0_{16} = 1011,1111,1010,0000_2$

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 |

Hexa decimal 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

Hexadecimal Arithmetic:

The rules for arithmetic is same as decimal octal & binary. Arithmetic operations are not done directly in hex. The hex no.s are first converted into binary & arithmetic operations are done in binary. Hex decimal subtraction can be performed using 1's compliment method or 2's compliment methods performed directly by 15's & 16's compliment methods. Similar to the 9's & 10's compliment of decimal system..

Ex:: Add $6E_{16}$ & $C5_{16}$

$$\begin{array}{r}
 6E_{16} = 0110\ 1110_2 \\
 C5_{16} = +1100\ 0101_2 \\
 \hline
 133_{16} \qquad 1010\ 100.001
 \end{array}$$

Subtract $7B_{16}$ from $C4_{16}$

$$\begin{array}{r}
 C4_{16} = 1100\ 0100_2 \\
 -7B_{16} = +100001\ 01_2 \\
 \hline
 49_{16} \quad (1)010\ 010\ 01_2 \\
 \text{Ignore carry ans: +ve.}
 \end{array}$$

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 ease 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 6₁₀(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 = 0110 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 1111 borrows are present | | | | | | | | |
| | | -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

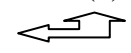
| | | | | | | | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------|----------------------|------|---|------|--------------------------|
| 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 168.8 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 |
| 1001 | | 339 | corrected difference in decimal |
| +1001 | | 1011 | 687 in xs-3 |
| | | 1000 | 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 |
| 0110 | 0110 | 1100 | adding 0011) |
| | | | 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 gray codes:

| 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 to Gray conversion:

N bit binary no is rep by $B_n B_{n-1} \dots B_1$

Gray code equivalent is by $G_n G_{n-1} \dots G_1$

B_n, G_n are the MSB's then the gray code bits are obtained from the binary code as

| | | | | | |
|-------------|--------------------------------|------------------------------------|-------|------------------------|--|
| $G_n = B_n$ | $G_{n-1} = B_n \oplus B_{n-1}$ | $G_{n-2} = B_{n-1} \oplus B_{n-2}$ | ----- | $G_1 = B_2 \oplus B_1$ | |
|-------------|--------------------------------|------------------------------------|-------|------------------------|--|

$\oplus \rightarrow$ EX-or symbol

Procedure: ex-or the bits of the binary no with those of the binary no shifted one position to the right . The LSB of the shifted no. is discarded & the MSB of the gray code no.is the same as the MSB of the original binaryno.

EX: 10001

- (a). Binary : $1 \oplus \rightarrow 0 \oplus \rightarrow 0 \oplus \rightarrow 1$
- Gray : 1 1 0 1
- (b). Binary: 1 0 0 1
- Shifted binary: 1 0 0 (1)
-
- 1 1 0 1 \rightarrow gray

Gray to Binary Conversion:

If an n bit gray no. is rep by $G_n G_{n-1} \dots G_1$

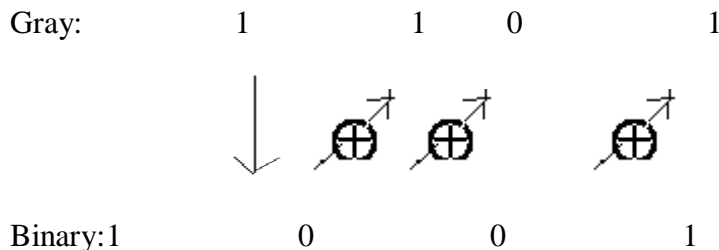
its binary equivalent by $B_n B_{n-1} \dots B_1$ then the binary bits are obtained from gray bits as

| | | | | |
|-------------|--------------------------------|------------------------------------|-------|------------------------|
| $B_n = G_n$ | $B_{n-1} = B_n \oplus G_{n-1}$ | $B_{n-2} = B_{n-1} \oplus G_{n-2}$ | ----- | $B_1 = B_2 \oplus G_1$ |
|-------------|--------------------------------|------------------------------------|-------|------------------------|

To convert no. in any system into given no. first convert it into binary & then binary to gray. To convert gray no into binary no & convert binary no into require no system.

Ex: $10110010(\text{gray}) = 11011100_2 = DC_{16} = 334_8 = 220_{10}$

EX:1101



Ex: $3A7_{16} = 0011,1010,0111_2 = 1001110100(\text{gray})$

$527_8 = 101,011,011_2 = 111110110(\text{gray})$

$652_{10} = 1010001100_2 = 1111001010(\text{gray})$

XS-3 gray code:

In a normal gray code , the bit patterns for 0(0000) & 9(1101) do not have a unit distance between them i.e, they differ in more than one position.In xs-3 gray code , each decimal digit is encoded with gray code patter of the decimal digit that is greater by 3. It has a unit distance between the patterns for 0 & 9.

XS-3 gray code for decimal digits 0 through 9

| Decimal digit | Xs-3 gray code | Decimal digit | Xs-3 gray code |
|---------------|----------------|---------------|----------------|
| 0 | 0010 | 5 | 1100 |
| 1 | 0110 | 6 | 1101 |
| 2 | 0111 | 7 | 1111 |
| 3 | 0101 | 8 | 1110 |
| 4 | 0100 | 9 | 1010 |

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 | data |
|-------|------------|-------|
| 10110 | 0 | 10110 |
| 10001 | 1 | 10001 |
| 10101 | 0 | 10101 |
| 00010 | 0 | 00010 |
| 11000 | 1 | 11000 |
| 00000 | 1 | 00000 |
| 11010 | 0 | 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 only correct 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 | For 12 bit code | For 7 bit code |
|----------------|-------------------------------------------------------------|-------------------------------------------------------------|----------------------------------------------|
| | C ₄ C ₃ C ₂ C ₁ | C ₄ C ₃ C ₂ C ₁ | 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 , 2^1 & 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 | For Excess-3 |
|---------------|----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| | P ₁ P ₂ D ₃ P ₄ D ₅ D ₆ D ₇ | P ₁ P ₂ D ₃ P ₄ D ₅ D ₆ D ₇ |
| 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

$P_1 P_2 D_3 P_4 D_5 D_6 D_7$

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→ $C_1=0$

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

Bits 4,5,6,7 (C_4 1001)) →no error →put a 0 in the 4's position→ $C_3=0$

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 | D_{10} | D_{11} | D_{12} | D_{13} | D_{14} | D_{15} |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|----------|----------|

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 | D_{10} | D_{11} | D_{12} |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|

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.

Boolean Algebra And Switching Functions

Boolean algebra:

Switching circuits called Logic circuits, gate circuits & digital circuits. Switching algebra called Boolean Algebra. Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of element (0,1) two binary operators called OR & AND & One unary operator NOT. Binary Digits 0 & 1 used to represent two voltage levels. Binary 1 is for high i.e, +5v . Binary 0 for Low i.e, 0v.

$A+A=A$ $A.A=A$ because variable has only a logic value.

Also there are some theorems of Boolean Algebra.

| | | |
|------------------------------------|----------------------------------------|-------------------|
| 1. (a) $A + A = A$ | (b) $A.A = A$ | Tautology Law |
| 2. (a) $A + 1 = 1$ | (b) $A.0 = 0$ | Union Law |
| 3. $(A')' = A$ | | Involution Law |
| 4. (a) $A + (B + C) = (A + B) + C$ | (b) $A.(B.C) = (A.B).C$ | Associative Law |
| 5. (a) $(A + B)' = A'B'$ | (b) $(A.B)' = A' + B'$ | De Morgan's Law |
| 6. (a) $A + AB = A$ | (b) $A(A + B) = A$ | Absorption Law |
| 7. (a) $A + A'B = A + B$ | (b) $A(A' + B) = AB$ | |
| 8. (a) $AB + AB' = A$ | (b) $(A + B)(A + B') = A$ | Logical adjacency |
| 9. (a) $AB + A'C + BC = AB + A'C$ | (b) $(A + B)(A' + C)(B + C) = (A + B)$ | Consensus Law |

Logic Operators:

AND,OR,NOT are 3 basic operations or functions that performed in Boolean Algebra. & derived operations as NAND , NOR,X-OR, X-NOR.

AXIOMS & Laws of Boolean Algebra:

Axioms or Postulates are a set of logical expressions i.e, without proof. & also we can build a set of useful theorems. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

| AND | OR | NOT |
|-------|-------|-------------|
| 0.0=0 | 0+0=0 | $\bar{1}=0$ |
| 0.1=0 | 0+1=1 | $\bar{0}=1$ |
| 1.0=0 | 1+0=1 | |
| 1.1=1 | 1+1=1 | |

Complementation Laws:

Complement means invert(0 as 1 & 1 as 0)

Law1: $\bar{\bar{0}}=1$

Law2: $\bar{\bar{1}}=0$

Law3:If $A=0$ then $\bar{A}=1$

Law4:If $A=1$ then $\bar{A}=0$

Law5: $\bar{\bar{A}}=A$ (double complementation law)

AND laws:

Law 1: $A.0=0$ (Null law)

Law 2: $A.1=A$ (Identity law)

Law 3: $A.A=A$

Law 4: $A.\bar{A}=0$

OR laws:

Law 1: $A+0=A$ (Null law)

Law 2: $A+1=1$

Law 3: $A+A=A$

Law 4: $A+\bar{A}=1$

Commutative laws: allow change in position of AND or OR variables.

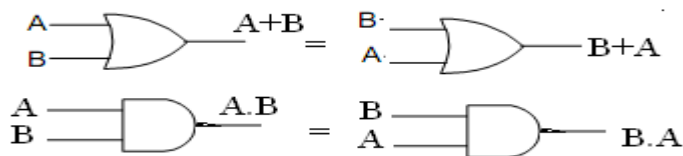
2 commutative laws

Law 1: $A+B=B+A$

Law 2: $A.B=B.A$

| A | B | A+B | = | B | A | B+A |
|---|---|-----|---|---|---|-----|
| 0 | 0 | 0 | | 0 | 0 | 0 |
| 0 | 1 | 1 | | 0 | 1 | 1 |
| 1 | 0 | 1 | | 1 | 0 | 1 |
| 1 | 1 | 1 | | 1 | 1 | 1 |

| A.B | B.A |
|-----|-----|
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 1 | 1 |

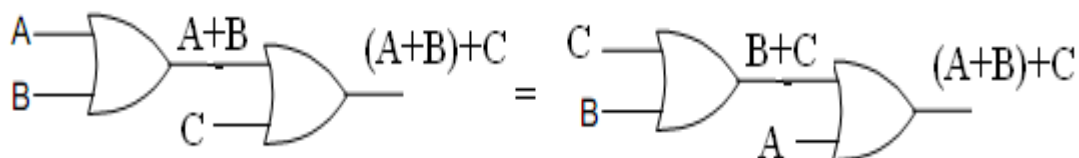


Associative laws: This allows grouping of variables. It has 2 laws.

Law 1: $(A+B)+C=A+(B+C)$ = A OR B ORed with C

This law can be extended to any no. of variables

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



| A | B | C | A+B | (A+B)+C |
|---|---|---|-----|---------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

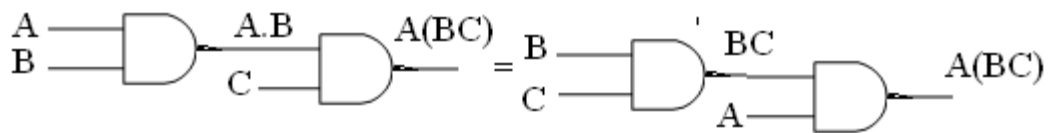
=

| A | B | C | B+C | A+(B+C) |
|---|---|---|-----|---------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Law2: $(A.B).C=A(B.C)$

This law can be extended to any no. of variables

$(A.B.C).D=(A.B.C).D$



| A | B | C | AB | (AB)C |
|---|---|---|----|-------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

=

| A | B | C | BC | A(BC) |
|---|---|---|----|-------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Distributive Laws:

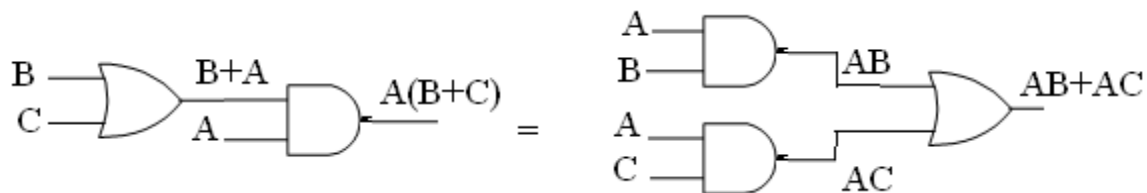
This has 2 laws

Law 1: $A(B+C)=AB+AC$

This law applies to single variables.

EX: $ABC(D+E)=ABCD+ABCE$

$AB(CD+EF)=ABCD+ABEF$

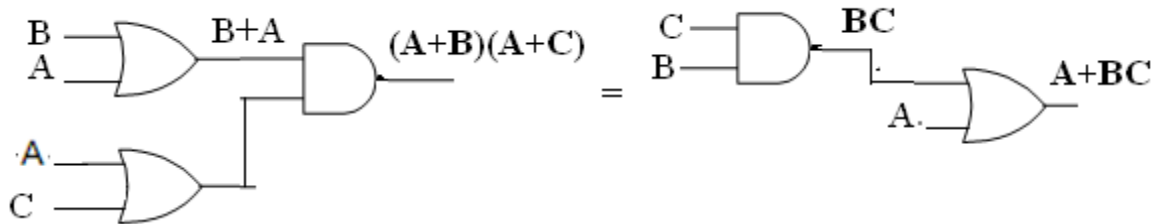


| A | B | C | B+C | A(B+C) |
|---|---|---|-----|--------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

| A | B | C | AB | AC | AB+AC |
|---|---|---|----|----|-------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

Law 2. $A+BC=(A+B)(A+C)$

$$\begin{aligned}
 \text{RHF} &= (A+B)(A+C) \\
 &= AA+AC+BA+BC \\
 &= A+AC+AB+BC \\
 &= A(1+C+B)+BC \\
 &= A.1+BC \\
 &= A+BC \quad \text{LHF}
 \end{aligned}$$



| A | B | C | BC | A+BC |
|---|---|---|----|------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

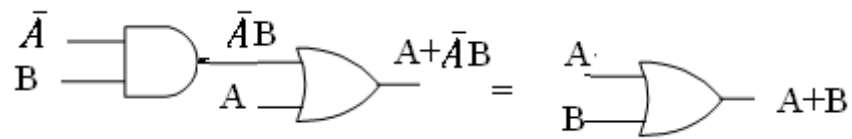
| A | B | C | A+B | A+C | (A+B)(A+C) |
|---|---|---|-----|-----|------------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

Redundant Literal Rule(RLR):

Law 1: $A+\bar{A}B=A+B$

$$\begin{aligned}
 \text{LHF} &= (A+\bar{A})(A+B) \\
 &= 1.(A+B) \\
 &= A+B \quad \text{RHF}
 \end{aligned}$$

ORing of a variable with the AND of the compliment of that variable with another variable, is equal to the ORing of the two variables.



| A | B | $\bar{A}B$ | $A+\bar{A}B$ |
|---|---|------------|--------------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |

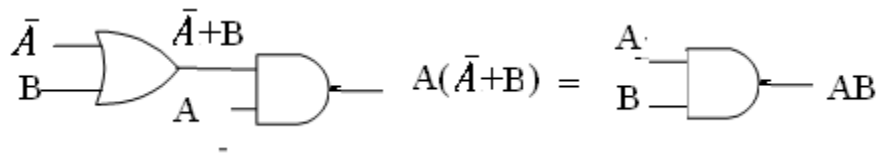
=

| A | B | A+B |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

Law 2: $A(\bar{A} + B) = AB$

$$\begin{aligned} \text{LHF} &= A\bar{A} + AB \\ &= 0 + AB \\ &= AB \quad \text{RHF} \end{aligned}$$

ANDing of a variable with the OR of the complement of that variable with another variable, is equal to the ANDing of the two variables.



| A | B | $\bar{A} + B$ | $A(\bar{A} + B)$ |
|---|---|---------------|------------------|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

=

| A | B | A+B |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

Idempotence Laws:

Idempotence means same value. It has 2 laws.

Law 1 = $A.A = A$

This law states that ANDing of a variable with itself is equal to that variable only.



If $A=0$, then $A.A=0.0=0=A$
 If $A=1$, then $A.A=1.1=1=A$

Law 2 = $A+A = A$

This law states that ORing of a variable with itself is equal to that variable only.



If $A=0$, then $A+A=0+0=0=A$

If $A=1$, then $A+A=1+1=1=A$

Absorption Laws:

Law 1 $= A + A \cdot B = A$

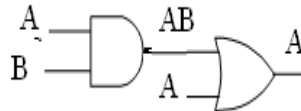
$$= A(1+B)$$

$$= A \cdot 1$$

$$= A$$

i.e., $A + A \cdot \text{any term} = A$

| A | B | AB | $A+AB$ |
|---|---|------|--------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |



Law 2 $= A(A+B) = A$

$A(A+B) = A \cdot A + A \cdot B$

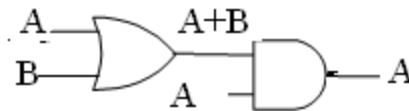
$$= A + AB$$

$$= A(1+B)$$

$$= A \cdot 1$$

$$= A$$

| A | B | $A+B$ | $A(A+B)$ |
|---|---|-------|----------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |



Consensus theorem:

Theorem 1: $AB + \bar{A}C + BC = AB + \bar{A}C$

LHS: $AB + \bar{A}C + BC$

$$= AB + \bar{A}C + BC(A + \bar{A})$$

$$= AB + \bar{A}C + BCA + BC\bar{A}$$

$$= AB(1+C) + \bar{A}C(1)$$

$$= AB + \bar{A}C$$

RHS

This can be extended to any no. of variables

EX: $AB + \bar{A}C + BCD = AB + \bar{A}C$

Theorem 2: $(A+B)(\bar{A} + C)(B+C) = (A+B)(\bar{A}+C)$

Transposition Theorem:

$AB + \bar{A}C = (A+C)(\bar{A}+B)$

RHS: $(A+C)(\bar{A}+B)$
 $= A\bar{A} + C\bar{A} + AB + CB$
 $= 0 + \bar{A}C + AB + BC$
 $= \bar{A}C + AB + BC(A + \bar{A})$
 $= AB + ABC + \bar{A}C + \bar{A}BC$
 $= AB + \bar{A}C$

LHS

DeMorgans Theorem:

It represents two of the most powerful laws in Boolean algebra

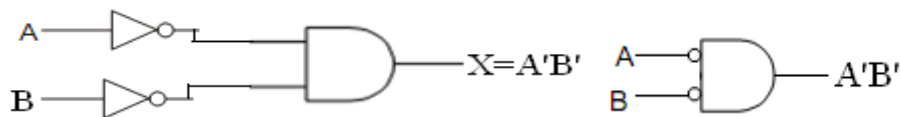
Law 1: $\overline{A + B} = \bar{A}\bar{B}$

This law states that the compliment of a sum of variables is equal to the product of their individual complements.

LHS



RHS



NOR gate

Bubbled AND gate

| A | B | A+B | (A+B)' |
|---|---|-----|--------|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

| A | B | A' | B' | A'B' |
|---|---|----|----|------|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |

NOR gate= Bubbled AND gate

This can be extended to any variables.

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

Law 2: $(AB)' = A' + B'$

Complement of the product of variables is equal to the sum of their individual components.

| A | B | $(AB)'$ |
|---|---|---------|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| A | B | A' | B' | $A'+B'$ |
|---|---|------|------|---------|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |

This law also can extend to any no. Of variables.

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

It can be extended to complicated expressions by

1. Complement the entire function
2. Change all the ANDs to ORS and all the ORs to ANDS
3. Complement each of the individual variables.
4. Change all 0s to 1s and 1s to 0s.

This procedure is called demorganization or complementation of switching expressions.

Shannon's expansion Theorem:

This theorem states that any switching expression can be decomposed w.r.t. a variable A into two parts, one containing A & other containing A'. It is useful in decomposing complex machines into an interconnection of smaller components.

$$f(A,B,C\dots) = A.f(1,B,C\dots) + A'.f(0,B,C\dots)$$

$$f(A,B,C,\dots) = [A+f(0,B,C,\dots)].[A'+f(1,B,C,\dots)]$$

Ex: DeMorganize $f = ((A+B')(C+D'))'$, $f = ((A+B')(C+D'))'$

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

$$= A.B' + C.D'$$

$$= A'.B + C'.D$$

Duality:

In a positive Logic system the more positive of the two voltage levels is represented by a 1 & the more negative by a 0. In a negative logic system the more positive of the two voltage levels is represented by a 0 & more negative by a 1. This distinction between positive & negative logic systems is important because an OR gate in the positive logic system becomes an AND gate in the negative logic system & vice versa. Positive & Negative logics give a basic duality in Boolean identities. Procedure dual identity by changing all '+' (OR) to '.' (AND) & complementing all 0's & 1's. Once a theorem or statement is proved, the dual also thus stands proved called Principle of duality.

$$[f(A,B,C, \dots, 0, 1, +, \dots)]_d = f(A,B,C, \dots, 1, 0, \dots, +)$$

Relations between complement

$$(f_c(A,B,C, \dots)) = \overline{(f_c(A, B, C, \dots))} = (f_d(\bar{A}, \bar{B}, \bar{C}, \dots))$$

$$(f_d(A,B,C, \dots)) = \overline{(f_d(A, B, C, \dots))} = (f_c(\bar{A}, \bar{B}, \bar{C}, \dots))$$

Duals:

| Expression | Dual |
|------------------------------------------|---------------------------------|
| $\bar{0}=1$ | $\bar{1}=0$ |
| $0.1=0$ | $1+0=1$ |
| $0.0=0$ | $1+1=1$ |
| $1.1=1$ | $0+0=0$ |
| $A.0=0$ | $A+1=1$ |
| $A.1=A$ | $A+0=A$ |
| $A.A=A$ | $A+A=A$ |
| $A.\bar{A}=0$ | $A+\bar{A}=1$ |
| $A.B=B.A$ | $A+B=B+A$ |
| $A.(B.C)=(A.B).C$ | $A+(B+C)=(A+B)+C$ |
| $A.(B+C)=(AB+AC)$ | $A+BC=(A+B)(A+C)$ |
| $A(A+B)=A$ | $A+AB=A$ |
| $A.(A.B)=A.B$ | $A+A+B=A+B$ |
| $\overline{AB}=\bar{A}+\bar{B}$ | $\overline{A+B}=\bar{A}\bar{B}$ |
| $(A+B)(\bar{A}+C)(B+C)=(A+B)(\bar{A}+C)$ | $AB+\bar{A}C+BC=AB+\bar{A}C$ |

Reducing Boolean Expressions:

Procedure:

1. Multiply all variables necessary to remove parenthesis

2. Look for identical terms. Only one of those terms to be retained & other dropped.

Ex: $AB+AB+AB+AB=AB$

4. Look for a variable & its negation in the same term. This term can be dropped 1

Ex: $ABC\bar{D}+ABC\bar{C}=ABC(\bar{D}+1)=ABC.1=ABC$

5. Look for pairs of terms which have the same variables, with one or more variables complemented. If a variable in one term of such a pair is complemented while in the second term it is not then such terms can be combined into a single term with variable dropped.

Ex: $ABC\bar{D}+ABC\bar{D}=ABC(\bar{D}+D)=ABC.1=ABC$ unctions

Boolean functions & their representation:

A function of n Boolean variables denoted by $f(x_1, x_2, x_3, \dots, x_n)$ is another variable denoted by & takes one of the two possible values 0 & 1.

The various way of represent a given function is

1. Sum of Product(SOP) form:

It is called the Disjunctive Normal Form(DNF)

Ex: $f(A,B,C)=(\bar{A}B+\bar{B}C)$

2. Product of Sums (POS) form:

It is called the Conjunctive Normal Form(CNF). This is implemented using Consensus theorem.

Ex: $f(A,B,C)=(\bar{A}+B)(B+C)$

3. Truth Table form:

The function is specified by listing all possible combinations of values assumed by the variables & the corresponding values of the function.

Truth table for $f(A,B,C)=(\bar{A}B+\bar{B}C)$

| Decimal Code | A | B | C | F(A,B,C) |
|--------------|---|---|---|----------|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 |

4. Standard Sum of Products form: Called Disjunctive Canonical form (DCF) & also called Expanded SOP form or Canonical SOP form.

$$f(A,B,C)=(\bar{A}B+\bar{B}C)=\bar{A}B(C+\bar{C})+\bar{B}C(A+\bar{A})$$

$$=\bar{A}\bar{B}C+\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}C$$

A Product term contains all the variables of the function either in complemented or Uncomplemented form is called a minterm. A minterm assumes the value 1 only for one combination of the variables. An n variable function can have in all 2^n minterms to 1 is the standard sum of products form of the function. Min terms are denoted as m_0, m_1, m_2, \dots . Here suffixes are denoted by the decimal codes.

Ex: 3 variable functions

$$m_0=\bar{A}\bar{B}\bar{C}$$

$$m_1=\bar{A}\bar{B}C$$

$$m_2=\bar{A}B\bar{C}$$

$$m_3=\bar{A}BC$$

|

$m_7 = CBA$ no other way of representation in canonical SOP form is , the SUM of minterms for which the function equals 1. Thus

$$f(A,B,C)=m_1+m_2+m_3+m_5$$

The function in DCF is listing the decimal codes of the minterms for which $f=1$

$$f(A,B,C)=\sum m(1,2,3,5).$$

5. Standard Product of Sums form: It is called as Conjunctive Canonical form (CCF). It is also called Expanded POS or Canonical POS.

If $\bar{A}=0$ ($A=1$) $B=0$ $C=0$, term=0

Thus function $f(A, B, C) = (\bar{A} + \bar{B})(A + B)$ given by POS

$$f(A,B,C)=(\bar{A} + \bar{B} + C\bar{C})(A+B+C\bar{C})$$

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

A sum term which contains each of the n variables in either complemented form is called a *Maxterm*. A maxterm assumes the value '0' only for one combination of the variables. The most there are 2^n maxterms. It is represented as M_0, M_1, M_2, \dots . Here the suffixes are decimal codes.

The CCF of $f(A,B,C)=M_0.M_4.M_6.M_7$

$$f(A,B,C)=\pi M(0,4,6,7)$$

π or \wedge represents the product of all maxterms.

6. Octal designation:

| m_7m_6 | m_5 | m_4 | m_3 | m_2 | m_1 | m_0 | |
|----------|-------|-------|-------|-------|-------|-------|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

7. Karnaugh Map:

Put the Truth Table in a compact form by labeling the row & columns of a map. It is used in the minimization of functions 3,4,5,6 variables.

m_0, m_1, m_2 ----- are minterms

M_0, M_1, M_2, M_3 ----- are Maxterms.

Expansion of a Boolean expression in SOP form to the standard SOP form:

1. Write down all the terms.
 2. If one or more variables are missing in any term. Expand that term by multiplying it with the sum of each one of the missing variable and its complement.
 3. Drop out redundant terms.
- * Interms of minterms:

1. Write down all the terms.
2. Put Xs in terms where variables must be inserted to form a minterm.
3. Replace the non-complemented variables by 1s and the complemented variables by 0s, and use all combinations of Xs in terms of 0s and 1s to generate minterms.
4. Drop out redundant terms.

Expansion of a Boolean expression in POS form to standard POS form:

1. Write down all the terms.
2. If one or more variables are missing in any sum term. expand that term by adding the product of each of the missing variable and its complement.
3. Drop out redundant terms.
- Interms of Maxterms:
 1. Write down all the terms.
 2. Put x's in terms where variable inserted
 3. Replace complemented variable by 1's & non complemented variable by 0's. & use all combinations.
 4. Drop out redundant terms.

Conversion between Canonical form:

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function is expressed by those minterms that make the function equal to 1 for those minterms that make the function equal to 0.

$$\text{Ex: } f(A,B,C)=\sum m(0,2,4,6,7)$$

Complement is

$$\overline{f(A,B,C)}=\sum m(1,3,5)=m_1+m_3+m_5$$

complement of \overline{f} by deMorgans theorem

$$f = \overline{(m_1 + m_3 + m_5)} = \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_5} = M_1 M_3 M_5 = \prod M(1,3,5)$$

$\overline{m_j} = M_j$, the maxterm with subscript j is a complement of the minterm with the same subscript j and vice versa. To convert one canonical form to another, interchange the symbol \sum and \prod , and list those numbers missing from the original form.

Computation of total gate inputs:

The total number of gate inputs required to realize a Boolean expression is computed as, If the expression is in the SOP form, count the number of AND inputs and number of AND gates feeding the OR gate. If the expression is in the POS form, count the number of OR inputs and the number of OR gates feeding the AND gate. If it is in hybrid form, count the gate inputs and the gates feeding other gates. The cost of implementing circuit is proportional to no. of gate inputs required.

$$\text{EX: } ABC + A\overline{B}CD + E\overline{F} + AD$$

- | | |
|----------------------------------------|------------|
| 1. Count the AND Inputs | 3+4+2+2=11 |
| 2. Count AND gates feeding the OR gate | 1+1+1+1=4 |
| 3. Total gate inputs | =15 |

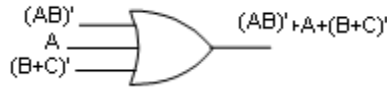
Boolean Expression & Logic Diagrams:

Boolean expressions can be realized as hardware using logic gates. Conversely, hardware can be translated into Boolean expressions for the analysis of existing circuits.

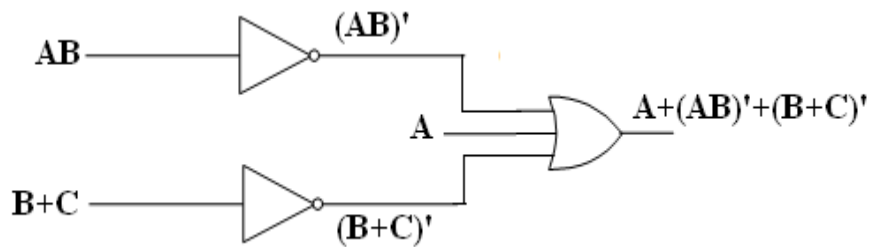
1. Converting Boolean Expressions to Logic:

To convert, start with the output & work towards the input.

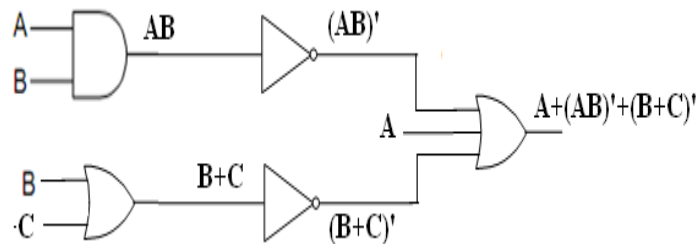
Assume the expression $\overline{AB} + A + \overline{B+C}$ is to be realized using AOI logic. Start with this expression. Since it is three terms, it must be the output of a three-input OR gates. So, draw an OR gate with three inputs as



$(AB)'$ is the output of an inverter whose inputs is AB and $(B+C)'$ must be the output of an inverter whose input is $B+C$. so, those two inverters are as



Now AB must be output of a two-input AND gate whose inputs are A and B . And $B+C$ must be the output of a two-input OR gate whose inputs are B and C . so, an AND gate and an OR gate are as

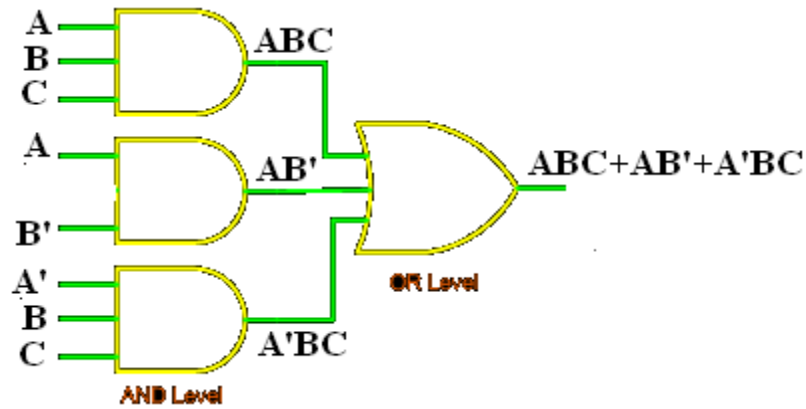


2. Converting Logic to Boolean Expressions:

To convert logic to algebra, start with the input signals and develop the terms of the Boolean expression until the output is reached.

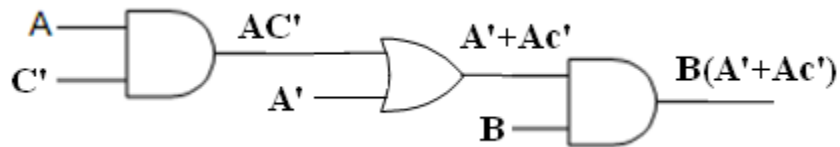
Converting AND/OR/INVERT logic to NAND/NOR logic:

1. The SOP expression $ABC + AB' + A'BC$ can be implemented in AND/ OR logic as



The POS expression $(A+B+C)(A+B')(A'+B+C)$ can be implemented using OR and AND gates

The expression $ABC' + A'B = B(A' + AC')$ can be implemented in hybrid form as



Hybrid Logic reduces the no. of gate inputs required for realization (from 7 to 6 in this case), but results in multilevel logic. Different inputs pass through number of gates to reach the output. It leads to non-uniform propagation delay between different numbers of gates to give rise to logic race. The SOP and POS realizations give rise to two-level logic. The two-level logic provides uniform time delay between input and outputs, because each input signal has to pass through two gates to reach the output. So, it does not suffer from the problem of logic race.

Since NAND logic and NOR logic are universal logic circuits which are first computed and converted to AOI logic may then be converted to either NAND logic or NOR logic depending on the choice. The procedure is

1. Draw the circuit in AOI logic
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the AND gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates
4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram
5. Replace bubbled OR by NAND and bubbled AND by NOR
6. Eliminate double inversions.

LOGIC GATES: Logic gates are fundamental building blocks of digital systems. Logic gate produces one output level when some combinations of input levels are present. & a different output level when other combination of input levels is present. In this, 3 basic types of gates are there. AND OR & NOT

The interconnection of gates to perform a variety of logical operation is called *Logic Design*. Inputs & outputs of logic gates can occur only in two levels. 1,0 or High, Low or True, False or On, Off. A table which lists all the possible combinations of input variables & the corresponding outputs is called a Truth Table. It shows how the logic circuits output responds to various combinations of logic levels at the inputs. *Level Logic*, a logic in which the voltage levels represent logic 1 & logic 0. Level logic may be Positive Logic or Negative Logic. In *Positive Logic* the higher of two voltage levels represent logic 1 & Lower of two voltage levels represent logic 0. In *Negative Logic* the lower of two voltage levels represent logic 1 & higher of two voltage levels represent logic 0.

In TTL (Transistor-Transistor Logic) Logic family voltage levels are +5v, 0v. Logic 1 represent +5v & Logic 0 represent 0v.

AND Gate:

It is represented by ‘.’(dot) It has two or more inputs but only one output. The output assume the logic 1 state only when each one of its inputs is at logic 1 state. The output assumes the logic 0 state even if one of its inputs is at logic 0 state. The AND gate is also called an All or Nothing gate.

Boolean Expression: $A \cdot B = Y$
A and B



Logic Symbol

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Truth Table

- IC 7408 contains 4 two input AND gates
- IC 7411 contains 3 three input AND gates
- IC 7421 contains 2 four input AND gates

OR Gate:

It is represented by '+' (plus) It has two or more inputs but only one output. The output assumes the logic 1 state only when one of its inputs is at logic 1 state. The output assumes the logic 0 state even if each one of its inputs is at logic 0 state. The OR gate is also called an any or All gate. Also called an inclusive OR gate because it includes the condition both the inputs can be present.



Logic Symbol

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Truth Table

Boolean Expression:

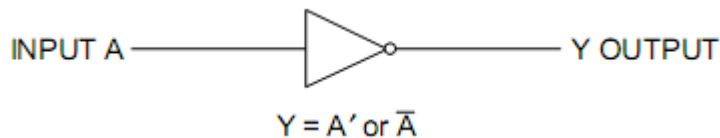
$$A \text{ OR } B \\ A + B = Y$$

IC 7432 Contains 4 two input OR gates.

NOT Gate:

It is represented by '-' (bar). It is also called an *Inverter or Buffer*. It has only one input & one output. Whose output always the compliment of its input? The output assumes logic 1 when input is logic 0 & output assume logic 0 when input is logic 1.

Logic Symbol



Truth Table

| A | X |
|---|---|
| 1 | 0 |
| 0 | 1 |

Boolean Expression:

$$X = A'$$

Logic circuits of any complexity can be realized using only AND, OR, NOT gates. Using these 3 called AND-OR-INVERT i.e, AOI Logic circuits.

The Universal Gates:

The universal gates are NAND, NOR. Each of which can also realize Logic Circuits Single handedly. NAND-NOR called Universal Building Blocks.. Both NAND-NOR can perform all the three basic logic functions. AOI logic can be converted to NAND logic or NOR logic.

NAND Gate:

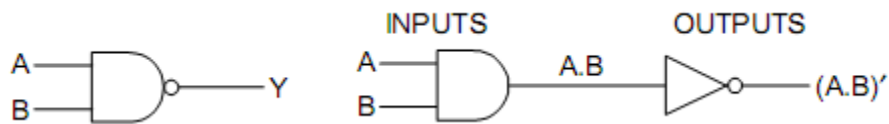
NAND gate mean NOT AND i.e, AND output is NOTed.
 NAND → AND & NOT gates

Boolean Expression:

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

= A .B.C whole bar.

NAND assumes Logic 0 when each of inputs assume logic 1.



Logic Symbol

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

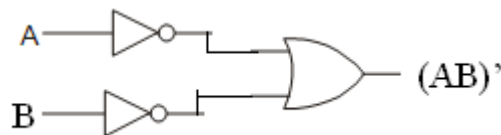
Truth table

Bubbled OR gate: The output of this is same as NAND gate.

Bubbled OR gate is OR gate with inverted inputs.

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

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



Truth Table

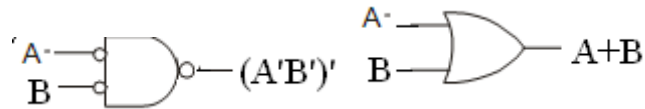
Logic Symbol

- NAND gate as an Inverter.

All its input terminals together & applying the signal to be inverted to the common terminal by connecting all input terminals except one to logic 1 & applying the signal to be inverted to the remaining terminal. It is also called Controlled Inverter.



Bubbled NAND Gate:

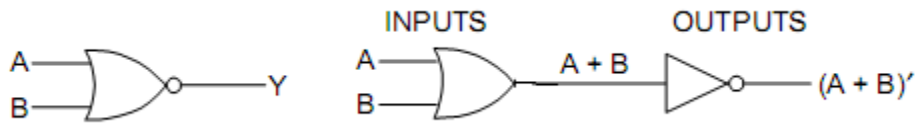


NOR Gate:

NOR gate is NOT gate with OR gate. i.e, OR gate is NOTed.

Boolean expression:

$$X = \overline{A + B + C + \dots}$$



Logic Symbol

Logic symbol with OR and NOT

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

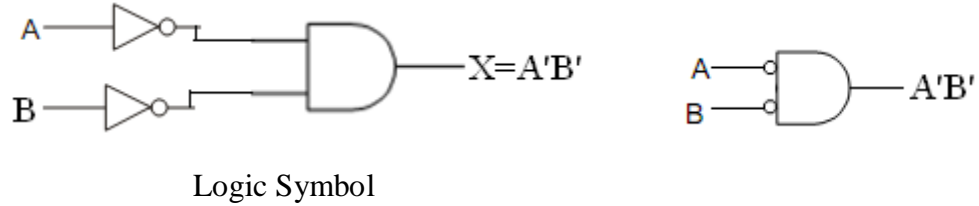
Truth Table

Bubbled AND gate:

is AND gate with inverted inputs. The AND gate with inverted inputs is called a bubbled And gate. So a NOR gate is equivalent to a bubbled and gate. A bubbled AND gate is also called a negative AND gate. Since its output assumes the HIGH state only when all its

inputs are in LOW state, a NOR gate is also called active-LOW AND gate. Output Y is 1 only when both A & B are equal to 0. i.e., only when both A' and B' are equal to 1.

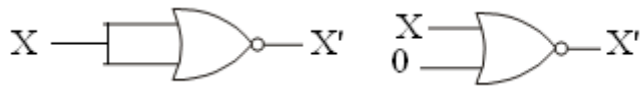
NOR can also be realized by first inverting the inputs and ANDing those inverted inputs.



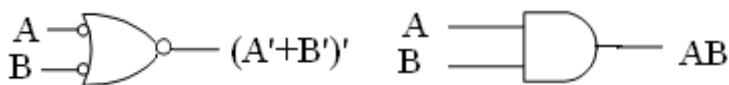
| Inputs A B | | Inverted Inputs A' B' | Output Y |
|---------------|---|-----------------------------|-------------|
| 0 | 0 | 1 1 | 1 |
| 0 | 1 | 1 0 | 0 |
| 1 | 0 | 0 1 | 0 |
| 1 | 1 | 0 0 | 0 |

NOR gate as an inverter:

is tying all input terminals together & applying the signal to be inverted to the common terminals or all inputs set as logic 0 except one & applying signal to be inverted to the remaining terminal.



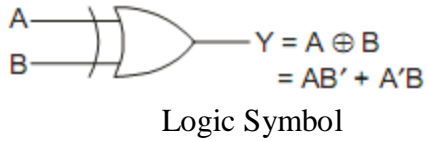
Bubbled NOR Gate: is AND gate.



- IC 7402 is 4 two input NOR gate
- IC 7427 is 3 three input NOR gate
- IC 7425 is 2 four input NOR gate

The Exclusive OR (X-OR) gate:

It has 2 inputs & only 1 output. It assumes output as 1 when input is not equal called *anti-coincidence gate* or *inequality detector*.



| A | B | $A \oplus B$ |
|---|---|--------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Proof:

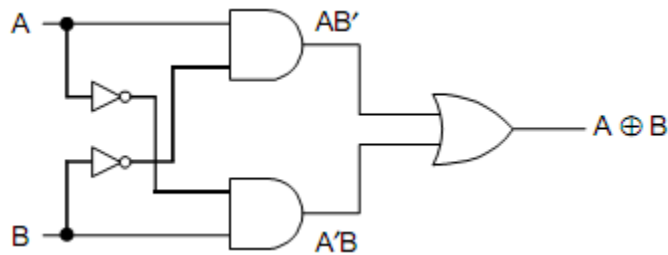
Truth Table

$$\begin{aligned}
 Y &= (A \oplus B) \oplus C \\
 &= (AB' + A'B) \oplus C \\
 X &= AB' + A'B
 \end{aligned}$$

The high outputs are generated only when odd number of high inputs is present. This is why x-or function also known as odd *function*.



The X-OR gate using AND-OR-NOT gates:



X-OR gate as an Inverter:

By connecting one of two input terminals to logic 1 & feeding the sequence to be inverted to other terminal

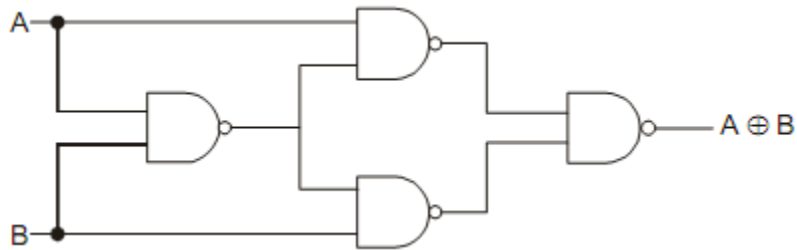


Logic Symbol

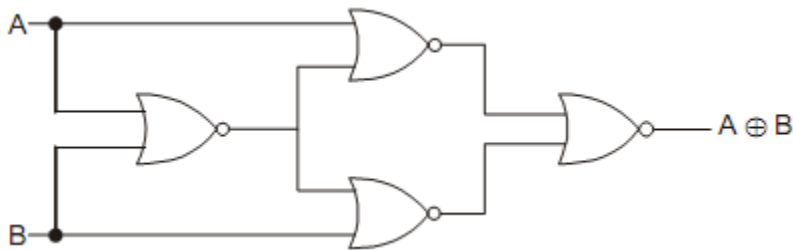
TTL IC 746 has 4 x-OR gate

CMOS IC 74C8C has 4 X-OR gates.

X-OR gate using NAND gates only:



X-OR gate using NOR gates only:



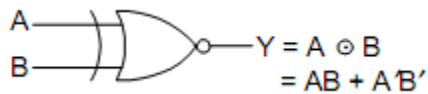
The EX-NOR Gate:

It is X-OR gate with a NOT gate. It has two inputs & one output logic circuit. It assumes output as 0 when one if inputs are 0 & other 1. It can be used as an equality detector because it outputs a 1 only when its inputs are equal.

$$X = A \odot B = AB + A'B' = (A \oplus B)' = (AB' + A'B)'$$

$$\begin{aligned} A \odot B &= (A \oplus B)' \\ &= (AB' + AB)' \\ &= (A' + B) \cdot (A + B') \\ &= AA' + A'B' + AB + BB' \\ &= AB + A'B' \end{aligned}$$

Proof:

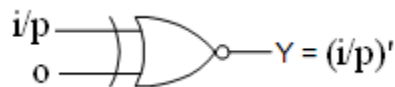


| Inputs A B | Output X = A ⊙ B |
|---------------|---------------------|
| 0 0 | 1 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

Logic Symbol.

X-NOR gate as an inverter:

by connecting one of 2 input terminals to logic 0 & feeding the input sequence to be inverted to the other terminal.



Logic Symbol as an inverter

| | |
|----------|-------------------------|
| i/p 0 | o/p $0 \oplus 0 = 1$ |
| i/p 1 | o/p $1 \oplus 0 = 0$ |

It can be used as Controlled inverter.

$$A \oplus B = (A \oplus B)'$$
 is compliment of X-OR

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

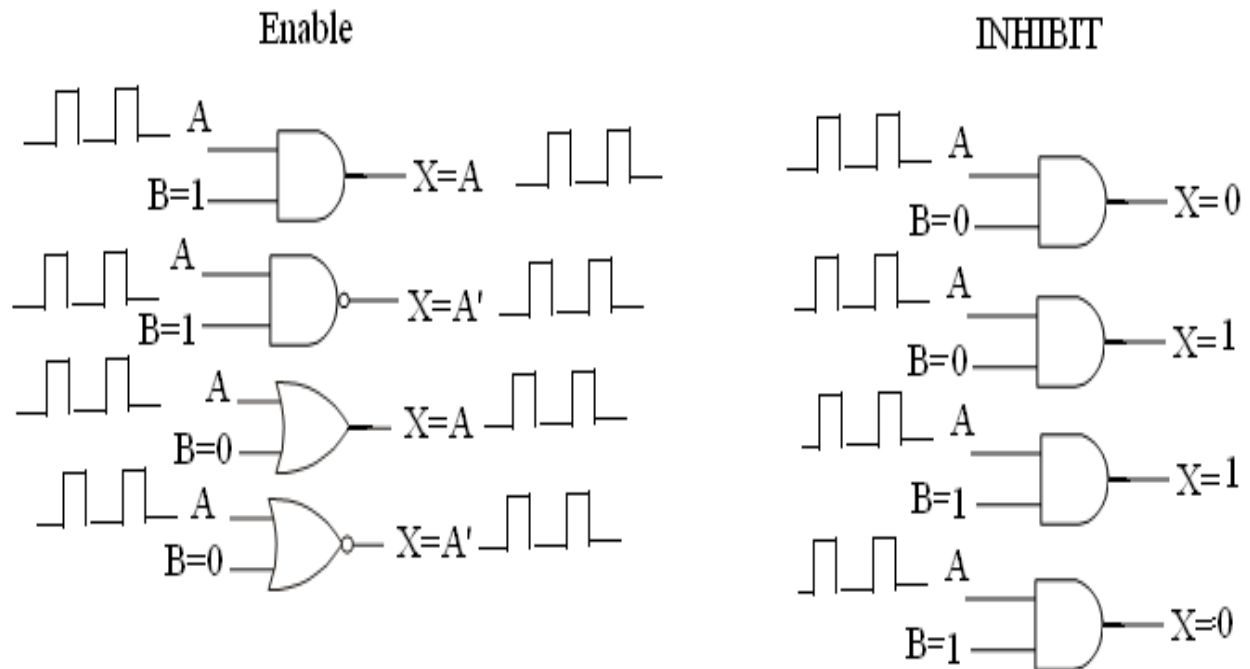
TTI IC74LS266 contain 4 each X-NOR gates.

CMOS 74C266 contain 4 each X-NOR gates.

Highspeed CMOS IC 74HC266 contain 4 each X-NOR gates.

INHIBIT CIRCUITS:

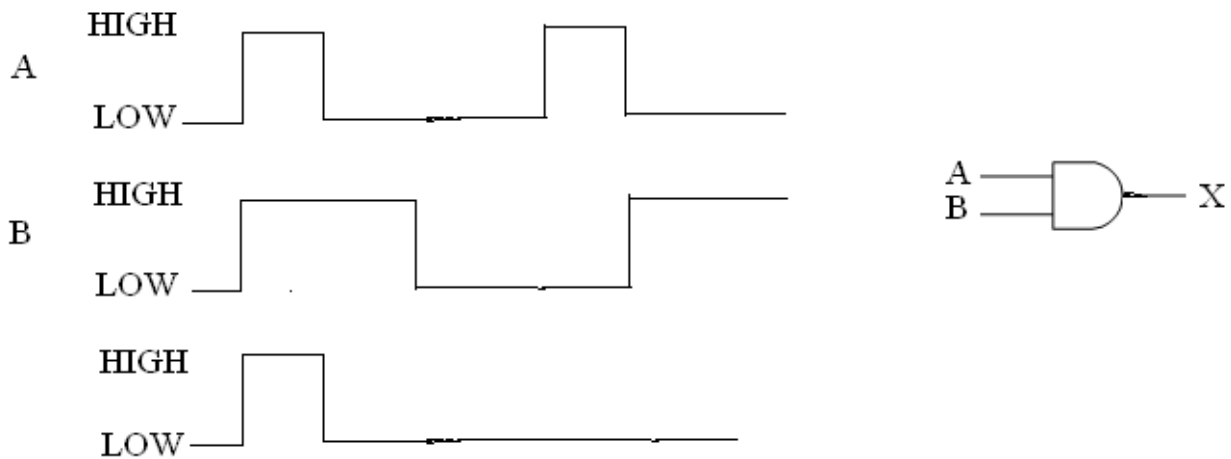
AND , OR , NAND , NOR gates can be used to control the passage of an input logic signal through the output.



Pulsed operation of Logic gates:

The inputs to a gate are not stationary levels, but are voltages that change frequently between two logic levels & can be classified as pulse waveform.

EX:AND



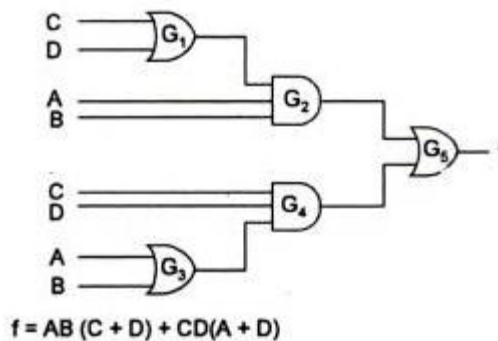
Hybrid Logic:

Both SOP & POS reductions result in a logic circuit in which each input signal has to pass through two gates to reach the output called Two-level logic. It has the advantage of providing uniform time delay between input signals & the output. The disadvantage is that the minimal or POS reductions may not be the actual minimal.

Actual minimal obtained by manipulating the minimal SOP & POS forms into a hybrid form.

EX: $ABC+ABD+ACD+BCD$ ----(SOP) has 16 inputs

$AB(C+D)+CD(A+B)$ ----has 12 inputs.



The C input to the OR gate must go through 3 levels of logic before reaching the output where as C input to the AND gate must only go through two levels, can result critical timing problem called *Logic Race*.

Implementation of Logic functions:

Two level implementation:

The implementation of a logic expression such that each one of the inputs has to pass through only two gates to reach the output is called Two-level implementation.

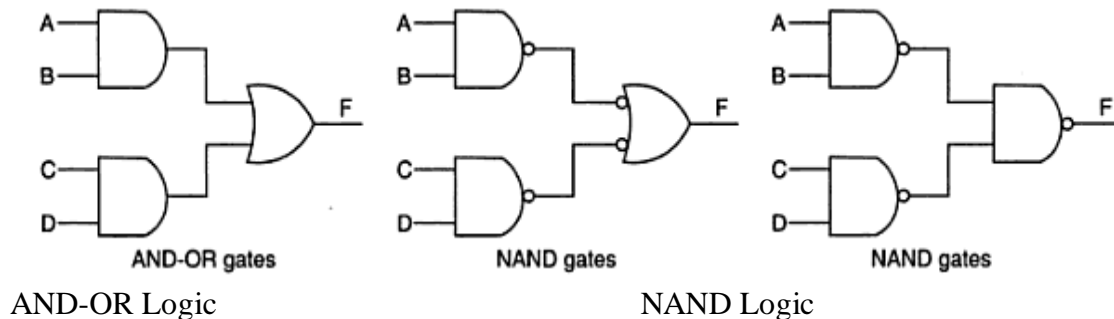
- Both SOP , POS forms result in two-level logic
- Two level implementation can be with AND, OR gates or only NAND or with only NOR gates
- Boolean expression with only NAND gates requires that the function be in SOP form.

$$\text{Function } F = AB + CD$$

(A) AND-OR logic

(B) NAND-NAND logic

$$F = AB + CD = \overline{\overline{AB} + \overline{CD}} = \overline{\overline{AB} \cdot \overline{CD}}$$

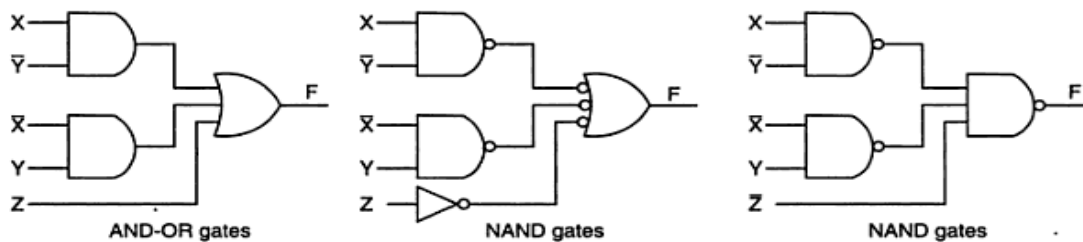


Two –level implementation using AND-OR and NAND logic

The implementation of the form:

$F = XY' + X'Y + Z$ using AND-OR logic and NAND- NAND logic is

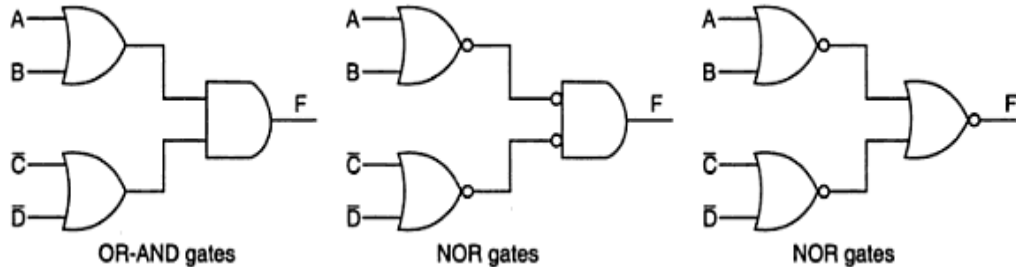
$$F = X\bar{Y} + \bar{X}Y + Z = \overline{\overline{X\bar{Y} + \bar{X}Y + Z}} = \overline{\overline{X\bar{Y}} \cdot \overline{\bar{X}Y} \cdot \bar{Z}}$$



Two –level implementation using AND-OR and NAND logic

The implementation of Boolean expressions with only NOR gates requires that the function be in the form of POS form.

Implementation of the function $(A+B)(C'+D')$

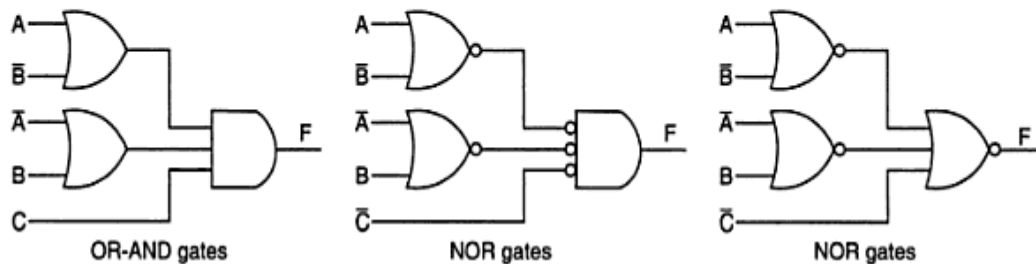


Two –level implementation using OR-AND and NOR logic

The implementation of the function

$$F = (A + \bar{B})(\bar{A} + B)C$$

with (a) OR-AND logic and (b) NOR logic is shown in Figure 6.68.



Two –level implementation using OR-AND and NOR logic

Other two level implementations:

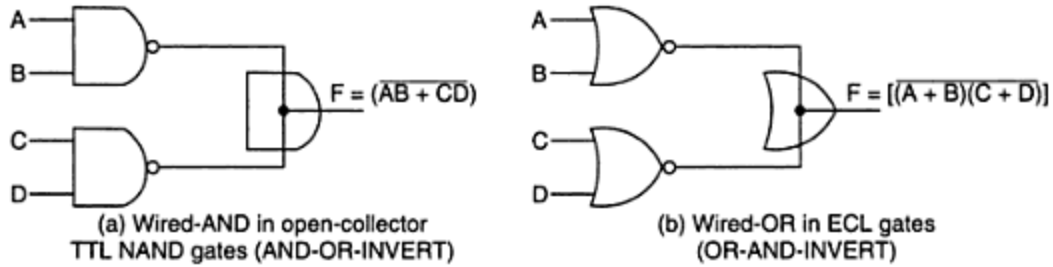
The types of gates most often found in IC's are NAND and NOR

Some NAND or NOR gates allow the possibility of wire connection between the outputs of two gates to provide a specific logic function called Wired Logic.

The logic function implemented by the circuit

$$F = (\overline{AB}) \cdot (\overline{CD}) = \overline{(AB + CD)}$$

Is called an AND-OR Invert function.



Similarly NOR outputs of ECL gates can be tied together to form Wired NOR function.

The logic function implemented by this circuit is

$$F = \overline{(A + B)} + \overline{(C + D)} = \overline{[(A + B)(C + D)]}$$

Is called OR-AND INVERT Function.

EX: Open Collector TTL NAND gates, when tied together perform the wired AND logic is called AOI

$$\begin{aligned} &= \overline{(AB)} \cdot \overline{(CD)} \\ &= \overline{AB + CD} \end{aligned}$$

Similarly NOR outputs of ECL can tied together to perform a wired NOR function.

$$\begin{aligned} F &= \overline{(A + B)} + \overline{(C + D)} \\ &= \overline{[(A + B)(C + D)]} \end{aligned}$$

Non Degenerate forms:

Considering 4 types of gates AND, OR, NAND, NOR & assign one type of gate for the first level & one type of gate for the second level. Find 16 possible combinations of two level form. Eight of these are degenerate forms. Because they generate to a single operation. i.e, AND gate in first level & AND gate in second The output is nearly the AND function of all input variables.

The other non degenerate forms produce an implementation in SOP or POS are

| | |
|-----------|-----------|
| AND-OR | OR-AND |
| NAND-NAND | NOR-NOR |
| NOR-OR | NAND-NAND |
| OR-NAND | AND-NOR |

The two forms are dual of each other.

AND-OR & OR-AND forms are the basic two-level forms.

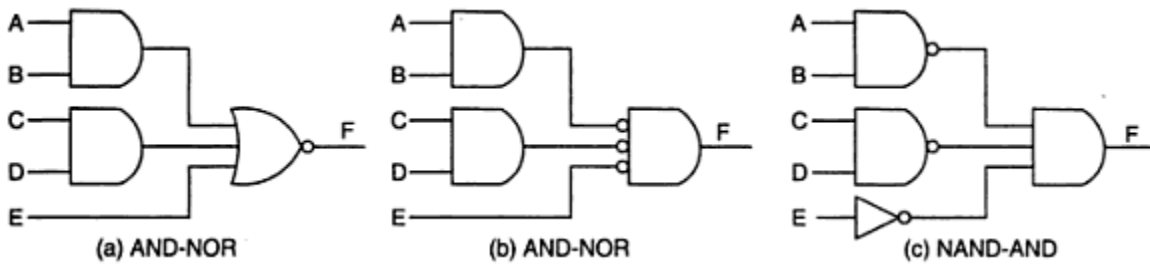
NAND-NAND, NOR_NOR

AOI Implementation:

The two forms Nandi-And and And-Nor perform AOI function.

Inversion isand-Nor form resembles the and-Or form done by the bubble in the output of the NOR gate.

Its function is $F = \overline{AB + CD + E}$



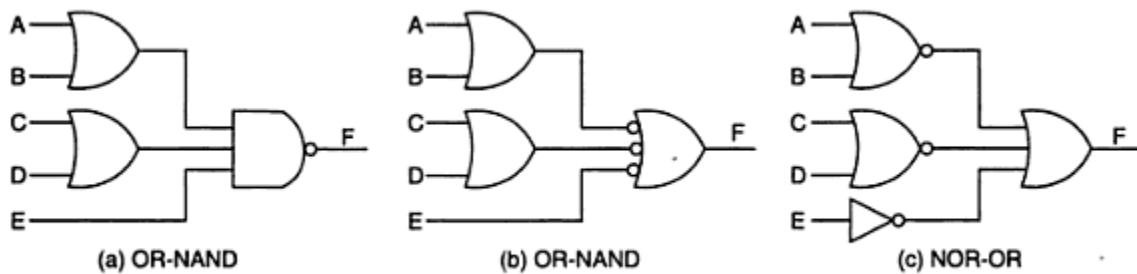
Two-level implementation in AND-NOR and NAND-AND form

OAI Implementation:

The twoforms OR-NAND and NOR-NOR perform OAI function.

OR-NAND form OR-AND form except inversion done by bubble in NAND gate.

Function $F = \overline{(\overline{A + B})(\overline{C + D})E}$



Two-level implementation in OR-NAND form and NOR-OR form.

Summary:

| Equivalent nondegenerate form | | Implements the function | Simplify \bar{F} in | To get an output of |
|--------------------------------------|-------------|--------------------------------|-------------------------------------------------------------------|----------------------------|
| (a) | (b)* | | | |
| AND-NOR | NAND-AND | AND-OR-INVERT | Sum of products by combining 0s in the map | F |
| OR-NAND | NOR-OR | OR-AND-INVERT | Product of sums by combining 1s in the map and then complementing | F |