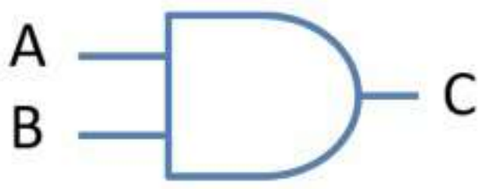
AND Gate

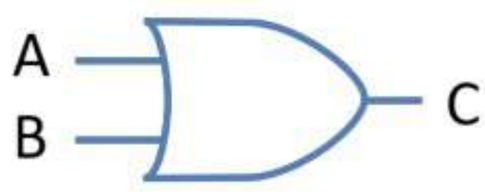
- An AND gate has two or more inputs but only one output.
- The output assumes the logic 1, only when each one of its inputs is at logic 1.
- The output assumes the logic 0 even if one of its inputs is at logic 0.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = A \cdot B$



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

OR Gate

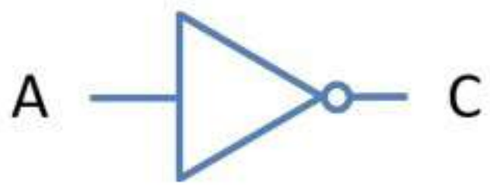
- An OR gate has two or more inputs but only one output.
- The output assumes the logic 0, only when each one of its inputs is at logic 0.
- The output assumes the logic 1 even if one of its inputs is at logic 1.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = A + B$



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

NOT Gate

- A NOT gate (also called an *inverter*) has only one input & one output.
- It is a device whose output is always the complement of its input.
- The output assumes the logic 1, when its input is at logic 0.
- The output assumes the logic 0, when its input is at logic 1.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = \bar{A}$

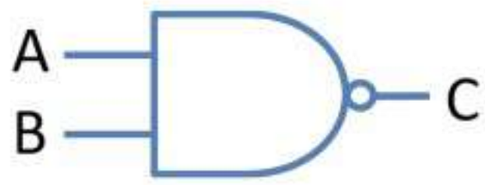


A	C
0	1
1	0

NAND Gate (Universal Gate)

- NAND means NOT AND, i.e. the AND output is NOTed.
- The output assumes the logic 0, only when each one of its inputs is at logic 1.

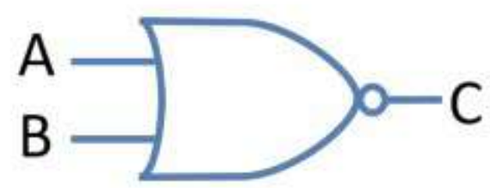
- The output assumes the logic 1 even if one of its inputs is at logic 0.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = \overline{A \cdot B}$



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

NOR Gate (Universal Gate)

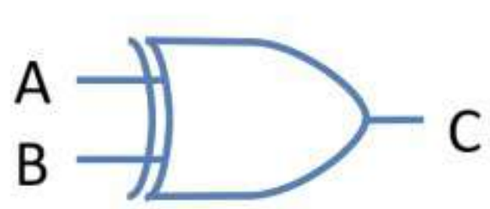
- NOR means NOT OR, i.e. the OR output is NOTed.
- The output assumes the logic 1, only when each one of its inputs is at logic 0.
- The output assumes the logic 0 even if one of its inputs is at logic 1.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = \overline{A + B}$



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

EX-OR Gate

- An X-OR gate has two or more inputs but only one output.
- The output assumes the logic 1 when one and only one of its inputs assumes a logic 1.
- Under the conditions when both the inputs assume the logic 0, or when both the inputs assume the logic 1, the output assumes a logic 0.
- Since, an X-OR gate produces an output 1 only when the inputs are not equal, it is called an *anti-coincidence gate* or *inequality detector*.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = A \oplus B$

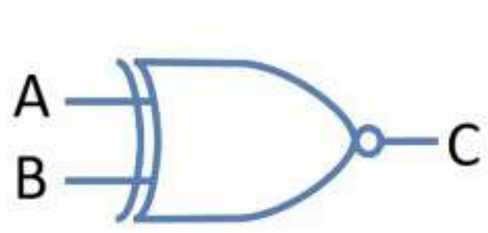


A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

EX-NOR Gate

- An X-NOR gate has two or more inputs but only one output.
- The output assumes the logic 0 when one and only one of its inputs assumes a logic 0.
- Under the conditions when both the inputs assume the logic 1, or when both the inputs assume the logic 1, the output assumes a logic 0.

- Since, an X-NOR gate produces an output 1 only when the inputs are equal, it is called a *coincidence gate* or *equality detector*.
- The logic symbol & truth table are shown in below figure.
- Notation:- $C = A \odot B$

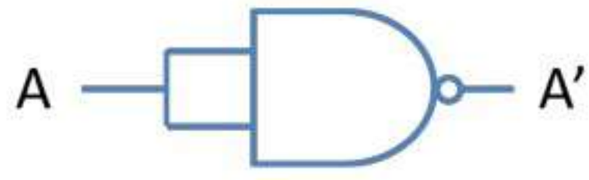


A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

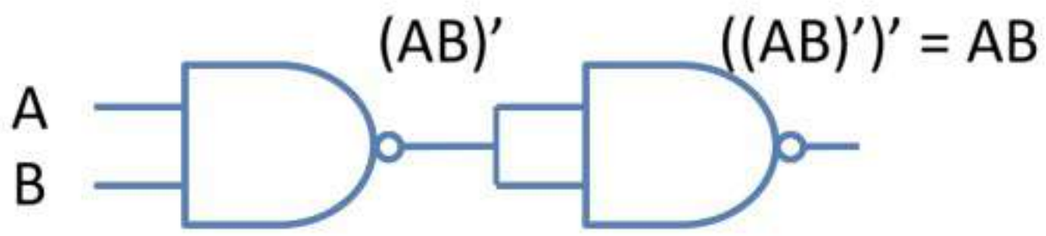
Basic Gates as Universal Gates

- ***Implementation of NOT, AND & OR gates using NAND gate only***

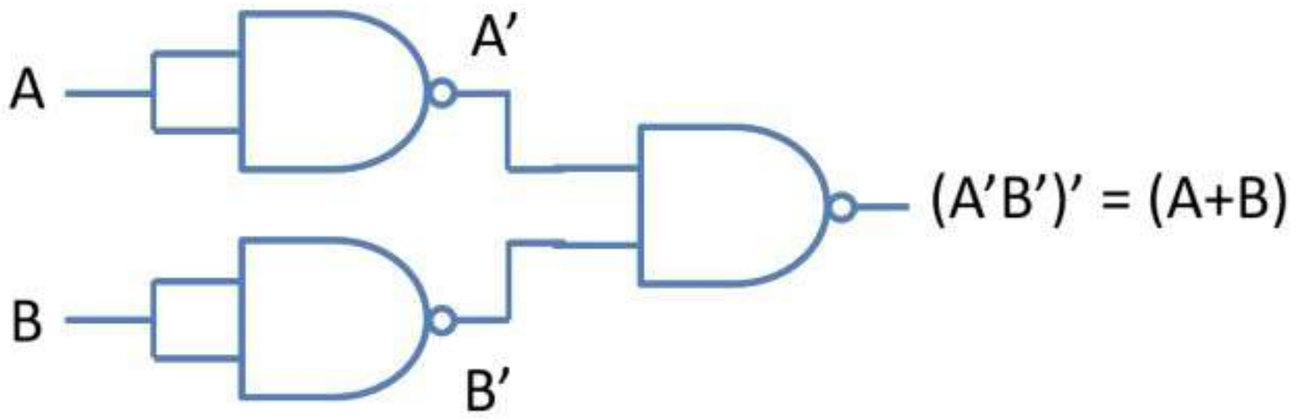
1. NOT using NAND gate
   - A NAND gate can also be used as an inverter by tying all its input terminals together and applying the signal to be inverted to the common terminal.



2. AND using NAND gate
   - NAND means NOT AND, i.e. the AND output is NOTed.
   - So, a NAND gate is combination of an AND gate and a NOT gate.

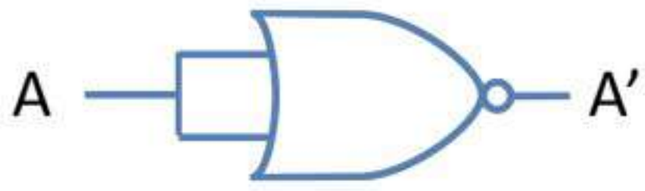


3. OR using NAND gate
   - By inverting inputs in NAND gate, a OR gate is constructed via De Morgan's theorem.
   - $\overline{\overline{A}\,\overline{B}} = \bar{\bar{A}} + \bar{\bar{B}} = A + B$

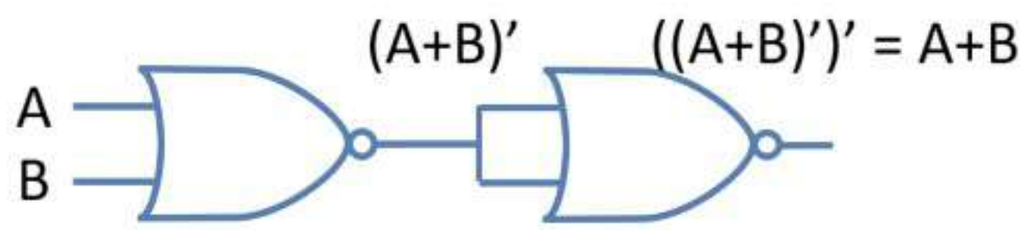


- ***Implementation of NOT, AND & OR gates using NOR gate only***

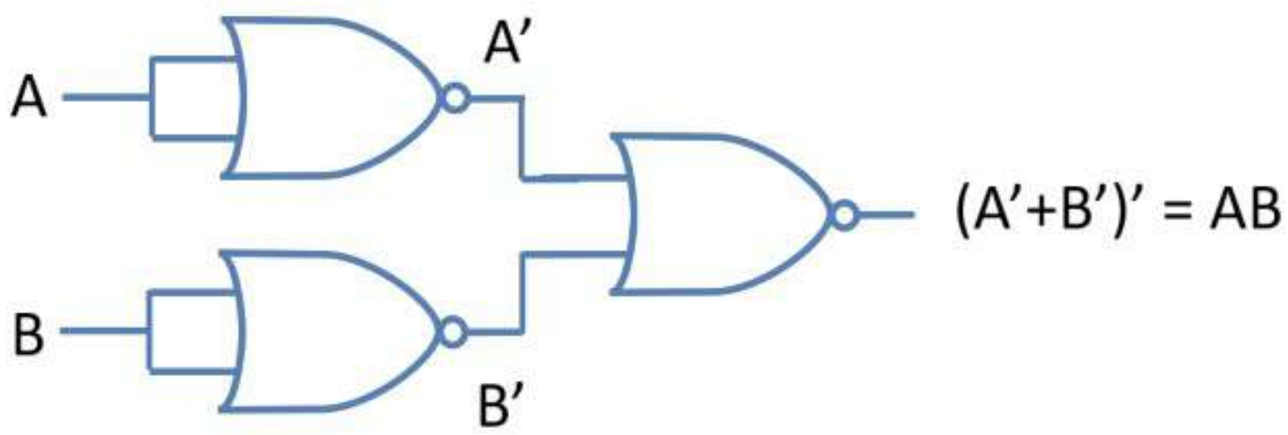
1. NOT using NOR gate
   - A NOR gate can also be used as an inverter by tying all its input terminals together and applying the signal to be inverted to the common terminal.



2. OR using NOR gate
   - NOR means NOT OR, i.e. the OR output is NOTed.
   - So, a NOR gate is combination of an OR gate and a NOT gate.



3. AND using NOR gate
   - By inverting inputs in NOR gate, a AND gate is constructed via De Morgan's theorem.
   - $\overline{\bar{A} + \bar{B}} = \bar{\bar{A}}\,\bar{\bar{B}} = A\,B$



Boolean Algebra Laws

- AND laws
  1. $A \cdot 0 = 0\ (Null\ Law)$
  2. $A \cdot 1 = A\ (Identity\ Law)$
  3. $A \cdot A = A$
  4. $A \cdot \bar{A} = 0$
- OR laws
  1. $A + 0 = A\ (Null\ Law)$
  2. $A + 1 = 1\ (Identity\ Law)$
  3. $A + A = A$
  4. $A + \bar{A} = 1$

- Commutative laws
  1. $A + B = B + A$
  2. $A \cdot B = B \cdot A$

- Associative laws
  1. $(A + B) + C = A + (B + C)$
  2. $(A \cdot B)C = A(B \cdot C)$

- Distributive laws
  1. $A(B + C) = AB + AC$
  2. $A + BC = (A + B)(A + C)$

- Redundant Literal Rule
  1. $A + \bar{A}B = A + B$
  2. $A(\bar{A} + B) = AB$

- Idempotence laws
  1. $A \cdot A = A$
  2. $A + A = A$

- Absorption laws
  1. $A + AB = A$
  2. $A(A + B) = A$

De Morgan's Theorem

1. Law 1 : $\overline{A + B + C} = \bar{A}\,\bar{B}\,\bar{C}$
   - This law states that the complement of a sum of variables is equal to the product of their individual complements.

A	B	C	A+B+C	(A+B+C)′	A′	B′	C′	A′B′C′
0	0	0	0	**1**	1	1	1	**1**
0	0	1	1	**0**	1	1	0	**0**
0	1	0	1	**0**	1	0	1	**0**
0	1	1	1	**0**	1	0	0	**0**
1	0	0	1	**0**	0	1	1	**0**
1	0	1	1	**0**	0	1	0	**0**
1	1	0	1	**0**	0	0	1	**0**
1	1	1	1	**0**	0	0	0	**0**

   - Hence, Law 1 is proved from the above truth table.

2. Law 2 : $\overline{A\,B\,C} = \bar{A} + \bar{B} + \bar{C}$
   - This law states that the complement of a product of variables is equal to the sum of their individual complements.

A	B	C	A B C	(A B C)′	A′	B′	C′	A′+B′+C′
0	0	0	0	**1**	1	1	1	**1**
0	0	1	0	**1**	1	1	0	**1**
0	1	0	0	**1**	1	0	1	**1**
0	1	1	0	**1**	1	0	0	**1**
1	0	0	0	**1**	0	1	1	**1**
1	0	1	0	**1**	0	1	0	**1**
1	1	0	0	**1**	0	0	1	**1**
1	1	1	1	**0**	0	0	0	**0**

Hence, Law 2 is proved from the above truth table.

Reduction of Boolean Expression

1. $f = A[B + \bar{C}\,(\overline{AB + A\bar{C}})]$

$= A\left[B + \bar{C}\left(\overline{AB}\;\overline{A\bar{C}}\right)\right]$ (De Morgan's Theorem)

$= A[B + \bar{C}\,(\bar{A} + \bar{B})(\bar{A} + C)]$ (De Morgan's Theorem)

$= A[B + \bar{C}\,(\bar{A}\bar{A} + \bar{A}C + \bar{B}\bar{A} + \bar{B}C)]$ (Distributive Law)

$= A[B + \bar{C}\,(\bar{A} + \bar{A}C + \bar{A}\bar{B} + \bar{B}C)]$ (A' · A' = A')

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

$= A(B + \bar{A}\bar{C} + 0 + \bar{A}\bar{B}\bar{C} + 0)$ (C · C' = 0)

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

$= AB$ (A · A' = 0)

2. $f = A + B[AC + (B + \bar{C})D]$

$= A + B[AC + BD + \bar{C}D]$ (Distributive Law)

$= A + BAC + BBD + B\bar{C}D$ (Distributive Law)

$= A + ABC + BD + B\bar{C}D$ (B · B = B)

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

$= A \cdot 1 + BD \cdot 1$ (1 + A = A)

$= A + BD$

Common Number Systems

- There are mainly four number systems which are used in digital electronics platform.

1. Decimal number system

- The decimal number system contains ten unique symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- The base or radix is 10.
- 9's and 10's complements are possible for any decimal number.

2. Binary number system

- The binary number system contains two unique symbols 0, 1.
- The base or radix is 2.
- 1's and 2's complements are possible for any binary number.

3. Octal number system

- The octal number system contains eight unique symbols 0, 1, 2, 3, 4, 5, 6, 7.
- The base or radix is 8.
- 7's and 8's complements are possible for any octal number.

4. Hexadecimal number system

- The hexadecimal number system contains sixteen unique symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
- The base or radix is 16.
- 15's and 16's complements are possible for any hexadecimal number.

- In general, if radix or base of a number system is "r", then there is possibility of r's complement and (r-1)'s complement of a number.

Decimal to Binary Conversion

- The decimal integer is converted to the binary integer number by successive division by 2, and the decimal fraction is converted to the binary fraction number by successive multiplication by 2.
- In the successive division-by-2 method, the given decimal integer number is successively divided by 2 till the quotient is 0.
- The remainders read from bottom to top give the equivalent binary integer number.
- In the successive multiplication-by-2 method, the given decimal fraction and the subsequent fractions are successively multiplied by 2, till the fraction part of the product is 0 or till the desired accuracy is obtained.
- The integers read from top to bottom give the equivalent binary integer number.
- To convert a mixed number to binary, convert the integer and fraction parts separately to binary and then combine them.

- Example:- $(125.6875)_{10} = (\ )_2$

2	125	1
2	62	0
2	31	1
2	15	1
2	7	1
2	3	1
2	1	1
	0	

0.6875 x 2 = 1.3750 | 1 + 0.3750
0.3750 x 2 = 0.7500 | 0 + 0.7500
0.7500 x 2 = 1.5000 | 1 + 0.5000
0.5000 x 2 = 1.0000 | 1 + 0.0000

Hence, $(125.6875)_{10} = (1111101.1011)_2$

Binary to Decimal Conversion

- Binary numbers may be converted to their decimal equivalents by the positional weights method.
- In this method, each binary digit of the number is multiplied by its position weight (2^n, where n is the weight of the bit) and the product terms are added to obtain the decimal number.

- Example:- $(101011.11)_2 = (\ )_{10}$

$= 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 + 1x2^{-1} + 1x2^{-2}$
$= 32 + 0 + 8 + 0 + 2 + 1 + 0.5 + 0.25$
$= 43.75$

Hence, $(101011.11)_2 = (43.75)_{10}$

Decimal to Octal Conversion

- The decimal integer is converted to the octal integer number by successive division by 8, and the decimal fraction is converted to the octal fraction number by successive multiplication by 8.
- In the successive division-by-8 method, the given decimal integer number is successively divided by 8 till the quotient is 0.
- The remainders read from bottom to top give the equivalent octal integer number.
- In the successive multiplication-by-8 method, the given decimal fraction and the subsequent fractions are successively multiplied by 8, till the fraction part of the product is 0 or till the desired accuracy is obtained.
- The integers read from top to bottom give the equivalent octal integer number.
- To convert a mixed number to octal, convert the integer and fraction parts separately to octal and then combine them.

- Example:- $(125.6875)_{10} = (\ )_8$

8	125	5
8	15	7
8	1	1
	0	

0.6875 x 8 = 5.5000 | 5 + 0.5000
0.5000 x 8 = 4.0000 ↓ 4 + 0.0000

Hence, $(125.6875)_{10} = (175.54)_8$

Octal to Decimal Conversion

- Octal numbers may be converted to their decimal equivalents by the positional weights method.
- In this method, each octal digit of the number is multiplied by its position weight (8^n, where n is the weight of the bit) and the product terms are added to obtain the decimal number.

- Example:- $(724.25)_8 = (\ )_{10}$

$= 7x8^2 + 2x8^1 + 4x8^0 + 2x8^{-1} + 5x8^{-2}$
$= 448 + 16 + 4 + 0.25 + 0.0781$
$= 468.3281$

Hence, $(724.25)_8 = (468.3281)_{10}$

Decimal to Hexadecimal Conversion

- The decimal integer is converted to the hexadecimal integer number by successive division by 16, and the decimal fraction is converted to the hexadecimal fraction number by successive multiplication by 16.
- In the successive division-by-16 method, the given decimal integer number is successively divided by 16 till the quotient is 0.
- The remainders read from bottom to top give the equivalent hexadecimal integer number.

- In the successive multiplication-by-16 method, the given decimal fraction and the subsequent fractions are successively multiplied by 16, till the fraction part of the product is 0 or till the desired accuracy is obtained.
- The integers read from top to bottom give the equivalent hexadecimal integer number.
- To convert a mixed number to hexadecimal, convert the integer and fraction parts separately to hexadecimal and then combine them.

- Example:- $(2598.675)_{10} = (\ )_{16}$

16	2598	6 ↑
16	162	2
16	10	10(A)
	0	

0.6750 x 16 = 10.8000 | 10(A) + 0.8000 ↓
0.8000 x 16 = 12.8000 | 12(C) + 0.8000
0.8000 x 16 = 12.8000 | 12(C) + 0.8000
0.8000 x 16 = 12.8000 | 12(C) + 0.8000

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

Hexadecimal to Decimal Conversion

- Hexadecimal numbers may be converted to their decimal equivalents by the positional weights method.
- In this method, each hexadecimal digit of the number is multiplied by its position weight (16^n, where n is the weight of the bit) and the product terms are added to obtain the decimal number.
- Example:- $(A0F9.0EB)_{16} = (\ )_{10}$

$= 10x16^3 + 0x16^2 + 15x16^1 + 9x16^0 + 0x16^{-1} + 14x16^{-2} + 11x16^{-3}$
$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$
$= 41209.0572$

Hence, $(A0F9.0EB)_{16} = (41209.0572)_{10}$

Octal to Binary Conversion

- To convert a given octal number to a binary, just replace each octal digit by its 3-bit binary equivalent as per below table.

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

- Example:- $(367.52)_8 = (\ )_2$

  = 011 110 111 . 101 010
  = 11110111.10101

  Hence, $(367.52)_8 = (11110111.10101)_2$

Binary to Octal Conversion

- To convert a binary number to an octal number, starting from the binary point make groups of 3 bits each (i.e. from point (".") in binary number, group of 3 bits in left side and group of 3 bits in right side), if there are not 3 bits available at last, just stuff "0" to make 3 bits group.
- Replace each 3-bit binary group by the equivalent octal digit.

- Example:- $(110101.101010)_2 = (\ )_8$

  = 110 101 . 101 010
  = 65.52

  Hence, $(110101.101010)_2 = (65.52)_8$

Hexadecimal to Binary Conversion

- To convert a given hexadecimal number to a binary, just replace each hexadecimal digit by its 4-bit binary equivalent as per below table.

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

- Example:- $(3A9E.B0D)_{16} = (\ )_2$

  = 0011 1010 1001 1110 . 1011 0000 1101
  = 11101010011110.101100001101

  Hence, $(3A9E.B0D)_{16} = (11101010011110.101100001101)_2$

Binary to Hexadecimal Conversion

- To convert a binary number to a hexadecimal number, starting from the binary point make groups of 4 bits each (i.e. from point (".") in binary number, group of 4 bits in left side and group of 4 bits in right side), if there are not 4 bits available at last, just stuff "0" to make 4 bits group.
- Replace each 4-bit binary group by the equivalent hexadecimal digit.

- Example:- $(01011111011.011111)_2 = (\ )_{16}$

  = 0010 1111 1011 . 0111 1100
  = 2FB.7C

  Hence, $(01011111011.011111)_2 = (2FB.7C)_{16}$

Octal to Hexadecimal Conversion

- To convert an octal number to hexadecimal, the simplest way is to first convert the given octal number to binary and then the binary number to hexadecimal. (i.e. first from table of 3-bit and then table of 4-bit)

- Example :- $(756.603)_8 = (\ )_{16}$

  = 7 5 6 . 6 0 3
  111 101 110 . 110 000 011

  = 0001 1110 1110 . 1100 0001 1000
  1 E E . C 1 8

  Hence, $(756.603)_8 = (1EE.C18)_{16}$

Hexadecimal to Octal Conversion

- To convert a hexadecimal number to octal, the simplest way is to first convert the given hexadecimal number to binary and then the binary number to octal. (i.e. first from table of 4-bit and then table of 3-bit)

- Example :- $(B9F.AE)_{16} = (\ )_8$

  = B 9 F . A E
  1011 1001 1111 . 1010 1110

= 101 110 011 111 . 101 011 100
  5 6 3 7 . 5 3 4

Hence, $(B9F.AE)_{16} = (5637.534)_8$

Accuracy in Binary Number Conversion

- Example:- Convert $(0.252)_{10}$ to binary with an error less than 1%
- Absolute value of allowable error is found by calculating 1% of the number.

$$E_{allow} = 0.01\ x\ 0.252\ =\ 0.00252_{10}$$

- Maximum error due to truncation is set to be less than allowable error by solving from $E_{10} = 2^{-n}$

This equation is written as $2^{-n} < 0.00252$

- Inverting both sides of the inequality

$$2^n > 397$$

- Taking log of both sides and solving for n

$$n \log 2 = \log 397$$

$$n = \frac{\log 397}{\log 2} = 8.63\ \approx\ 9 \text{ (next largest integer)}$$

- This indicates that the use of 9 bits in the binary number will guarantee an error less than 1%.
- So, the conversion is carried out to 9 places.

0.252 x 2 = 0.504	0
0.504 x 2 = 1.008	1
0.008 x 2 = 0.016	0
0.016 x 2 = 0.032	0
0.032 x 2 = 0.064	0
0.064 x 2 = 0.128	0
0.128 x 2 = 0.256	0
0.256 x 2 = 0.512	0
0.512 x 2 = 1.024	1

- Therefore, $(0.252)_{10} = (0.010000001)_2$

Complement Forms

- 9's Complement
  - The 9's complement of a decimal number is obtained by subtracting each digit of that decimal number from 9.
  - Example:- 782.54

```
  9 9 9 . 9 9
- 7 8 2 . 5 4
  2 1 7 . 4 5 (9's complement of 782.54)
```

- 10's Complement
  - The 10's complement of a decimal number is obtained by adding a 1 to its 9's complement.
  - *Shortcut:- Subtract LSB(Least Significant Bit) from 10 and rest of the digits from 9.*
  - Example:- 1056.074

```
  9 9 9 9 . 9 9 9
- 1 0 5 6 . 0 7 4
  8 9 4 3 . 9 2 5
          + 1
  8 9 4 3 . 9 2 6 (10's complement of 1056.074)
```

- 1's Complement
  - The 1's complement of a binary number is obtained by subtracting each digit of that binary number from 1.
  - *Shortcut: Change 1's to 0's and 0's to 1's in binary number.*
  - Example:- 101101.1001

```
  1 1 1 1 1 1 . 1 1 1 1
- 1 0 1 1 0 1 . 1 0 0 1
  0 1 0 0 1 0 . 0 1 1 0 (1's complement of 101101.1001)
```

- 2's Complement
  - The 2's complement of a binary number is obtained by adding a 1 to its 1's complement.
  - *Shortcut: A shortcut to manually convert a binary number into its 2's complement is to start at the least significant bit (LSB), and copy all the zeros, working from LSB toward the most significant bit (MSB) until the first 1 is reached; then copy that 1, and flip all the remaining bits.*
  - Example:- 110101.10100

```
  1 1 1 1 1 1 . 1 1 1 1 1
- 1 1 0 1 0 1 . 1 0 1 0 0
  0 0 1 0 1 0 . 0 1 0 1 1
              +         1
  0 0 1 0 1 0 . 0 1 1 0 0 (2's complement of 110101.10100)
```

Signed Number Representation

- The 2's complement system for representing signed numbers works like this:
  1) If the number is positive, the magnitude is represented in its true binary form and a sign bit 0 is placed in front of the MSB.
  2) If the number is negative, the magnitude is represented in its 2's complement form and a sign bit 1 is placed in front of the MSB.

- Example:-
  (1) Express -45 in 8-bit 2's complement form.
  Ans. +45 in 8-bit form is 00101101 (By taking decimal to binary conversion).
  take 2's complement of it, 11010011
  Hence, -45 in 2's complement form is 11010011

  (2) Express -73.25 in 12-bit 2's complement form.
  Ans. +73.25 in 12-bit form is 01001001.1100 (By taking decimal to binary conversion).
  take 2's complement of it, 10110110.0100
  Hence, -73.25 in 2's complement form is 10110110.0100

Subtraction using Complement Forms

- Subtraction using 9's complement form
  - To perform decimal subtraction using the 9's complement method, obtain 9's complement of the subtrahend and add it to the minuend.
  - Call this number the intermediate result.
  - If there is a carry, it indicates that the answer is positive.
  - Add the carry to the LSD of this result to get the answer.
  - If there is no carry, it indicates that the answer is negative and the intermediate result is its 9's complement.
  - Take the 9's complement of this result and place a negative sign in front to get the answer.

  - Example:-
    (1) 745.81 – 436.62

```
Ans. 7 4 5 . 8 1          7 4 5 . 8 1
   - 4 3 6 . 6 2  ------> + 5 6 3 . 3 7  (9's complement of 436.62)
     3 0 9 . 1 9          1 3 0 9 . 1 8  (Intermediate result)
                                + 1
                          -------------
                          3 0 9 . 1 9  (Answer)
```

    (2) 436.62 – 745.81

```
Ans. 4 3 6 . 6 2          4 3 6 . 6 2
   - 7 4 5 . 8 1  ------> + 2 5 4 . 1 8  (9' complement of 745.81)
   - 3 0 9 . 1 9          6 9 0 . 8 0  (Intermediate result)
```

There is no carry indicating that the answer is negative. So, take the 9's complement of the intermediate result and put a minus sign. Therefore, the answer is -309.19

- Subtraction using 10's complement form
  - To perform decimal subtraction using the 10's complement method, obtain the 10's complement of the subtrahend and 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 obtained is itself the answer.
  - If there is no carry, it indicates that the answer is negative and the result obtained is its 10's complement.
  - Obtain the 10's complement of the result and place a negative sign in front to get the answer.

  - Example:-

(1) 2928.54 – 416.73

```
Ans. 2 9 2 8 . 5 4          2 9 2 8 . 5 4
   - 0 4 1 6 . 7 3  ------> + 9 5 8 3 . 2 7 (10's complement of 416.73)
     2 5 1 1 . 8 1        (1) 2 5 1 1 . 8 1 (Ignore the carry)
```

(2) 416.73 – 2928.54

```
Ans. 0 4 1 6 . 7 3          0 4 1 6 . 7 3
   - 2 9 2 8 . 5 4  ------> +7 0 7 1 . 4 6 (10's complement of 2928.54)
   - 2 5 1 1 . 8 1          7 4 8 8 . 1 9 (No carry)
```

There is no carry indicating that the answer is negative. So, take the 10's complement of the intermediate result and put a minus sign. Therefore, the answer is -2511.81

- Subtraction using 1's complement form
  - To perform binary subtraction using the 1's complement method, obtain 1's complement of the subtrahend and add it to the minuend.
  - Call this number the intermediate result.
  - If there is a carry, it indicates that the answer is positive.
  - Add the carry to the LSB of this result to get the answer.
  - If there is no carry, it indicates that the answer is negative and the intermediate result is its 1's complement.
  - Take the 1's complement of this result.

  - Example:-

(1) 11010 – 1101

```
Ans. 1 1 0 1 0              1 1 0 1 0
   - 0 1 1 0 1  ------> +1 0 0 1 0 (1's complement)
                      (1)0 1 1 0 0 (Intermediate result)
                         ------> + 1
                        0 1 1 0 1 (Answer)
```

(2) 100 – 110000

```
Ans. 0 0 0 1 0 0          0 0 0 0 1 0 0
   - 1 1 0 0 0 0  ------> +1 0 0 1 1 1 1 (1's complement)
                          1 0 1 0 0 1 1 (Intermediate result)
```

There is no carry. The MSB is 1. Hence, the answer is negative. Take the 1's complement of the remaining bits. So, it is 101100 (negative).

- Subtraction using 2's complement form
  - To perform binary subtraction using the 2's complement method, obtain the 2's complement of the subtrahend and 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 obtained is itself the answer.
  - If there is no carry, it indicates that the answer is negative and the result obtained is its 2's complement.
  - Obtain the 2's complement of the result.

  - Example:-

(1) 11011 – 1101

Ans. 1 1 0 1 1 → 1 1 0 1 1

- 0 1 1 0 1 → + 1 0 0 1 1 (2's complement)

(1) 0 1 1 0 1 (Ignore the carry, result is positive)

(2) 100 – 110000

Ans. 0 0 0 1 0 0 → 0 0 0 0 1 0 0

- 1 1 0 0 0 0 → +1 0 1 0 0 0 0 (2's complement)

1 0 1 0 1 0 0 (Result is negative)

There is no carry. The MSB is 1. Hence, the answer is negative. Take its 2's complement and put a minus sign. So it is, 101100 (negative).

Binary Operation

- Binary Addition

  0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 10 (i.e. 0 with a carry of 1); 1 + 1 + 1 = 11

  Example:- Add the binary numbers 1101.101 and 111.011.

```
  1 1 1 1   1 1 ← Carry
  1 1 0 1 . 1 0 1
+ 0 1 1 1 . 0 1 1
-----------------
1 0 1 0 1 . 0 0 0
```

- Binary Subtraction

  0 – 0 = 0; 1 – 1 = 0; 1 – 0 = 1; 0 – 1 = 1 with a borrow of 1.

  Example:- Subtract 111.111 from 1010.01

```
  0 1 10 1   1 10 10 ← Borrow
  1 0 1 0 . 0 1 0
- 0 1 1 1 . 1 1 1
-----------------
  0 0 1 0 . 0 1 1
```

- Binary Multiplication
  Example:- Multiply 1011.101 by 101.01.

```
    1 0 1 1 . 1 0 1
  X   1 0 1 . 0 1 0
  ---------------------
      1 0 1 1 1 0 1
    0 0 0 0 0 0 0 0
  1 0 1 1 1 0 1 0 0 0
  0 0 0 0 0 0 0 0 0 0
---------------------------
1 1 1 1 0 1.0 0 0 0 1
```

  Therefore, 1011.101 x 101.010 = 111101.00001

- Binary Division
  Example:- Divide 101101 by 110.

```
1 1 0 ) 1 0 1 1 0 1 ( 1 1 1 . 1
        - 1 1 0
        -------
          1 0 1 0
          - 1 1 0
          -------
            1 0 0 1
            - 1 1 0
            -------
              1 1 0
              1 1 0
              -----
              0 0 0
```

  Therefore, 101101 / 110 = 111.1

BCD Code

- In this code, each decimal digit, 0 through 9, is coded by 4-bit binary number.
- It is a weighted code and is also sequential. Therefore, it is useful for mathematical operations.
- The main advantage of this code is its ease of conversion to and from decimal.
- It is less efficient than the pure binary, in the sense that it requires more bits.
- For example, the decimal number 14 can be represented as 1110 in pure binary but as 0001 0100 in 8421 BCD code.
- Another disadvantage of the BCD code is that, arithmetic operations are more complex than they are in pure binary.
- There are six illegal combinations 1010, 1011, 1100, 1101, 1110 and 1111 in this code.

BCD Addition

- If there is no carry and 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 illegal code, then 6 (0110) is added to the sum term of that group and the resulting carry is added to the next group.
- Example:-
  (1) 25 + 13

```
Ans.    25        0 0 1 0   0 1 0 1  (25 in BCD)
       +13      + 0 0 0 1   0 0 1 1  (13 in BCD)
        38        0 0 1 1   1 0 0 0  (No carry, no illegal code. So, this is the correct sum)
```

(2) 679.6 + 536.8

Ans. 679.6	⇨	0110 0111 1001 . 0110	(679.6 in BCD)
+536.8		+0101 0011 0110 . 1000	(536.8 in BCD)
1216.4		1011 1010 1111 . 1110	(All are illegal codes)
		+0110 +0110 +0110 .+0110	(Add 0110 to each)
		0001 0010 0001 0110 . 0100	(Corrected sum = 1216.4)

BCD Subtraction

- If there is no borrow from the next higher group then no correction is required.
- If there is a borrow from the next group, then 6_{10} (0110) is subtracted from the difference term of this group.
- Example:-

(1) 38 – 15

Ans. 38	⇨	0011 1000	(38 in BCD)
-15		-0001 0101	(15 in BCD)
23		0010 0011	(No borrow. So, this is the correct difference.)

(2) 206.7 – 147.8

Ans. 206.7	⇨	0010 0000 0110 . 0111	(206.7 in BCD)
-147.8		-0001 0100 0111 . 1000	(147.8 in BCD)
58.9		0000 1011 1110 . 1111	(Borrows are present, subtract 0110)
		-0110 -0110 .-0110	
		0101 1000 . 1001	(Corrected difference = 58.9)

Excess-3 Code

- The Excess-3 code, also called XS-3, is a non-weighted BCD code.
- This code derives its name from the fact that each binary code word is the corresponding 8421 code word plus 0011(3).
- It is sequential code and, therefore, can be used for arithmetic operations.
- It is a self-complementing code.
- The XS-3 code has six invalid states 0000, 0001, 0010, 1101, 1110 and 1111.

XS-3 Addition

- If there is no carry out from the addition of any of the 4-bit groups, subtract 0011 from the sum term of those groups.
- If there is a carry out, add 0011 to the sum term of those groups.
- Example:-

(1) 247.6 + 359.4

Ans. 247.6	⇨	0101 0111 1010 . 1001	(247.6 in XS-3)
+359.4		+0110 1000 1100 . 0111	(359.4 in XS-3)
607.0		1100 0000 0111 . 0000	(Add 0011 to 0000, 0111, 0000
		-0011 +0011 +0011 .+0011	and subtract 0011 from 1100)
		1001 0011 1010 0011	(Corrected sum in XS-3 = 607.0)

XS-3 Subtraction

- If there is no borrow from the next 4-bit group, add 0011 to the difference term of such groups.
- If there is a borrow, subtract 0011 from the difference term.

- Example:-

(1) 267 – 175

Ans. 267 → 0101 1001 1010 (267 in XS-3)
- 175 - 0100 1010 1000 (175 in XS-3)
092 0000 1111 0010 (Subtract 0011 from 1111 and
+0011 -0011 +0011 add 0011 to 0000 & 0010)
0011 1100 0101 (Corrected difference in XS-3 = 92)

Gray Code

- The gray code is a non-weighted 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.
- It is also a reflective code.
- The n least significant bits for 2^n through 2^{n+1}-1 are the mirror images of those 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, and putting the MSB of 0 above the axis and the MSB of 1 below the axis.
- One reason for the popularity of the gray code is its ease of conversion to and from binary.
- Reflection of gray code is shown in table.

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
		100	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

- If an n-bit binary number is represented by $B_n\ B_{n-1}\ \ldots\ B_1$ and its gray code equivalent $G_n\ G_{n-1}\ \ldots\ G_1$, where B_n and G_n are the MSBs, then the gray code bits are obtained from the binary code as follows:

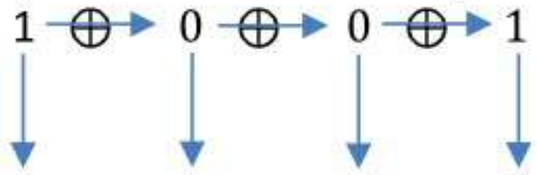
$G_n = B_n$	$G_{n-1} = B_n \oplus B_{n-1}$	$G_{n-2} = B_{n-1} \oplus B_{n-2}$	.....	$G_1 = B_2 \oplus B_1$

- The conversion procedure is as follows:
  1. Record the MSB of the binary as the MSB of the gray code.
  2. Perform X-ORing between the MSB of the binary and the next bit in binary. This answer is the next bit of the gray code.

3. Perform X-ORing between 2nd bit of the binary and 3rd bit of the binary, the 3rd bit with the 4th bit, and so on.
4. Record the successive answer bits as the successive bits of the gray code until all the bits of the binary number are exhausted.

- Example:- Convert the binary 1001 to Gray code.

Ans: 1 ⊕→ 0 ⊕→ 0 ⊕→ 1



1 1 0 1

Gray to Binary Conversion

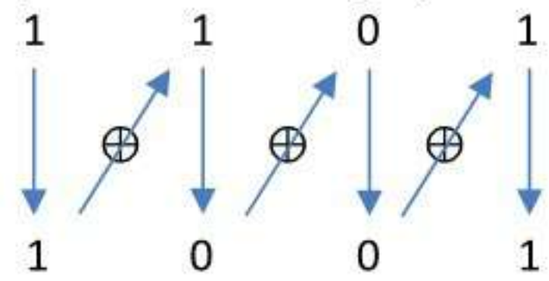
- If an n-bit gray number is represented by $G_n\ G_{n-1} \ldots G_1$ and its binary code equivalent $B_n\ B_{n-1} \ldots B_1$, where G_n and B_n are the MSBs, then the binary bits are obtained from the gray bits as follows:

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

- The conversion procedure is as follows:
  1. The MSB of the binary number is the same as the MSB of the gray code number.
  2. Perform X-ORing between the MSB of the binary and next significant bit of gray code. This answer is the next bit of binary.
  3. Perform X-ORing between the 2nd bit of the binary and 3rd bit of the gray code, the 3rd bit of the binary with the 4th bit of gray code, and so on.
  4. Record the successive answers as the successive bits of the binary until all the bits of the gray code are exhausted.

- Example:- Convert the gray code 1101 to binary.

Ans. 1 1 0 1



1 0 0 1

Error Detecting Code

- When binary data is transmitted and processed, it is susceptible to noise that can alter or distort its contents.
- The 1s may get changed to 0s and 0s to 1s.
- Because digital systems must be accurate to the digit, errors can pose a serious 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 and retransmitted.

Parity

- The simplest technique for detecting errors is that of adding an extra bit, known as the *parity* bit, to each word being transmitted.
- There are two types of parity – odd parity and even parity.
- For odd parity, the parity bit is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an odd number.
- For even parity, the parity bit is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an even number.

- When the digital data is received, a parity checking circuit generates an error signal if the total number of 1s 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 can not detect two or more errors within the same word.
- In any practical system, there is always a finite probability of the occurrence of single error.
- Odd parity is used more often than even parity because even parity does not detect the situation where all 0s are created by a short-circuit or some other fault condition.
- Example:- If Data is 1011010, then even parity must be 0 and odd parity must be 1.

Check Sums

- Simple parity cannot detect two errors within the same word.
- One way of overcoming this difficulty is to use a sort of two-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 is 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.
- This is the type of transmission used in teleprocessing systems.

Block Parity

- When several binary words are transmitted or stored in succession, the resulting collection of bits can be regarded as a block of data, having rows and columns.
- Parity bits can then be assigned to both rows and columns.
- This scheme makes it possible to *correct* any single error occurring in a data word and to *detect* any two errors in a word.
- This technique also called word parity, is widely used for data stored on magnetic tapes.
- For example, six 8-bit words in succession can be formed into a 6x8 block for transmission.
- Parity bits are added so that odd parity is maintained both row-wise and column-wise and the block is transmitted as a 7x9 block as shown in Figure 1.
- At the receiving end, parity is checked both row-wise and column-wise and suppose errors are detected as shown in Figure 2.
- These single-bit errors detected can be corrected by complementing the error bit.
- In Figure 2, parity errors in the 3^{rd} row and 5^{th} column mean that the 5^{th} bit in 3^{rd} row is in error.
- It can be corrected by complementing it.
- Two errors as shown in Figure 3 can only be detected but not corrected.
- In Figure 3, parity errors are observed in both columns 2 and 4.
- It indicated that in one row there two errors.

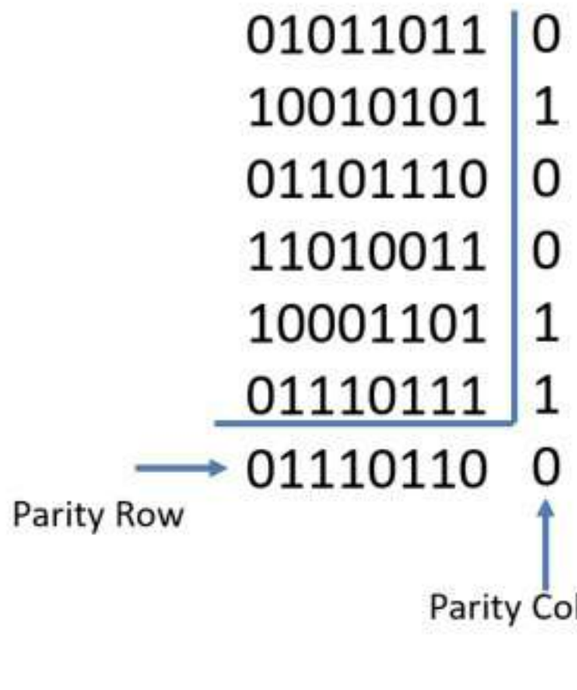


Figure 1

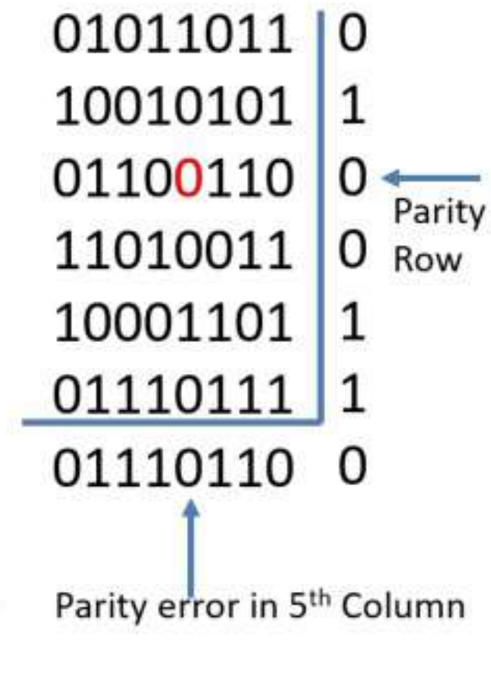


Figure 2

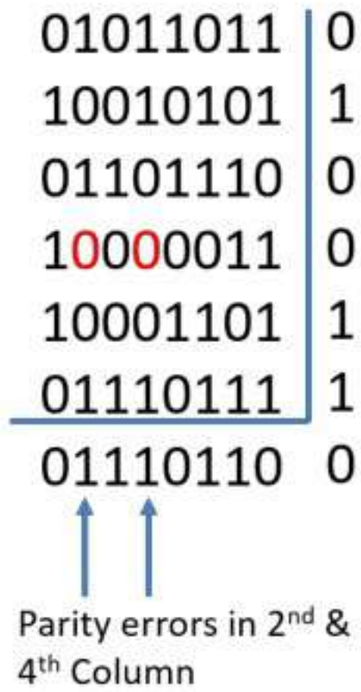


Figure 3

Error Correcting Code

- A code is said to be an error-correcting code, if the correct code word can always be deducted 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 a code is the smallest number of bits by which any two code words must differ.
- A code with minimum distance of three can not only correct single-bit errors, but also detect (but cannot correct) two-bit errors.
- The key to error correction is that it must be possible to detect and locate erroneous digits.
- If the location of an error correction is determined, then by complementing the erroneous digit, the message can be corrected.
- 7-bit hamming code is one type of error-correcting code.

The 7-bit hamming code

- To transmit four data bits, three parity bits located at positions 2^0, 2^1, and 2^2 from left are added to make a 7-bit code word which is then transmitted.
- The word format would be as shown below:

  $$\mathbf{P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7}$$

  Where the D bits are the data bits and the P bits are the parity bits.
- P_1 is to be set a 0 or 1 so that it establishes even parity bits 1, 3, 5, and 7 (i.e. $P_1\ D_3\ D_5\ D_7$).
- P_2 is to be set a 0 or 1 so that it establishes even parity bits 2, 3, 6, and 7 (i.e. $P_2\ D_3\ D_6\ D_7$).
- P_4 is to be set a 0 or 1 so that it establishes even parity bits 4, 5, 6, and 7 (i.e. $P_4\ D_5\ D_6\ D_7$).

- Example:-

  (1) Encode data bits 1101 into the 7-bit even-parity hamming code.

  Ans: The bit pattern is $P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7 = P_1\ P_2\ 1\ P_4\ 1\ 0\ 1$

  Bits 1, 3, 5, 7 (i.e. P_1 1 1 1) must have even parity. So, P_1 must be a 1.
  Bits 2, 3, 6, 7 (i.e. P_2 1 0 1) must have even parity. So, P_2 must be a 0.
  Bits 4, 5, 6, 7 (i.e. P_4 1 0 1) must have even parity. So, P_4 must be a 0.

  Therefore, the final code is 1 0 1 0 1 0 1.

(2) The message 1001001 in the 7-bit hamming code is transmitted through a noisy channel. Decode the message assuming that at most a single error occurred in each code word.

Ans. Message is 1001001

Bits 1, 3, 5, 7 (1001) ⟶ no error ⟶ $C_1 = 0$
Bits 2, 3, 6, 7 (0001) ⟶ error ⟶ $C_2 = 1$
Bits 4, 5, 6, 7 (1001) ⟶ no error ⟶ $C_3 = 0$

The error word is $C_3C_2C_1 = 010 = 2_{10}$. So, complement the 2nd bit from left.
Therefore, the correct code is 1 1 0 1 0 0 1.

Digital IC Specifications

1. ***Threshold voltage:-***
   - It is defined as that voltage at the input of a gate which causes a change in the state of the output from one logic level to the other.

2. ***Propagation delay:-***
   - A pulse through a gate takes a certain amount of time to propagate from input to output. This interval of time is known as the *propagation delay* of gate.
   - It is the average transition delay time t_{pd}, expressed by

$$t_{pd} = \frac{t_{PLH}+t_{PHL}}{2}$$

where, t_{PLH} is the signal delay time when the output goes from a logic 0 to a logic 1 state and t_{PHL} is the signal delay time when the output goes from a logic 1 to logic 0 state.

3. ***Power dissipation:-***
   - The *power dissipation*, P_D, of a logic gate is the power required by the gate to operate with 50% duty cycle at a specified frequency and is expressed in milliwatts.

$$P_D = V_{CC} \times I_{CC}(avg)/n, \text{ where } Icc(avg) = \frac{I_{CCH}+I_{CCL}}{2}$$

4. ***Fan-in:-***
   - The *fan-in* of a logic gate is defined as the number of inputs that the gate is designed to handle.

5. ***Fan-out:-***
   - The *fan-out* (also called the *loading factor*) of a logic gate is defined as the maximum number of standard loads that the output of the gate can drive without impairing its normal operation.

6. ***Voltage parameter:-***
   - $V_{IH}(min)$*:* It is the minimum voltage level required at the input of a gate for that input to be treated as a logic 1. Any voltage below this level will not be accepted as a logic 1 input by the logic circuit.
   - $V_{OH}(min)$*:* It is the minimum voltage level required at the output of a gate for that output to be treated as a logic 1. Any voltage below this level will not be accepted as a logic 1 output.
   - $V_{IL}(max)$*:* It is the maximum voltage level that can be treated as logic 0 at the input of the gate. Any voltage above this level will not be treated as a logic 0 input by the logic circuit.

- $V_{IH}(min)$: It is the maximum voltage level that can be treated as logic 0 at the output of the gate. Any voltage above this level will not be treated as a logic 0 output.

7. ***Current parameter:-***
   - I_{IH}: The current that flows into an input when a specified HIGH level voltage is applied to that input.
   - I_{IL}: The current that flows into an input when a specified LOW level voltage is applied to that input.
   - I_{OH}: The current that flows from an output in a logic 1 state under specified load conditions.
   - I_{OL}: The current that flows from an output in a logic 0 state under specified load conditions.
8. ***Noise margin:-***
   - When the digital circuits operate in noisy environment the gates may malfunction if the noise is beyond certain limits.
   - The noise immunity of a logic circuit refers to the circuit's ability to tolerate noise voltages at its inputs.
   - A quantitative measure of noise immunity is called *noise margin*.
9. ***Operating temperature:-***
   - The IC gates and other circuits are temperature sensitive being semiconductor devices.
   - However, they are designed to operate satisfactorily over a specified range of temperature.
   - The range specified for commercial applications is 0 to 70°C, for industrial it is 0 to 85°C, and for military applications it is -55°C to 125°C.
10. ***Speed power product:-***
    - A common means for measuring and comparing the overall performance of an IC family is the *speed power product*, which is obtained by multiplying the gate propagation delay by the gate power dissipation.
    - A low value of speed power product is desirable. The smaller the product, the better the overall performance.
    - It is figure of merit of an IC family.

TTL v/s CMOS v/s ECL

Characteristic	TTL	CMOS	ECL
Power Input	Moderate	Low	Moderate-High
Frequency limit	High	Moderate	Very high
Circuit density	Moderate-high	High-very high	Moderate
Circuit types per family	High	High	Moderate

Logic Family	Propagation delay time (ns)	Power dissipation per gate (mW)	Noise Margin (V)	Fan-in	Fan-out	Cost
TTL	9	10	0.4	8	10	Low
CMOS	<50	0.01	5	10	50	Low
ECL	1	50	0.25	5	10	High

Transistor-Transistor Logic (TTL)

- The TTL or T^2L family is so named because of its dependence on transistors alone to perform basic logic operations.
- It is the most popular logic family. It is also the most widely used bipolar digital IC family.
- The TTL uses transistors operating in saturated mode. It is the fastest of the saturated logic families.
- The basic TTL logic circuit is the NAND gate. Good Speed, low manufacturing cost, wide range of circuits, and the availability in SSI and MSI are its merits.
- Tight V_{CC} tolerance, relatively high-power consumption, moderate packing density, generation of noise spikes and susceptibility to power transients are its demerits.
- The TTL logic family consists of several subfamilies or series such as:
- *Standard TTL, High Speed TTL, Low power TTL, Schottky TTL, Low power Schottky TTL, Advanced Schottky TTL, Advanced low power Schottky TTL and F(fast)TTL.*
- The differences between the various TTL subfamilies are in their electrical characteristics such as delay time, power dissipation, switching speed, fan-out, fan-in, noise margin, etc. For Standard TTL, propagation delay time = 9 ns, power dissipation per gate = 10 mW. now margin = 0.4 mV, fan-in = 8, and fan-out = 10.
- For standard TTL, 0 V to 0.8 V is treated as a logic 0 and 2 V to 5 V is treated as logic 1.
- Signals in 0.8 V to 2 V range should not be applied as input as the corresponding response will be independent.
- If a terminal is left open in TTL, it is equivalent to connecting it to HIGH, i.e. +5 V.
- But such a practice is not recommended, since the floating TTL is extremely susceptible to picking up noise signals that can adversely affect the operation of the device.

Schottky TTL

- The standard TTL, low power TTL, and high speed TTL series operate using saturated switching
- When a transistor is saturated, excess charge carriers will be stored in the base region and they must be removed before the transistor can be turned off.
- So, owing to storage time delay, the speed is reduced.
- The Schottky TTL 748 series reduces this storage time delay by not allowing the transistor to go into full saturation.
- This is accomplished by using a Schottky barrier diode (SBD) between the base and the collector of each transistor.
- Virtually, all modern TTL devices incorporate this so-called Schottky clamp.
- The SBD has a forward voltage of only 0.25 V. The circuits in the 74S series also use smaller resistance values to improve the speed of operation.
- The speed of the 74S series is twice that of the 74H series.

- Schottky TTL has more than three times the switching speed of standard TTL, at the expense of approximately doubling the power consumption.

Tri-State TTL

- The third TTL configuration is the tri-state configuration.
- It utilizes the advantage of high speed of operation of the totem-pole configuration and wire ANDing of the open-collector configuration.
- It is called the tri-state TTL, because it allows three possible output states: HIGH, LOW, and HIGH impedance (Hi-Z).
- In the Hi-Z state, both the transistors in the totem-pole arrangement are turned off, so that the output terminal is a HIGH impedance to ground or VCC.
- In fact, the output is an open or floating terminal, that is, neither a LOW nor a HIGH.
- In practice, the output terminal is not an exact open circuit, but has a resistance of several MΩ or more relative to ground and V_{CC}.