Program : **B.Tech**

Subject Name: **Digital Systems**

Subject Code: **CS-304**

Semester: **3rd**

**Subject Name:** Digital Systems **Subject Code**: CS303

# Subject Notes

## UNIT-I

Review of number systems and number base conversions. Binary codes, Boolean algebra, Boolean functions, Logic gates. Simplification of Boolean functions, Karnaugh map methods, SOP-POS simplification, NAND-NOR implementation

### 1.1 NUMBER SYSTEMS:

**BINARY NUMBER SYSTEM : -** This number system has a base or radix of 2. The symbols or digits used in this system are 0 & 1.

**OCTAL NUMBER SYSTEM : -** This number system has a base or radix of 8. The symbols or digits used in this system are 0 through 7 i.e.( 0, 1, 2, 3, 4, 5, 6, 7 )

**DECIMAL NUMBER SYSTEM :-** This number system has a base or radix of 10. The symbols or digits used in this system are 0 through 9 i.e. ( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 )

**HEXADECIMAL NUMBER SYSTEM :-** This number system has a base or radix of 16. The symbols or digits used in this system are 0 through F i.e. ( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F )

**(1) BINARY TO DECIMAL CONVERSION:-**

**$( 1111 . 11010 )_2 = ( ? )_{10}$**

**<u>Integeral part</u> :** $(1111)_2 = ( 1 \times 2^3 ) + ( 1 \times 2^2 ) + ( 1 \times 2^1 ) + ( 1 \times 2^0 )$

$= 8 + 4 + 2 + 1$

$= 15$

ie. $(1111)_2 = (15)_{10}$

**<u>Fractional Part</u> :** $(0.11010)_2 = ( 1\times 1/2 ) + ( 1\times 1/4 ) + ( 0\times 1/8 ) + ( 1 \times 1/16 ) + ( 0\times 1/32 )$

$= 0.5 + 0.25 + 0 + 0.0625 + 0$

$= 0.8125$

ie. $(0.11010)_2 = ( 0.8125 )_{10}$

Thus $( 1111.11010 )_2 = ( 15 . 8125 )_{10}$

**2) DECIMAL TO BINARY CONVERSION :-**

**$( 15 . 812 )_{10} = ( ? )_2$**

**<u>Integeral part</u> :**

| | | |
|---|---|---|
| 2 | 15 | 1 |
| 2 | 7 | 1 |
| 2 | 3 | 1 |
| | 1 | |

$\uparrow$

$\boxed{\text{ie. } ( 15 )_{10} = ( 1111 )_2}$

**<u>Fractional Part</u> :**

$( 0.812 \times 2 ) = 1.624 \rightarrow 1$
$( 0.624 \times 2 ) = 1.248 \rightarrow 1$
$( 0.248 \times 2 ) = 0.496 \rightarrow 0$
$( 0.496 \times 2 ) = 0.992 \rightarrow 0$
$( 0.992 \times 2 ) = 1.984 \rightarrow 1$

$\downarrow$

$\boxed{\text{ie. } ( 0 . 812 )_{10} = ( 0.11001 )_2}$

Thus $( 15 . 812 )_{10} = ( 1111 . 11001)_2$

**3) OCTAL TO DECIMAL CONVERSION:-**

**( 57 . 245 )$_8$ = ( ? )$_{10}$**

**Integeral part :**

$(57)_8 = (5 \times 8^1) + (7 \times 8^0)$

$= 40 + 7$

$= 47$

ie. $(57)_8 = (47)_{10}$

**Fractional Part :**

$(0.245)_8 = (2\times 1/8) + (4\times 1/64) + (5\times 1/512)$

$= 0.25 + 0.0625 + 0.0097$

$= 0.3222$

ie. $(0.245)_8 = (0.3222)_{10}$

Thus $(57.245)_8 = (47.3222)_{10}$

## 4) DECIMAL TO OCTAL CONVERSION :-

**( 303 . 322 )$_{10}$ = ( ? )$_8$**

**Integeral part :**

| | | |
|---|---|---|
| 8 | 303 | 7 |
| 8 | 37 | 5 |
| | 4 | |

ie. $(303)_{10} = (457)_8$

**Fractional Part :**

$(0.322 \times 8) = 2.576 \rightarrow 2$

$(0.576 \times 8) = 4.608 \rightarrow 4$

$(0.608 \times 8) = 4.864 \rightarrow 4$

$(0.864 \times 8) = 6.912 \rightarrow 6$

$(0.912 \times 8) = 7.296 \rightarrow 7$

ie. $(0.322)_{10} = (0.24467)_8$

Thus $(303.322)_{10} = (457.24467)_8$

## 5) HEXADECIMAL TO DECIMAL CONVERSION :-

**( EA6 . 2FA )$_{16}$ = ( ? )$_{10}$**

**Integeral part :** $(EA6)_{16} = (E \times 16^2) + (A \times 16^1) + (6\times 16^0)$

$= (14\times 16^2) + (10 \times 16^1) + (6\times 1)$

$= 3584 + 160 + 6$

$= 3750$

ie. $(EA6)_{16} = (3750)_{10}$

**Fractional Part :** $(0.2FA)_{16} = (2\times 1/16) + (F\times 1/16^2) + (A\times 1/16^3)$

$= (2\times 1/16) + (15\times 1/256) + (10\times 1/4096)$

$= 0.125 + 0.0586 + 0.00244$

$= 0.18604$

ie. $(0.2FA)_{16} = (0.18604)_{10}$

Thus $(EA6.2FA)_8 = (3750.18604)_{10}$

## 6) DECIMAL TO HEXADECIMAL CONVERSION:

$(3750.365)_{10} = (?)_{16}$

**Integeral part :**

$$\begin{array}{c|c|c} 16 & 3750 & 6 \rightarrow 6 \\ \hline 16 & 234 & 10 \rightarrow A \\ \hline & 14 & \rightarrow E \end{array} \uparrow \qquad \boxed{\text{ie. } (3750)_{10} = (EA6)_{16}}$$

**Fractional Part :**

$$\begin{array}{lcll} (0.365 \times 16) & = & 5.84 & \rightarrow 5 \rightarrow 5 \\ (0.84 \times 16) & = & 13.44 & \rightarrow 13 \rightarrow D \\ (0.44 \times 16) & = & 7.04 & \rightarrow 7 \rightarrow 7 \\ (0.04 \times 16) & = & 0.64 & \rightarrow 0 \rightarrow 0 \\ (0.64 \times 16) & = & 10.24 & \rightarrow 10 \rightarrow A \end{array} \downarrow \qquad \boxed{\text{ie. } (0.365)_{10} = (0.5D70A)_{16}}$$

Thus $(3750.365)_{10} = (EA6.5D70A)_{16}$

**7) BINARY TO HEXADECIMAL CONVERSION:**

$(1001111010100110.001011111010)_2 = (\,?)_{16}$

**Integeral part :**

$(1001111010100110)_2 = \{\ \overleftarrow{1001,\quad 1110,\quad 1010,\quad 0110}\ \}$

$= (\underbrace{1001}_{9},\ \underbrace{1110}_{E},\ \underbrace{1010}_{A},\ \underbrace{0110}_{6})_2$

ie. $(1001111010100110)_2 = (9EA6)_{16}$

**Fractional Part :**

$(0.001011111010)_2 = \{\ \overrightarrow{\underbrace{0010}_{2},\ \underbrace{1111}_{F},\ \underbrace{1010}_{A},}\ \}$

ie. $(0.001011111010)_2 = (0.2FA)_{16}$

Thus $(1001111010100110.001011111010)_2 = (9EA6.2FA)_{16}$

**8) HEXADECIMAL TO BINARY CONVERSION:**

**$(99E.2FA)_{16} = (\,?\,)_2$**

**Integeral part :**

$(99E)_{16} = \{\overbrace{1001}^{9}\ \ \overbrace{1001}^{9}\ \ \overbrace{1110}^{E}\} = (10011001110)_2$

ie. $(99E)_{16} = (10011001110)_2$

**Fractional Part :**

$(0.2FA)_{16} = \{\overbrace{0010}^{2}\ \ \overbrace{1111}^{F}\ \ \overbrace{1010}^{A}\} = (0.001011111010)_2$

Thus $(99E.2FA)_{16} = (1001111010100110.001011111010)_2$

**9) OCTAL TO BINARY CONVERSION:**

**$(404.245)_8 = (?)_2$**

**Integeral part :**

$(404)_8 = \{\underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{100}_{4}\} = (100000100)_2$

ie. $(404)_8 = (100000100)_2$

**Fractional Part :**

$(0.245)_8 = \{\underbrace{010}_{2}\ \underbrace{100}_{4}\ \underbrace{101}_{5}\} = (0.010100101)_2$

Thus $(404.245)_8 = (100000100.010100101)_2$

**10) BINARY TO OCTAL CONVERSION:**

**$(10011110.00101)_2 = (?)_8$**

**Integeral part :**

$$(10011110)_2 = \{010,\ 011,\ 110\} \xleftarrow{\quad}$$

$$= (\underbrace{010,}_{2}\ \underbrace{011,}_{3}\ \underbrace{110}_{6})_2$$

ie. $(10011110)_2 = (236)_8$

**Fractional Part :**

$$(0.00101)_2 = \{\underbrace{001,}_{1}\ \underbrace{010}_{2}\} \xrightarrow{\quad}$$

ie. $(0.00101)_2 = (0.12)_8$

Thus $(10011110.00101)_2 = (236.12)_8$

**11) OCTAL TO HEXADECIMAL CONVERSION:**

**$(174654.273054)_8 = (?)_{16}$**

**Integeral part :**

$(174654)_8 = \{\underbrace{001}_{1}\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{110}_{6}\ \underbrace{101}_{5}\ \underbrace{100}_{4}\}$

$= (\underbrace{0000,}_{0}\ \underbrace{1111,}_{F}\ \underbrace{1001,}_{9}\ \underbrace{1010,}_{A}\ \underbrace{1100}_{C})_2$

ie. $(174654)_8 = (F9AC)_{16}$

**Fractional Part :** $\underbrace{}_{2}\ \underbrace{}_{7}\ \underbrace{}_{3}\ \underbrace{}_{0}\ \underbrace{}_{5}\ \underbrace{}_{4}$

$(0.273054)_8 = 010 \quad 111 \quad 011 \quad 000 \quad 101 \quad 100$

$= \overrightarrow{(0\,.\,\underbrace{0101}_{5},\ \underbrace{1101}_{D},\ \underbrace{1000}_{8},\ \underbrace{1011}_{B},\ \underbrace{0000}_{0})}$

ie. $(0.273054)_8 = (0\,.\,5D8B0)_{16}$

Thus $(174654\,.\,273054)_8 = (F9AC\ \ .\,5D8B)_{16}$

**12) HEXADECIMAL TO OCTAL CONVERSION:**

$(F9AC\,.\,5D8B)_{16} = (\,?\,)_8$

<u>Integeral part</u> :

$(F9AC)_{16} = \underleftarrow{\{\overbrace{1111}^{F}\ \ \overbrace{1001}^{9}\ \ \overbrace{1010}^{A}\ \ \overbrace{1100}^{C}\}} = (1111100110101100)_2$

ie. $(F9AC)_{16} = (1,\ \ 111,\ \ 100,\ \ 110,\ \ 101,\ \ 100)_2$

$= (001,\ \ 111,\ 100,\ \ 110,\ \ 101,\ \ 100)_2 \quad = \quad 174654$

ie. $(F9AC)_{16} = (174654)_8$

<u>Fractional Part</u> :

$(0\,.\,5D8B)_{16} = \underrightarrow{\{\overbrace{0101}^{5}\ \ \overbrace{1101}^{D}\ \ \overbrace{1000}^{8}\ \ \overbrace{1011}^{B}\}} = (0\,.\,010,\ 111,\ 011,\ 000,\ 101,\ 100)_2$

$= (\underbrace{010}_{2},\ \ \underbrace{111}_{7},\ \ \underbrace{011}_{3},\ \ \underbrace{000}_{0},\ \ \underbrace{101}_{5},\ \underbrace{100}_{4})_2$

$= (0\,.\,273054)_8$

ie. $(0.5D8B) = \ = (0\,.\,273054)_8$

Thus $(F9AC\,.\,5D8B)_{16} = (174654\ \ .\,273054)_8$

**1.2 CODES**

Numbers, letters or words are represented by a specific group of symbols, called code.

**1) Weighted code:** Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Example: Straight bit binary code, BCD code.

**Straight bit binary code:**

| Decimal Number | 2 | | | |
|---|---|---|---|---|
| Positional weights | $2^3 = 8$ | $2^2 = 4$ | $2^1 = 2$ | $2^0 = 1$ |
| Binary Code | 0 | 0 | 1 | 0 |

**Binary Coded Decimal (BCD) code:**

In this code each decimal digit (0 to 9) is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the Binary, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|

| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |
|---|---|---|---|---|---|---|---|---|---|---|

**Advantages of BCD Codes**

1. It is very similar to decimal system.
2. We need to remember binary equivalent of decimal numbers 0 to 9 only.

**Disadvantages of BCD Codes**

1. The addition and subtraction of BCD have different rules.
2. The BCD arithmetic is little more complicated.
3. BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

**2) Non-weighted code:** In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

**Excess-3 code**

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess-3 codes is obtained, as follows

Decimal Number $\longrightarrow$ (8421) BCD $\longrightarrow$ ADD 3 i.e.(+0011) $\longrightarrow$ Excess-3

Example:

| Decimal | BCD | | | | Excess-3 | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

**Gray Code**

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that has only one bit will change each time the decimal number is incremented . As only one bit changes at a time, the gray code is called as a unit distance code. Gray code cannot be used for arithmetic operation.

**3) Alphanumeric codes**

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

1. American Standard Code for Information Interchange (ASCII).
2. Extended Binary Coded Decimal Interchange Code (EBCDIC).
3. Five bit Baudot Code.

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

## 1.3 CODE CONVERSION:

**1. ( 0010)$_{BCD}$ to gray code**

Steps: 1) Write the MSB bit as it is i.e. 0

2) Start EXORing the consecutive bits from LHS i.e. 0 (EXOR) 0 = 0

3) 0 (EXOR) 1 = 1, 1 (EXOR) 0 = 1.

4) Final Result. ( 0011 ) $_{gray}$

| BCD code | | | | Gray code | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

**2. ( 0010)$_{gray}$ to BCD code**

Steps: 1) Write the MSB bit as it is i.e. 0

2) Starting from LHS, EXORing the result obtained in step 1 with 2nd bits i.e. 0 (EXOR) 0 = 0

3) Follow step 2.

4) Final Result. ( 0010 ) $_{BCD}$

| Gray code | | | | BCD code | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

## 1.4 BINARY ARITHMETIC:

**Unsigned Binary Numbers:** In unsigned binary numbers all the bits are used for representing only the magnitude of the corresponding decimal number. For example, the smallest 8 bit binary number is 0000 0000 and the largest 8 bit binary number is 1111 1111. Hence the total range of unsigned 8 bit binary number is from (00)$_H$ to (FF)$_H$ or from (00)$_{10}$ to (255)$_{10}$ . With 16- bit binary numbers, the total range is from (0000)$_H$ to (FFFF)$_H$

$$\underbrace{1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0}_{\text{All the bits are used for representing only the magnitude}}$$

**Sign- Magnitude Numbers**: Binary numbers which contains a sign bit followed by magnitude bits are called Sign- Magnitude Numbers. 0 is used to represent the (+) sign and 1 is used to represent a (-) sign. The MSB of binary number is used to represent the sign and remaining bit is used to represent the magnitude. MSB represents the sign and rest of bits represent the magnitude.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 / 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| **MSB** | **MAGNITUDE** | | | | | | |

For an 8-bit sign-magnitude number, the largest negative number is (-127)$_{10}$ and positive number is (+127)$_{10}$.

### 1) BINARY ADDITION:-

1. 0 + 0 = Sum 0 with carry of 0.
2. 0 + 1 = Sum 1 with carry of 0.
3. 1 + 0 = Sum 1 with carry of 0.
4. 1 + 1 = Sum 0 with a carry of 1.
5. 1 + 1 + 1 = Sum 1 with carry of 1.

**Example: Add (111011.1101)$_2$ with (011111.0110)$_2$**

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | | | | **carry** |
| | 1 | 1 | 1 | 0 | 1 | 1 | . | 1 | 1 | 0 | 1 | **Augend** |
| + | 0 | 1 | 1 | 1 | 1 | 1 | . | 0 | 1 | 1 | 0 | **Addend** |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | . | 0 | 0 | 1 | 1 | **sum** |

**2) BINARY SUBTRACTION:**

The basic principles of binary subtraction include the following:

**A)** 0 – 0 = 0. **B)** 1 – 0 = 1. **C)** 1 – 1 = 0. **D)** 0 – 1 = 1 with a borrow of 1 from the next more significant bit.

**Example: Subtract $(11111.011)_2$ from $(111011.1101)_2$**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 0 | 1 | 1 | . | 1 | 1 | 0 | 1 | **Minuend** |
| – | 0 | 1 | 1 | 1 | 1 | 1 | . | 0 | 1 | 1 | 0 | **Subtraend** |
| | 0 | 1 | 1 | 1 | 0 | 0 | . | 0 | 1 | 1 | 1 | **Difference** |

**3) BINARY SUBTRACTION USING 1'S COMPLEMENT METHOD:**

1's complement of any binary number is obtained by subtracting each binary bit by 1.

For example: $(1110011)_2 \xrightarrow{\text{1's complement}} (1111111 - 1110011) = (0001100)_2$

**Example(a) Subtract (5 – 3) using 1's complement method.**

$(5)_{10} = (0\ 1\ 0\ 1)_2$ and $(3)_{10} = (0\ 0\ 1\ 1)_2$

First obtained 1's complement of negative number i.e.1's complement of 3 = 1 1 0 0

Second add the result with positive number i.e

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 0 | 1 |
| + | 1 | 1 | 0 | 0 |
| **End around carry 1** | 0 | 0 | 0 | 1 |

Now in the result we can see that there is an overflowing bit/end around carry which we have to add with the remaining result.

| | | | | |
|---|---|---|---|---|
| | 0 | 0 | 0 | 1 |
| + | | | | 1 |
| | 0 | 0 | 1 | 0 |

**$(0010)_2=(2)_{10}$ is the desired result.**

**(b) Subtract (3 – 5) using 1's complement method.**

$(5)_{10} = (0\ 1\ 0\ 1)_2$ and $(3)_{10} = (0\ 0\ 1\ 1)_2$

First obtained 1's complement of negative number i.e.1's complement of 5 = 1 0 1 0

Second add the result with positive number i.e

| | | | | |
|---|---|---|---|---|
| | 0 | 0 | 1 | 1 |
| + | 1 | 0 | 1 | 0 |
| **No carry bit generated** | 1 | 1 | 0 | 1 |

From the result, we can see that no carry bit/end around carry bit is generated. So to get the desired result, take the 1's complement of the result and attach a negative sign i.e.

1's complement of $(1101)_2$ is $-(0010)_2$

**$-(0010)_2= -(2)_{10}$ is the desired result.**

**4) BINARY SUBTRACTION USING 2'S COMPLEMENT METHOD:**

2's complement is obtained by adding 1 to the 1's complement.

For example, we have to find out 2's complement of 0100.

we have to subtract 0100 from 1111 since it is the highest four digit number to find out 1's complement i.e. 1111 – 0100 = 1011. Hence, 2's complement will be 1011 + 1 = 1100.

**Example (a) Subtract (65 – 63) using 2's complement method.**

1. Find 2's complement of the negative number.i.e.

   2's complement of 63 = 1000001

2. Add the positive number with 2's complement of the negative number.i.e.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| + | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 Carry generated | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

3. Discard the carry generated.
4. After discarding the carry, keep the result which is the result of subtraction.

**i.e. Final answer is (0000010)**

**(b) Subtract (63 – 65) using 2's complement method.**

1. Find 2's complement of the negative number.i.e.2's complement of 65 = 0111111
2. Add the positive number and 2's complement of the negative number.i.e.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| + | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| No Carry generated | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

From the result, we can see that no carry bit/end around carry bit is generated. So to get the desired result, take the 2's complement of the result and attach a negative sign i.e.
2's complement of $(1111110)_2$ is $-(0000010)_2$ **i.e. Final answer is –(0000010).**

**5) BINARY MULTIPLICATION:**

The basic rules of multiplication are listed as follows:
1. 0 × 0 = 0.
2. 0 × 1 = 0.
3. 1 × 0 = 0.
4. 1 × 1 = 1.

**Example: Multiply $(10.11)_2$ by $(11)_2$**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 0 | . | 1 | 1 | **Multiplicand** |
| | | | X | | 1 | 1 | **Multiplier** |
| | | 1 | 0 | . | 1 | 1 | |
| + | 1 | 0 | 1 | . | 1 | 0 | |
| 1 | 0 | 0 | 0 | . | 0 | 1 | **Product** |

**6) BINARY DIVISION:**

**Example: Divide $(110001)_2$ by $(111)_2$**

| | | |
|---|---|---|
| | | |
| Divisor 111 | 110001-Divident<br>-0111 | 111 Quotient |
| | 01010<br>-111 | |
| | 0111<br>-111 | |
| | 0000-Remainder | |

## 1.5 BOOLEAN ALGEBRA:

Boolean algebra or switching algebra is a system of mathematical logic to perform different mathematical operations in binary system. There is only three basis binary operations, AND, OR and NOT by which all simple as well as complex binary mathematical operations are to be

done. There are many rules in Boolean algebra by which those mathematical operations are done. In Boolean algebra, the variables are represented by Capital Letter like A, B, C etc and the value of each variable can be either 1 or 0, nothing else. In Boolean algebra an expression given can also be converted into a logic diagram using different logic gates like AND gate, OR gate and NOT gate, NOR gates, NAND gates, XOR gates, XNOR gates etc.
Some basic logical Boolean operations,

| **AND operation** | **OR operation** | **NOT operation** |
|---|---|---|
| 0.0 = 0 | 0 + 0 = 0 | (1)' = 0 |
| 1.0 = 0 | 0 + 1 = 1 | (0)' = 1 |
| 0.1 = 0 | 1 + 0 = 1 | |
| 1.1 = 1 | 1 + 1 = 1 | |

## Some basic laws and theorems for Boolean Algebra:

1] Idempotent Law : (i) A + A = A (ii) A . A = A

2] Commutative Law : (i) A+ B = B+ A (ii) A. B = B . A

3] Associative Law : (i) (A + B) + C = A + (B+ C) (ii) (A . B) . C = A . ( B . C)

4] Distributive Law : (i) A + (B . C) = (A + B) . (A+ C) (ii) A . (B+ C) = ( A . B)+ ( A . C)

5] Complement Law : (i) A + A' = 1 (ii) A . A' = 0

6] Double Complement Law : (A')' = A

7] Identity Law : (i) A + 0 = A (ii) A . 1 = A

8] Null ( Dominance ) Law : (i) A + 1 = 1 (ii) A . 0 = 0

9] Absorption Law : (i) A. (A + B) = A (ii) A + (A . B) = A

(iii) A . (A'+B) = ( A . B ) (iv) A + A'. B = ( A + B )

### 1.6 De Morgan's Theorem:

**I Theorem :** Complement of sum is equal to the product of complements.

(A+B)' = A'.B'

**II Theorem :** Complement of product is equal to the sum of complements.

(A.B)' = A' + B'

**Proof :**

| Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| A | B | (A+B)' | A'.B' | (A.B)' | A'+B' |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

Column for (A+B)' and A'.B' are same. Column for (A.B)' and A' + B' are same. Hence proved.

### 1.7 DUALITY THEOREM :

The Duality theorem states that the Dual of a Boolean expression can be obtained by :
(i) Replacing OR operator by AND operator. (iii) Replacing AND operator by OR operator.
(ii) Replacing 0s by 1s and 1s by 0s.
(iv) Complementing the variables ie. A is replaced by A' and A' is replaced by A.
Example of Duality principle are stated below :

(i) A + A = A (ii) A . A = A BY DUALITY :- (i) A’ . A’ = A’ (ii) A’ + A’ = A’

**1.8 APPLICATION OF BOOLEAN ALGEBRA :**

**Simplification of Boolean Functions:-**

**(1) Y = A.B.C + A.B’.C + A.B.C’**

| | |
|---|---|
| Y= A.C.(B+B’) + A.B.C’ | {B + B’ = 1 Complementation law} |
| Y= A.C.(1) + A.B.C’ | {A.C .1 = A.C – Identity law} |
| Y= A.C + A.B.C’ | |
| Y= A.[C + C’.B] | {C + C’.B = C + B –Absorption law} |
| **Y= A.[C+B]** | |

**(2) Y = A.B.C’ + A.B’.C’ + A.B.C + A.B’.C + A’.B.C**

| | |
|---|---|
| Y = A.B.(C+C’) + A.B’. (C+C’) + A’.B.C | {C + C’ = 1 - Complementation law} |
| Y = A.B.(1) + A.B’.(1) + A’.B.C | {A.1 = A - Identity law} |
| Y = A.B + A.B’ + A’.B.C | |
| Y = B.[A + A’.C] + A.B’ | {A+A’.C = A+C - Absorption law} |
| Y= B.[A+C] + A.B’ | {Using Distributive law} |
| Y= A.B + B.C + A.B’ | |
| Y = A.[B+B’] + B.C | {B + B’ = 1 - Complementation law} |
| Y = A.(1) + B.C | {A.1 = A - Identity law} |
| **Y = A + B.C** | |

**(3)Y = A.B.C + A’.B.C + A.B’.C + A.B.C’ + A’.B’.C’**

| | |
|---|---|
| Y = A.B.(C+C’) + A’.B.C + A.B’.C + A’.B’.C’ | {C + C’ = 1 - Complementation law} |
| Y = A.B.(1) + A’.B.C + A.B’.C + A’.B’.C’ | |
| Y = A.(B+B’.C) + A’.B.C + A’.B’.C’ | {(B + B’.C) = B+C - Absorption law} |
| Y = A.(B+C) + A’.B.C + A’.B’.C’ | {Using Distributive law} |
| Y = A.B + A.C + A’.B.C + A’.B’.C’ | {Using Associative law} |
| Y = A.B + A’.B.C + A.C + A’.B’.C’ | {A+A’.C = A+C -Absorption law} |
| Y = B.(A + A’.C) + A.C + A’.B’.C’ | |
| **Y = B.(A+C) + A.C + A’.B’.C’** | |

**1.9 TECHNIQUES TO MINIMIZE THE BOOLEAN FUCTION:**

Two types of minimization techniques: 1) Karnaugh-Map Method 2) Quines McCluskey Method

**1) The Karnaugh map method (K-Map)**

A Karnaugh map is a graphical representation of the logic system. It can be drawn directly from either minterm (sum-of-products) or maxterm (product-of-sums) Boolean expressions.

**2) Sum of Product(Minterm):** Each minterm is obtained from an ANDterm of the ‘n’ variables, with each variable being primed if the corresponding bit of the binary numbers is a ‘0’ and unprimed if a ‘1’. (**$m_j$**) is the symbol of minterm where subscript ‘j’ is the decimal equivalent of binary number of minterm designated. Table of minterms as follows:

| Decimal number | Variables X Y Z | Term | Designation |
|---|---|---|---|
| 0 | 0 0 0 | X’.Y’.Z’ | $m_0$ |

| | | | |
|---|---|---|---|
| 1 | 0 0 1 | X’.Y’.Z | $m_1$ |
| 2 | 0 1 0 | X’.Y.Z’ | $m_2$ |
| 3 | 0 1 1 | X’.Y.Z | $m_3$ |
| 4 | 1 0 0 | X.Y’.Z’ | $m_4$ |
| 5 | 1 0 1 | X.Y’.Z | $m_5$ |
| 6 | 1 1 0 | X.Y.Z’ | $m_6$ |
| 7 | 1 1 1 | X.Y.Z | $m_7$ |

Example: Boolean expression in SOP form:
F = X’.Y’.Z’ + X’.Y’.Z + X’.Y.Z + X.Y’.Z + X.Y.Z
F(A,B,C,D) = ∑m ( 0, 1, 3, 5, 7 )

**3) Product of Sum(Maxterm):** Each maxterm is obtained from an ORterm of the ‘n’ variables, with each variable being primed if the corresponding bit of the binary numbers is a ‘1’ and unprimed if a ‘0’. ($M_j$) is the symbol of maxterm where subscript ‘j’ is the decimal equivalent of binary number of maxterm designated. Table of maxterms as follows:

| Decimal number | Variables X Y Z | Term | Designation |
|---|---|---|---|
| 0 | 0 0 0 | X+Y+Z | $M_0$ |
| 1 | 0 0 1 | X+Y+Z’ | $M_1$ |
| 2 | 0 1 0 | X+Y’+Z | $M_2$ |
| 3 | 0 1 1 | X+Y’+Z’ | $M_3$ |
| 4 | 1 0 0 | X’+Y+Z | $M_4$ |
| 5 | 1 0 1 | X’+Y+Z’ | $M_5$ |
| 6 | 1 1 0 | X’+Y’+Z | $M_6$ |
| 7 | 1 1 1 | X’+Y’+Z’ | $M_7$ |

Example: Boolean expression in POS form:
F = (X’+Y+Z’)   ( X’+Y’+Z)   (X’+Y+Z)   ( X+Y’+Z)   (X+Y+Z)
F(A,B,C,D) = ∏M (5,6,4,2,0 )

**1.10 K-MAP Representation:**

**2-Variable K-Map**

| | B' | B |
|---|---|---|
| A' | m0 | m1 |
| A | m2 | m3 |

**3-Variable K-Map**

| | B'C' | B'C | BC | BC' |
|---|---|---|---|---|
| A' | m0 | m1 | m3 | m2 |
| A | m4 | m5 | m7 | m6 |

**4-Variable K-Map**

| | C'D' | C'D | CD | CD' |
|---|---|---|---|---|
| A'B' | m0 | m1 | m3 | m2 |
| A'B | m4 | m5 | m7 | m6 |
| AB | m12 | m13 | m15 | m14 |
| AB' | m8 | m9 | m11 | m10 |

**5-Variable K-Map**

| | C'D'E' | C'D'E | C'DE | C'DE' | CDE' | CDE | CD'E | CD'E' |
|---|---|---|---|---|---|---|---|---|
| A'B' | m0 | m1 | m3 | m2 | m6 | m7 | m5 | m4 |
| A'B | m8 | m9 | m11 | m10 | m14 | m15 | m13 | m12 |
| AB | m24 | m25 | m27 | m26 | m30 | m31 | m29 | m28 |
| AB' | m16 | m17 | m19 | m18 | m22 | m23 | m21 | m20 |

**Question:** Simplify the Boolean function: F = ∑m ( 1, 3, 5, 9, 11, 13 ). Using K-Map Method.
**Solution:** F = ∑m ( 1, 3, 5, 9, 11, 13 )

| | C'.D' | C'.D | C.D | C.D' |
|---|---|---|---|---|
| **A'.B'** | 0 | **1** | **1** | 0 |
| **A'.B** | 0 | **1** | 0 | 0 |
| **A.B** | 0 | **1** | 0 | 0 |
| **A.B'** | 0 | **1** | **1** | 0 |

**The minimized Boolean function is F = C'D + B'D**

**Question :** Using Karnaugh maps, write the minimized Boolean expressions for the output functions : Y1 = A'.B.C' + A.B'.C' + A.B.C + A'.B'.C' and Y2 = A'.B'.C + A.B.C' + A'.B'.C' + A.B'.C + A.B.C + A.B'.C'
**Solution:** Y1 = ∑m ( 0, 2, 4, 7 ) and Y2 = ∑m ( 0, 1, 4, 5, 6, 7 )

**K-Map for Y1:**

| | B'C' | B'C | BC | BC' |
|---|---|---|---|---|
| **A'** | 1 | 0 | 0 | 1 |
| **A** | 1 | 0 | 1 | 0 |

**K-Map for Y2:**

| | B'C' | B'C | BC | BC' |
|---|---|---|---|---|
| **A'** | 1 | 1 | 0 | 0 |
| **A** | 1 | 1 | 1 | 1 |

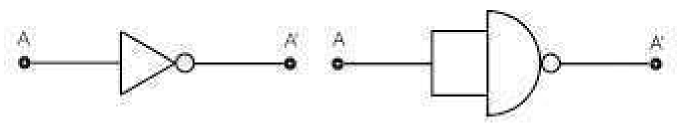
**The minimized expressions are as follows: Y1 = B'.C' + A'.C' + A.B.C and Y2 = A + B'**

**Question:** Simplify the Boolean function: F = ∏M ( 1,4,5,6,11,12,13,14,15) in POS form. Using K-Map Method.
**Solution:** F = ∏M ( 1,4,5,6,11,12,13,14,15)

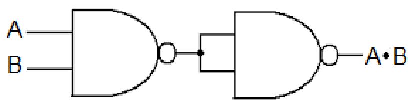
| | C'.D' | C'.D | C.D | C.D' |
|---|---|---|---|---|
| **A'.B'** | 1 | **0** | 1 | 1 |
| **A'.B** | **0** | **0** | 1 | **0** |
| **A.B** | **0** | 0 | **0** | **0** |
| **A.B'** | 1 | 1 | **0** | 1 |

**The minimized expression in POS form is F = (B'+C) (B'+D) (A+C+D') (A"+C'+D')**

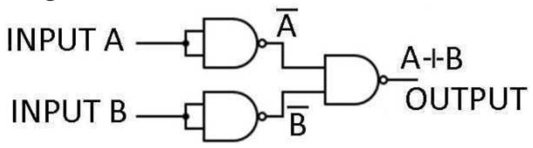
**1.11 NAND-NOR Implementation:**
NAND and NOR gate are called the universal gates and all the basic gates i.e. AND, OR, NOT and EX-OR can be implemented using these gates.
**1.11.1 NAND Implementation:**
(a) Implementing NOT gate using NAND Gate:



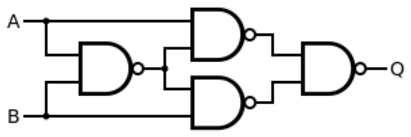
(a) Implementing AND gate using NAND Gate:



(c) Implementing OR gate using NAND Gate:

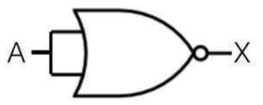


(d) ) Implementing EX-OR gate using NAND Gate:

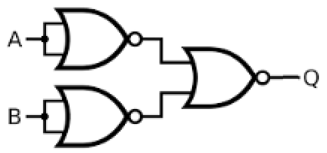


**1.11.2 NOR Implementation:**

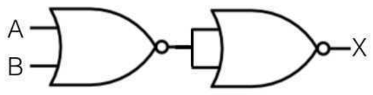
(a) Implementing NOT gate using NOR Gate:



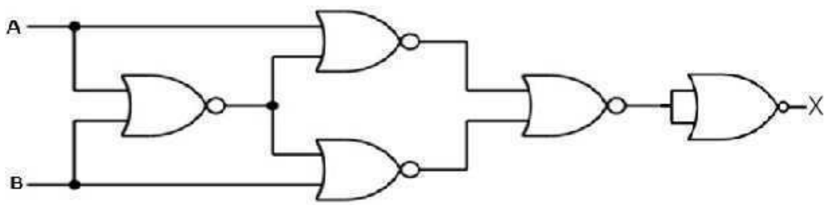
(a) Implementing AND gate using NOR Gate:



(c) Implementing OR gate using NOR Gate:

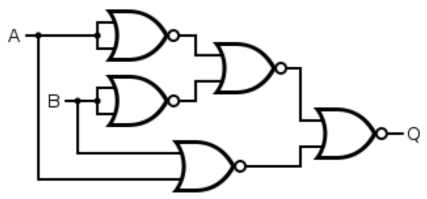


(d) ) Implementing EX-OR gate using NOR Gate:



$$X = [(A + (A + B)')' + (B + (A + B)')']'$$
$$X = [A'B + AB']$$

Or



$$X = [(A' + B')' + (A + B)']'$$
$$X = [(A.B) + (A + B)']'$$
$$X = [(AB)'.\{(A + B)'\}']$$
$$X = [(A' + B').(A + B)]$$
$$X = [A'A + A'B + B'A + B'B]$$
$$X = [A'B + AB']$$

-----------------End of Unit 1-----------------