DJK3C: DIGITAL ELECTRONICS

Unit I

Number System:

Decimal – binary – octal – hexadecimal number system – conversion from one system to another – binary arithmetic – 1's complement – 2's complement – BCD, excess 3, gray , alpha numeric codes.

Unit II

Boolean algebra:

Boolean operation – rules and laws of Boolean algebra – De Morgan's theorems – implication of expressions using Boolean algebra – Karnaugh map.

Unit III

Basic Logic gates:

AND, OR, NOT (symbol, truth table, circuit diagram, working) NAND, NOR, EX-OR, EX- NOR (symbol, truth table)

Unit IV

Combinational Circuits:

Half adder – full adder – half sub tractor – full sub tractor – binary adder – BCD adder – decoder – encoder – multiplexer – de multiplexer.

Unit V

Flip flops:

RS, JK, D, T flip flops – master slave flip flop - IC 555 timer – astable multi vibrator - mono stable multi vibrator.

Books for study and reference :

1. Digital principles & applications – Albert Paul Malvino & Leach
2. Digital Logic & Computer Design – Morris Mano.

Unit INumber System

Decimal - Binary- Octal - Hexadecimal number system – conversion from one system to another – Binary arithmetic – 1's complement – 2's complement – BCD, Excess, Gray, Alpha numeric codes

1.1 Number System:

A number is a mathematical object used to count, measure, and label. Numbers are represented by a string of digital symbols. A number system of base r is a system that uses distinct symbols for r digits. That is in a positional base$r$ numeral system r basic symbols (or digits) corresponding to the first r natural numbers including zero are used. To generate the rest of the numerals, the position of the symbol in the figure is used. The symbol in the last position has its own value, and as it moves to the left its value is multiplied by r. There are four systems of arithmetic used in digital system. These systems are Decimal, Binary, Hexadecimal and Octal.

System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

.

1.2 Decimal Number System:

The Decimal number system has a base ten. This system uses ten distinct digits 0 1 2 3 4 5 6 7 8 9 to form any number. Each digit can be used individually or they can be grouped to form a numeric value.Each of decimal digits, 0 through 9, has a place value or weight depending on its position. The weights are units, tens, hundreds, thousands and so on. The same can be expressed as the powers of its base as $10^0, 10^1, 10^2, 10^3 \cdots etc$for the integer partand $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4} \cdots etc$ for the fractional part. $10^0, 10^1, 10^2, 10^3 \cdots etc$represents the units, tens, hundreds, thousands etc. and the quantities $10^{-1}, 10^{-2}, 10^{-3}, \cdots etc$ represents one tenth, one hundredth, one thousandth etc. The integer part and fractional parts are separated by a decimal point. The position weights in decimal system is given as

$\cdots$	10^3	10^2	10^1	10^0	$\cdot$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	$\cdots$

Example:

(i) $7693 = 7 \times 10^3 + 6 \times 10^2 + 9 \times 10^1 + 3 \times 10^0$

$= 7 \times 1000 + 6 \times 100 + 9 \times 10 + 3 \times 1$

$= 7000 + 600 + 90 + 3$

(ii) $1936.46 = 1 \times 10^3 + 9 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 6 \times 10^{-2}$

$= 1000 + 900 + 30 + 6 + 0.4 + 0.06$

1.3 Binary Number System:

The base of the binary number system is two. It uses the digits0 and 1 only. The two digits 0 and 1 are called a bit. The place value of each position can be expressed in terms of powers of 2 like $2^0, 2^1, 2^2, etc$ for integer part and $2^{-1}, 2^{-2}, 2^{-3}, etc$ for the fractional part. A binary point separates the integer and fractional part. The position weights in the binary is given as

$\cdots$	2^3	2^2	2^1	2^0	$\cdot$	2^{-1}	2^{-2}	2^{-3}	2^{-4}	$\cdots$

$4\ bit\ binary\ word \Longrightarrow \quad nibble$

$8\ bit\ binary\ word \Longrightarrow \quad byte$

$16\ bit\ binary\ word \Longrightarrow \quad word$

$32\ bit\ binary\ word \Longrightarrow \quad double\ word$

1.4 Octal Number System:

The base of the octal number system is eight. It uses eight digits $0\ 1\ 2\ 3\ 4\ 5\ 6\ and\ 7$ to form a number. The place value of each position can be expressed in terms of powers of 8 like $8^0, 8^1, 8^2, etc$ for integer part and $8^{-1}, 8^{-2}, 8^{-3}, etc$ for the fractional part. An octal point separates the integer and fractional part. Sets of 3-bit binary numbers can be represented by octal numbers $(000, 001, 010, 011, 100, 101, 110, 111)$ and this can be conveniently be used for entering data in the computer. The position weights in the octal system is given as

$\cdots$	8^3	8^2	8^1	8^0	$\cdot$	8^{-1}	8^{-2}	8^{-3}	8^{-4}	$\cdots$

1.5 Hexadecimal Number System:

The Hexadecimal number system has a base of 16. It has 16 distinct digit symbols. It uses the digits $0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9$ plus the letters $A\ B\ C\ D\ E\ and\ F$. Any hexadecimal digit can be represented by a group of four bit binary sequence.That is the Hexadecimal numbersare represented by sets of 4-bit binary sequence (0000, 0001,0010, 0011, 0100,0101,0110, 0111,1000,1001,1010,1011,1100,1101,1110,1111). The position weight in the hexadecimal number system is given as

$\cdots$	16^3	16^2	16^1	16^0	$\cdot$	16^{-1}	16^{-2}	16^{-3}	16^{-4}	$\cdots$

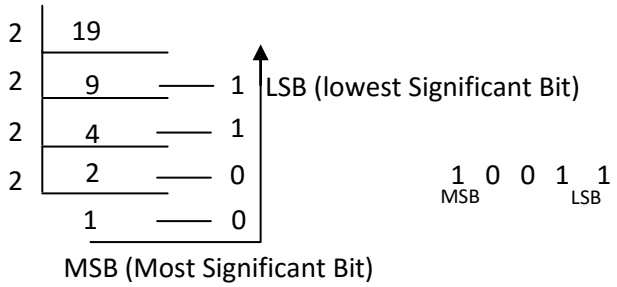
Number System

Decimal (Base 10)	**Binary (Base 2)**	**Octal (Base 8)**	**Hexadecimal (Base 16)**
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.6 Decimal to Binary Conversion:

Decimal number can be convertedto binary by repeatedly dividing the number by 2 for integer part and collecting the reminders.The remainders can be written in the reverse order(from bottom to top) to get binary result. For fractional part, it has to be multiplied by 2 successively and collecting the carries, to write from top to bottom. The multiplication is repeated till the fractional part becomes zero or the required number of significant bit is obtained.

1. Convert $(19)_{10}$ into its Binary equivalent



$$(19)_{10} = (10011)_2$$

2. Convert$(0.625)_{10}$ into its Binary equivalent

$0.625 \times 2 = 1.250$ $carry\ is$ 1 (MSB)

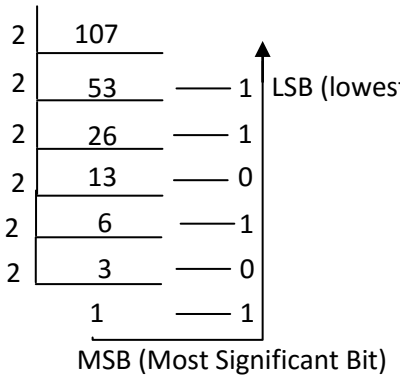
$0.250 \times 2 = 0.500$ $carry\ is$ 0

$0.500 \times 2 = 1.000$ $carry\ is$ 1 (LSB)

1 0 1
MSB LSB

$$(0.625)_{10} = (0.101)_2$$

3. Convert $(107.6875)_{10}$ to its equivalent Binary



$0.6875 \times 2 = 1.3750$ $carry\ is$ 1 (MSB)

$0.3750 \times 2 = 0.7500$ $carry\ is$ 0

$0.7500 \times 2 = 1.500$ $carry\ is$ 1

$0.5000 \times 2 = 1.000$ $carry\ is$ 1 (LSB)

1 0 1 1
MSB LSB

$$(107.6875)_{10} = (1101011.1011)_2$$

1.7 Binary to Decimal Conversion:

A binary number can be converted into decimal number by adding the products of eachbitanditscorresponding weight.

Example:

(i) $(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 4 + 0 + 1$

$= (5)_{10}$

(ii) $(10011)_2 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 16 + 0 + 0 + 2 + 1$

$= (19)_{10}$

(iii) $(0.101)_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 1 \times 0.5 + 0 + 1 \times 0.125$

$= 0.5 + 0 + 0.125$

$= (0.625)_{10}$

(iv) $(1101011.1011)_2$

$(1101011)_2 = 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^1$

$= 64 + 32 + 0 + 8 + 0 + 2 + 1$

$= (107)_{10}$

$(0.1011) \quad = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$

$= 0.5 + 0 + 0.125 + 0.0625$

$= (0.6875)_{10}$

$\therefore (1101011.1011)_2 = (107.6875)_{10}$

1.8 Hexadecimal to Decimal:

A hexadecimal number can be converted into decimal number by adding the products of each digit and its corresponding weight. The weights are power of 16.

Example:

1. Convert hexadecimal $(D5)_{16}$ to decimal

$(D5)_{16} = (13 \times 16^1 + 5 \times 16^0) \Longrightarrow \ (D)_{16} = (13)_{10}$

$= 13 \times 16 + 5 \times 1$

$= 208 + 5$

$= (213)_{10}$

$ie., (D5)_{16} = (213)_{10}$

2. Convert hexadecimal $(3FC.8)_{16}$ to decimal

$$(3FC.8)_{16} = (3 \times 16^2 + 15 \times 16^1 + 12 \times 16^0 + 8 \times 16^{-1})$$

$$\Longrightarrow (F)_{16} = (15)_{10} \& (C)_{16} = (12)_{10}$$

$$= 3 \times 256 + 15 \times 16 + 12 \times 1 + 8 \times \frac{1}{16}$$

$$= 768 + 240 + 12 + 0.5$$

$$= (1020.5)_{10}$$

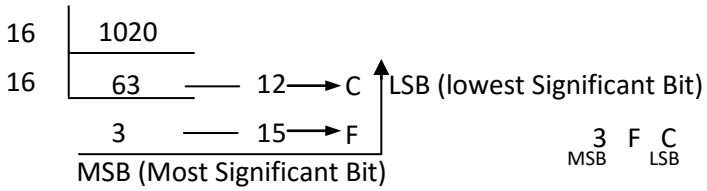
$$ie., (3FC.8)_{16} = (1020.5)_{10}$$

1.9 Decimal to Hexadecimal:

Decimal number can be converted to hexadecimal by repeatedly dividing the number by 16 for integer part and collecting the reminders. The remainders can be written in the reverse order (from bottom to top) to get hex result. For fractional part, it has to be multiplied by 16 successively and collecting the carries, to write from top to bottom. The multiplication is repeated till the fractional part becomes zero or the required numbers of significant digits are obtained.

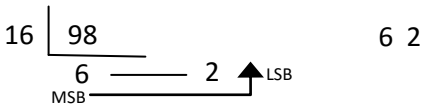
Example:

1. Convert $(1020)_{10}$ to hexadecimal



$$(1020)_{10} = (3FC)_{16}$$

2. Convert $(98.625)_{10}$ to hexadecimal



$$0.625 \times 16 = 10.000 \quad carry\ is\ \ 10 \Longrightarrow A$$

$$(98.625)_{10} = \ (62.A)_{16}$$

1.10 HexadecimaltoBinary:

Hexadecimal numbers can be converted into binary numbers by converting each hexadecimal digit to its 4-bit binary equivalent

1. Convert $(25)_H$ to Binary

   $(25)_H = \left(\underbrace{0010}_{2}\underbrace{0101}_{5}\right)_2$

2. Convert $(3A.7)_H$ to Binary

   $(3A)_H = \left(\underbrace{0011}_{3}\underbrace{1010}_{A}\right)_2$

   $(.7)_H = \left(\underbrace{.0111}_{7}\right)_2$

   $(3A.7)_H = \left(\underbrace{0011}_{3}\underbrace{1010}_{A} \cdot \underbrace{0111}_{7}\right)_2$

3. Convert $(CD.E8)_H$ to Binary

   $(CD)_H = \left(\underbrace{1100}_{C}\underbrace{1101}_{D}\right)_2$

   $(.E8)_H = \left(\underbrace{.1110}_{E}\underbrace{1000}_{8}\right)_2$

   $(CD.E8)_H = \left(\underbrace{1100}_{C}\underbrace{1101}_{D} \cdot \underbrace{1110}_{E}\underbrace{1000}_{8}\right)_2$

1.11 Binary to Hexadecimal:

Conversion from binary to hexadecimal is easily accomplished by partitioning the binary number into groups of four binary digits, starting from the binary point to the left and to the right. It may be necessary to add zeros to the last group, if it does not end in exactly four bits. Each group of 4-bits binary must be represented by its hexadecimal equivalent.

1. $(1010 \cdot 1101)_2 = (A \cdot D)_H$
2. $(110 \cdot 101)_2 = (0110 \cdot 1010)_2 = (6 \cdot A)_H$
3. $(1110 \cdot 11)_2 = (1110 \cdot 1100)_2 = (E \cdot C)_H$

1.12 Octal to Decimal:

Anoctal number can be converted into decimal number by adding the products of each digit and its corresponding weight. The weights are power of 8.

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

   $= 56 + 5$

   $= (61)_{10}$

2. $(45 \cdot 6)_8 = 4 \times 8^1 + 5 \times 8^0 + 6 \times 8^{-1}$

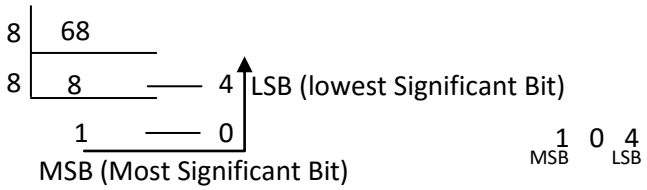
   $= 32 + 5 + 0.75$

   $= (37.75)_{10}$

1.13 Decimal to octal:

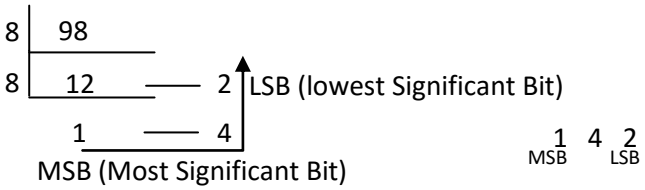
Decimal number can be converted to octal by repeatedly dividing the number by 8 for integer part and collecting the reminders. The remainders can be written in the reverse order (from bottom to top) to get octal result. For fractional part, it has to be multiplied by 8 successively and collecting the carries, to write from top to bottom. The multiplication is repeated till the fractional part becomes zero or the required numbers of significant digits are obtained.

1. Convert $(68)_{10}$ to octal



$$(68)_{10} = (104)_8$$

2. Convert $(98.625)_{10}$to Octal



$$0.625 \times 8 = 5.0000 \ \ carry\ is\ \ 5$$

$$(98.625)_{10} = (142.5)_8$$

1.14 Octal to Binary:

Octal numbers can be converted into binary numbers by converting each octal digit to its 3-bit binary equivalent

1. $(27)_8 = \left(\underbrace{010}_{2}\ \underbrace{111}_{7}\right)_2$
2. $(135)_8 = \left(\underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{101}_{5}\right)_2$
3. $(45.5)_8 = \left(\underbrace{100}_{4}\ \underbrace{101}_{5} \cdot \underbrace{101}_{5}\right)_2$

1.15 Binary to Octal:

Conversion from binary to octal is the simplest procedure by grouping the binary number into groups of three binary digits, starting from the binary point to the left and to the right. It may be necessary to add zeros to the last group, if it does not end in exactly three bits. Each group of 3-bits binary must be represented by its octal equivalent.

1. $(10\ 101)_2 = (010\ 101)_2 = (25)_8$
2. $(101011)_2 = (101\ 011)_2 = (53)_8$
3. $(11110.11)_2 = (011\ 110\ .\ 110)_2 = (36.6)_8$
4. $(11011011.1111)_2 = (011\ 011\ 011.\ 111\ 100)_2 = (333.74)_8$

1.16 Hexadecimal to Octal:

This can be achieved by first writing down the four bit binary equivalent of hexadecimal digit and then partitioning it into group of 3 bits each. Finally, the three bit octal equivalent is written down.

Example:

1. Convert $(2AB.9)_H$ to octal

$$\begin{array}{lcccccccc} Hexa\ decimal\ Number & \rightarrow & 2 & & A & & B & . & 9 \\ & & \downarrow & & \downarrow & & \downarrow & . & \downarrow \\ 4\ bit\ Binary\ Number & \rightarrow & 0010 & & 1010 & & 1011 & . & 1001 \\ 3\ bit\ Partition & \rightarrow & \underbrace{001}\ \underbrace{010}\ \underbrace{101}\ \underbrace{011} & & & & & . & \underbrace{100}\ \underbrace{100} \\ & & \downarrow & \downarrow & \downarrow & \downarrow & & . & \downarrow\ \downarrow \\ Octal\ Number & \rightarrow & 1 & 2 & 5 & 3 & . & 4 & 4 \end{array}$$

$$\therefore (2AB.9)_{16} = (1253.44)_8$$

2. Convert $(3FC.82)_H$ to octal

$$\begin{array}{lcccccccc} Hexa\ decimal\ Number & \rightarrow & 3 & & F & & C & . & 8\quad 2 \\ & & \downarrow & & \downarrow & & \downarrow & . & \downarrow\quad \downarrow \\ 4\ bit\ Binary\ Number & \rightarrow & 0011 & & 1111 & & 1100 & . & 1000\quad 0010 \\ 3\ bit\ Partition & \rightarrow & \underbrace{001}\ \underbrace{111}\ \underbrace{111}\ \underbrace{100} & & & & & . & \underbrace{100}\ \underbrace{000}\ \underbrace{100} \\ & & \downarrow & \downarrow & \downarrow & \downarrow & & . & \downarrow\ \downarrow\ \downarrow \\ Octal\ Number & \rightarrow & 1 & 7 & 7 & 4 & . & 4 & 0\quad 4 \end{array}$$

$$\therefore (3FC.82)_{16} = (1253.44)_8$$

1.17 Octal to Hexadecimal:

This can be achieved by first writing down the three bit binary equivalent of octal digit and then partitioning it into group of 4 bits each. Finally, the four bit hexadecimal equivalent is written down.

1. Convert $(16.2)_8$ to Hexadecimal

$$
\begin{array}{lcccccc}
octal\ Number & \rightarrow & 1 & 6 & . & 2 \\
 & & \downarrow & \downarrow & . & \downarrow \\
3\ bit\ Binary & \rightarrow & 001 & 110 & . & 010 \\
 & & \downarrow & \downarrow & . & \downarrow \\
4\ bit\ Partition & \rightarrow & \underbrace{0000}\underbrace{1110} & . & \underbrace{0100} \\
 & & \downarrow & \downarrow & . & \downarrow \\
Hex\ Number & \rightarrow & 0 & E & . & 4
\end{array}
$$

$$\therefore (16.2)_8 = (0E.4)_{16} = (E.4)_H$$

A zero is added to the right most group to make it a group of 4 bits and left most zeros are dropped

2. Convert $(764.352)_8$ to Hexadecimal

$$
\begin{array}{lcccccccccc}
octal\ Number & \rightarrow & 7 & 6 & 4 & . & 3 & 5 & 2 \\
 & & \downarrow & \downarrow & \downarrow & . & \downarrow & \downarrow & \downarrow \\
3\ bit\ Binary & \rightarrow & 111 & 110 & 100 & . & 011 & 101 & 010 \\
 & & \downarrow & \downarrow & \downarrow & . & \downarrow & \downarrow & \downarrow \\
4\ bit\ Partition & \rightarrow & \underbrace{0001}\underbrace{1111}\underbrace{0100} & . & \underbrace{0111}\underbrace{0101}\underbrace{0000} \\
 & & \downarrow & \downarrow & \downarrow & . & \downarrow & \downarrow & \downarrow \\
Hex\ Number & \rightarrow & 1 & F & 4 & . & 7 & 5 & 0
\end{array}
$$

$$\therefore (764.352)_8 = (1F4.750)_{16} = (1F4.75)_H$$

1.18 Binary Arithmetic:

Arithmetic operations such as addition, subtraction, multiplication and division can be performed on binary numbers.

1.18.1 Binary addition:

The addition of two Binary numbers is very similar to addition of two decimal numbers. It is key to binary subtraction, multiplication and division.The following rules are followed while adding two binary numbers.

Augend + Addend (A) (B)	Carry Sum (A) (B)	Result
0 + 0	0 0	0
0 + 1	0 1	1
1 + 0	01	1
1 + 1	1 0	10 ; read as 0 with a carry 1
1 + 1 +1	1 1	11 ; read as 1 with a carry 1

Example:

1. Add the binary numbers (i) 1011 and 1110 (ii) 10.001 and 11.110

(i)

Binary Number	Equivalent Decimal
1 1 1 Carry	
1 0 1 1	11
+ 1 1 1 0	14
Sum= 1 1 0 0 1	25

(ii)

Binary Number	Equivalent Decimal
1 ← Carry	
1 0 . 0 0 1	2 . 1 2 5
+ 1 1 . 1 1 0	+ 3 . 7 5 0
Sum= 1 0 1 . 1 1 1	5 . 8 7 5

1.18.2 Binary subtraction:

The subtraction of two Binary numbers is very similar to subtraction of two decimal numbers. Subtraction is the inverse operation of addition. The following rules are used in subtracting two binary numbers.

Minuend - Subtrahend	Difference Borrow
0 - 0	0 0
1 - 0	1 0
1 - 1	0 0
0 - 1	1 1 read as difference 1 with borrow 1

Example:

1. Subtract the binary numbers (i) 101 from 1001 (ii) 11and 10000

(i)

	Binary Number	Equivalent Decimal
	1 0 0 1	9
-	1 0 1	- 5
Difference =	1 0 0	4

(ii)

	Binary Number	Equivalent Decimal
	1 0 0 0 0	16
-	1 1	- 3
Difference =	1 1 0 1	13

1.18.3 Binary Multiplication:

Themultiplicationof two Binary numbers is very similar to multiplication of two decimal numbers. The following rules are used to multiply two binary numbers.

(i) $0 \times 0 = 0$

(ii) $0 \times 1 = 0$

(iii) $1 \times 0 = 0$

(iv) $1 \times 1 = 1$

Example:

Multiply the following binary numbers (i) 1011 and 1101 and (ii) 1.01 and 10.1

(i)

Binary Multiplication		Equivalent decimal	
Multiplicand	1011		11
Multiplier	× 1101	× 13	
	1011		33
	0000		11
	1011		143
	1011		
	10001111		

(ii)

Binary Multiplication		Equivalent decimal
Multiplicand	1.01	1.25
Multiplier	× 10.1	×2.5
	101	625
	000	240
	101	3025
11001		
11.001		3.025

1.18.4 Binary Division:

The division in Binary is exactly same as in decimal. The division by 0 is not allowed. The binary division has only two rules.

(i) $0 \div 1 = 0$ and (ii) $1 \div 1 = 1$

Example:

Divide the binary numbers (i) 11001 by 101 (ii) 1010 by 100

(i)

Binary Division

```
      101
  ____________
101 | 11001
      101
    -------
      00101
        101
    -------
      00000
    -------
```

Equivalent decimal

```
    5
  ____
5 | 25
    25
  ----
    0
  ----
```

(ii)

Binary Division

```
      10.1
  ____________
100 | 1010
      100
    -------
      00100
        100
    -------
      00000
    -------
```

Equivalent decimal

```
    2.5
  ____
4 | 10
    8
  ----
    20
    20
  ----
    0
  ----
```

1.19 Complements:

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements, one is r's complement and another is $(r-1)$'s complement, where r is base of the system. For binary system the base r is 2, therefore 2's complement and 1's complement is possible.

1.19.1 1's complement:

1's complement of a binary number is formed by simply changing each 1in the number to 0 and each 0 in the number to 1.

Example:

Binary Number		Equivalent 1's complement
1011	$\Longrightarrow$	0100
110001	$\Longrightarrow$	001110
100100	$\Longrightarrow$	011011
11001110	$\Longrightarrow$	00110001
1010110	$\Longrightarrow$	01010010

1.19.2 1's complement Subtraction:

(i) To subtract a smaller number from a larger number, the procedure is as follows:

1. Determine the 1's complement of the smaller number.
2. Add the 1's complement to the larger number
3. Remove the carry and add to the sum. The carry is called end-around carry.

The number of bits in the minuend and subtrahend must be equal.

Example:

1. Subtract $(01110)_2$ from $(10001)_2$ using 1's complement

Direct Method of binary subtraction	1's complement method of subtraction	Equivalent Decimal subtraction
10001 - 01110 00011	10001 + 10000 (1's complement of 10001) 100010(end-around carry1) → +1 (add carry to sum) 00011	17 - 14 3

2. Subtract $(101101)_2$ from $(110011)_2$using 1's complement

Direct Method of binary subtraction	1's complement method of subtraction	Equivalent Decimal subtraction
110011 - 101101 000110	110011 + 010010(1's complement of 101101**)** 1000101 (end-around carry1) ↳ +1 (add carry to sum) 000110	51 - 45 6

(ii) To subtract a larger number from a smaller binary number , the procedure is as follows

1. Determine the 1's complement of the larger number.
2. Add the 1's complement to the smaller number
3. The answer has an opposite sign and is the result. There is no carry

Example:

1.Subtract$(10010)_2$ from $(1100)_2$using 1's complement

Direct Method of binary subtraction	1's complement method of subtraction	Equivalent Decimal subtraction
01100 - 10010 - 00110	01100 + 01101(1's complement of 10010**)** 11001 put – sign and take 1's complement for the sum we get , -00110	51 - 45 -6

2. Subtract $(1101)_2$ from $(1010)_2$using 1's complement

Direct Method of binary subtraction	1's complement method of subtraction	Equivalent Decimal subtraction
1010 - 1101 - 0011	1010 + 0010(1's complement of 1101**)** 1100 put – sign and take 1's complement for the sum we get , -0011	10 - 13 - 3

1.19.2 2's Complement:

The 2's complement of a binary number is formed by taking 1's complement of the number and adding 1 to the least significant bit (LSB) position

$$\boxed{2's\ complement = 1's\ complement + 1}$$

Example:

Write 2's complement of a binary number $(1010)_2$

1's complement of 1010 = 0101
Add 1 = + 1
2's complement of 1010 = 0110

Some more Example:

Binary Number		1's complement	2's complement
1011	⟹	0100⟹	0101
110001	⟹	001110⟹	001111
100100	⟹	011011⟹	011100
11001110	⟹	00110001⟹	00110010
1010110	⟹	01010010⟹	01010011

1.20 Binary coded System:

In general, coding is the process of assigning a group of binary digits to represent multivalued items of information. Binary coded decimal (BCD) is a system of writing numerals that assigns a four-digit binary code to each digit 0 through 9 in a decimal (base-10) numeral. The four-bit BCD code for any particular single base-10 digit is its representation in binary notation, as follows:

0	1	2	3	4	5	6	7	8	9
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Numbers larger than 9, having two or more digits in the decimal system, are expressed digit by digit, For example, the BCD interpretation of the base-10 number 1895 is

1	*8*	*9*	*5*
0001	*1000*	*1001*	*0101*

ie., 1895 = *0001 1000 1001 0101*

The binary equivalents of 1, 8, 9, and 5, always in a four-digit format, go from left to right.

Binary codes are broadly classified into Numeric codes, Alphanumeric codes and Error detecting codes. Numeric codes are further classified into weighted codes and non-weighted codes. The most obvious way of encoding digits is "natural BCD" (NBCD), where each decimal digit is represented by its corresponding four-bit binary value. This is also called "8421" encoding.Standard binary coded decimal code is commonly known as a weighted 8421 BCD code, with 8, 4, 2 and 1 representing the weights of the different bits starting from the most significant bit (MSB) and proceeding towards the least significant bit (LSB). The weights of the individual positions of the bits of a BCD code are: $2^3 = 8$, $2^2 = 4$, $2^1 = 2$, $2^0 = 1$.

The main advantage of the Binary Coded Decimal system is that it is a fast and efficient system to convert the decimal numbers into binary numbers as compared to the pure binary system However, the disadvantage is that BCD code is inefficient as the states between 1010(decimal 10), and 1111 (decimal 15) are not used.

In non-weighted code, there is no positional weight i.e. each position within the binary number is not assigned a prefixed value. No specific weights are assigned to bit position in non –weighted code. The non-weighted codes are classified to

a) The Excess-3 codeb) The Gray code

1.21Excess-3 code:

Excess-3 code is an important BCD code, is a 4 bit code and used with BCD numbers as weights are not assigned, it is a kind of non- weighted codes. Excess-3 code was used on some older computers, cash registers and hand held portable electronic calculators.The Excess-3 code for a given decimal number is determined by adding '3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four bit binary equivalent. The table gives the Excess-3 code.

Decimal	8421	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

1.21.1 Decimal to Excess-3 code:

Excess-3 code of 24 is obtained as

2 4

+3 +3

5 7

0101 0111

Thus, Excess-3 code of 24 is 0101 0111.

Similarly, Excess-3 code for $(597)_{10}$ and $(14.57)_{10}$ is

$(597)_{10}$ = (100011001010)

$(14.57)_{10}$ = (01000111.10001010)

1.21.2 Excess-3 to Decimal:

From given Excess-3 code, the equivalent decimal number can be determined by first splitting the number into four-bit groups, starting from radix point and then subtracting 0011 from each four-bit group. This gives us 8421 BCD equivalent of the given Excess-3 code, which can then be converted into the equivalent decimal number.

Example:

Determine the decimal equivalent for the Excess-3 code 1000110.

$$\underbrace{0100}\,\underbrace{0110}$$

Then Subtracting 0011 from each group,

$$\begin{array}{rr} 0100 & 0110 \\ - \ 0011 & - \ 0011 \\ \hline 0001 & 0011 \end{array}$$

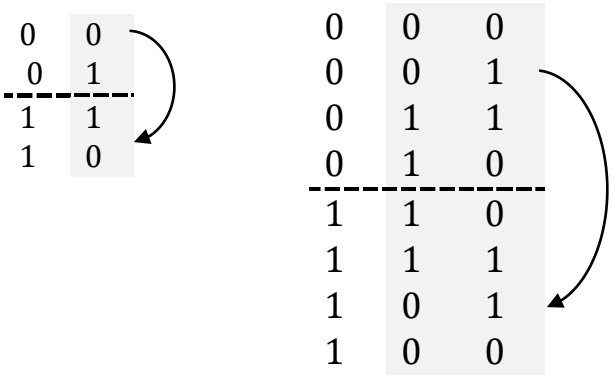
The new number as 00010011. Its decimal equivalent is 13.

1.22 Gray code:

The Gray code was designed by *Frank Gray* at Bell Labs in 1953. It belongs to a class of codes called the minimum change code. The successive coded characters never differ in more than one-bit.The Gray code is a non-weighted code. Because of this, the· gray code is not suitable for arithmetic operations but finds applications in input/output devices, some analog-to-digital converters and designation of rows and columns in Karnaugh map etc.

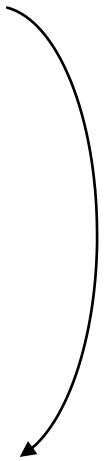
A three-bit gray code can be obtained by merely reflecting the two-bit code about an axis at the end of the code and assigning a third-bit as 0 above the axis and as 1 below the axis. The reflected gray code is nothing but code written in reverse order. By reflecting three-bit code, a four-bit code may be obtained.

Process of obtaining 3 bit Gray code by reflecting 2 bit Gray code:



Process of obtaining 4 bit Gray code by reflecting 3 bit Gray code:

Decimal	Binary	Gray Code
0	0000	0 0 0 0
1	0001	0 0 0 1
2	0010	0 0 1 1
3	0011	0 0 1 0
4	0100	0 1 1 0
5	0101	0 1 1 1
6	0110	0 1 0 1
7	0111	0 1 0 0
8	1000	1 1 0 0
9	1001	1 1 0 1
10	1010	1 1 1 1
11	1011	1 1 1 0
12	1100	1 0 1 0
13	1101	1 0 1 1
14	1110	1 0 0 1
15	1111	1 0 0 0



1.22.1 Decimal to Gray code conversion:

Example

1. Covert $(39)_{10}$ to gray code

The Gray code equivalent of decimal number 3 and 9 is $\underbrace{0010}_{3}$ $\underbrace{1101}_{9}$

The four-bit gray code for decimal number $(39)_{10} \Longrightarrow (00101101)_{Gray\ code}$.

Similarly, gray code for $(923.1)_{10}$ and (327) is

$(923.1)_{10}$ = (1101 0011 0010.0001) Gray code

$(327)_{10}$ = (100011 0100) Gray code

1.22.2 Binary to Gray code conversion:

- The most significant bit (MSB) in the Gray code is same as the corresponding bit in the binary number
- Going from left to right, add each adjacent pair of binary bit to get the next Gray code bit and discard the carry.

Example:

Convert $(1011)_2$ to Gray code

Step 1	1 0 1 1	Binary
	↓	
	1	Gray
Step 2	1 + 0 1 1	Binary
	1 1	Gray
Step 3	1 0 + 1 1	Binary
	1 1 1	Gray
Step 4	1 0 1 + 1	Binary
	1 1 1 0	Gray

$$(1011)_2 = (1110)_{Gray}$$

1.22.3 Grayto Binarycode conversion:

- The most significant bit (MSB) in the Binary code is same as the corresponding bit in the Gray code
- Going from left to right, add each binary bit generated to the gray digit in the next adjacent position and discard the carry.

Example:

Convert $(1110)_{Gray}$ to Binary code

Step 1	1 110	Gray
	↓	
	1	Binary
	+	
Step 2	1 110	Gray
	↓	Binary
	1 0	
Step 3	+	Gray
	1 11 0	
	↓	Binary
	1 0 1	
Step 4	+	Gray
	1 1 1 0	
	1 0 1 1	Binary

$$(1110)_{Gray} = (1011)_2$$

1.23 Alphanumeric code:

Alphanumeric codes are also called character codes due to their certain properties. These codes are basically binary. These codes are used to write alphanumeric data, including data, letters of the alphabet, numbers, mathematical symbols and punctuation marks which can be easily understandable and can be processed by the computers. Input output devices such as keyboards, monitors, mouse can be interfaced using these codes. A complete alphanumeric code would include the 26 lowercase letters , 26 uppercase letters, 10 numeric digits, 7 punctuation marks and anywhere from 20 to 40 other characters such as +, /, * , # and so on. That is it represents all of the various characters and functions that are found on a standard typewriter or computer keyboard. The most common alphanumeric codes used are ASCII code, EBCDIC code and Unicode.

1.23.1 ASCII code

The full form of ASCII code is American Standard Code for Information Interchange. It is a seven bit code based on the English alphabet. In 1967 this code was first published and since then it is being modified and updated. ASCII code has 128 characters some of which are enlisted below to get familiar with the code.

Dec	Octal	Hex	Binary	Symbol	Description
1	001	01	00000001	SOH	Start of Heading
2	002	02	00000010	STX	Start of text
3	003	03	00000011	ETX	End of text
4	004	04	00000100	EOT	End of transmission
5	005	05	00000101	ENQ	Enquiry
6	006	06	00000110	ACK	Acknowledgement
7	007	07	00000111	BEL	Bell
8	010	08	00001000	BS	Back Space
9	011	09	00001001	HT	Horizontal Tab
10	012	0A	00001010	LF	Line Feed
11	013	0B	00001011	VT	Vertical Tab
12	014	0C	00001100	FF	Form Feed
13	015	0D	00001101	CR	Carriage Return
14	016	0E	00001110	SO	Shift Out/X-On
15	017	0F	00001111	SI	Shift In/X-O

Example:

The following is a message encoded in ASCII code. What is the message?

01001000 1000101 1001100 1010000

Solution:

Convert each 7 bit code to its Hexadecimal equivalent

$$\begin{array}{cccccccc} 0100 & 1000 & & 0100 & 0101 & & 0100 & 1100 & & 0101 & 0000 \\ 4 & 8 & & 4 & 5 & & 4 & C & & 5 & 0 \end{array}$$

The result are 48 45 4C 50

Locate these hexadecimal values in table ASCII and determine the character represented by each. The result are: H E L P

1.23.2 EBCDIC code:

The EBCDIC stands for Extended Binary Coded Decimal Interchange Code. IBM invented this code to extend the Binary Coded Decimal which existed at that time. All the IBM computers and peripherals use this code. It is an 8 bit code and therefore can accommodate 256 characters. Below is given some characters of EBCDIC code to get familiar with it.

Char	EBCDIC	HEX	Char	EBCDIC	HEX	Char	EBCDIC	HEX
A	1100 0001	C1	P	1101 0111	D7	4	1111 0100	F4
B	1100 0010	C2	Q	1101 1000	D8	5	1111 0101	F5
C	1100 0011	C3	R	1101 1001	D9	6	1111 0110	F6
D	1100 0100	C4	S	1110 0010	E2	7	1111 0111	F7
E	1100 0101	C5	T	1110 0011	E3	8	1111 1000	F8
F	1100 0110	C6	U	1110 0100	E4	9	1111 1001	F9
G	1100 0111	C7	V	1110 0101	E5	blank	...	...
H	1100 1000	C8	W	1110 0110	E6	.	...	...
I	1100 1001	C9	X	1110 0111	E7	(	...	...
J	1101 0001	D1	Y	1110 1000	E8	+	...	...
K	1101 0010	D2	Z	1110 1001	E9	$	...	...
L	1101 0011	D3	0	1111 0000	F0	*	...	...
M	1101 0100	D4	1	1111 0001	F1	)	...	...
N	1101 0101	D5	2	1111 0010	F2	-	...	...
O	1101 0110	D6	3	1111 0011	F3	/		

1.24.3 Unicode

Unicode is the newest concept in digital coding. In Unicode every number has a unique character. Leading technological giants have adopted this code for its uniqueness.

Unit II Boolean algebra

Boolean operation-rules and laws of Boolean algebra-Demorgan's theorem-implications of expression using Boolean algebra-Karnaugh map

2.1 Boolean algebra:

Boolean algebra or switching algebra is a system of mathematical logic to perform different mathematical operations in binary system. Boolean algebra which was formulated by George Boole, an English mathematician (1815-1864) described propositions whose outcome would be either *true or false*. There 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 English 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
$0.0 = 0$	$0 + 0 = 0$	$\bar{1} = 0$
$0.1 = 0$	$0 + 1 = 0$	$\bar{0} = 1$
$1.0 = 0$	$1 + 0 = 0$	
$1.1 = 1$	$1 + 1 = 1$	

2.2 Some Basic laws for Boolean Algebra:

2.2.1 Boolean Postulates:

1. $A \,.\, 0 = 0 \; whereAcanbeeither\; 0\; or\; 1.$
2. $A \,.\, 1 = AwhereAcanbeeither\; 0\; or\; 1.$
3. $A \,.\, A = AwhereAcanbeeither\; 0\; or\; 1.$
4. $A \,.\, \bar{A} = 0\; whereAcanbeeither\; 0\; or\; 1.$
5. $\bar{\bar{A}} = AwhereAcanbeeither\; 0\; or\; 1.$
6. $A + 0 = AwhereAcanbeeither\; 0\; or\; 1.$
7. $A + 1 = 1\; whereAcanbeeither\; 0\; or\; 1.$
8. $A + \bar{A} = 1$

9. $A + A = A$

10. $A + B = B + A$ $where\ A\ and\ Bcan\ be\ either\ 0\ or\ 1.$

11. $A \,.\, B = B \,.\, A$ $where\ A\ and\ B\ can\ be\ either\ 0\ or\ 1.$

12. $\bar{0} = 1, \bar{1} = 0;$ $if A = 1\ then\ \bar{A} = 0\ and if A = 0\ then\ \bar{A} = 1$

2.2.2 Boolean Laws:

1. Commutative Law: According to Commutative Law, the order of OR operations and AND operations conducted on the variables makes no differences.

(a) $A + B = B + A$

(b) $AB = BA$

2. Associate Law: This law is for several variables, where the OR operation of the variables result is same though the grouping of the variables different. This law is quite same in case of AND operators.

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

(b) $(AB)\,C = A\,(BC)$

3. Distributive Law: This law is composed of two operators, AND and OR.

(a) $A\,(B + C) = AB + AC$

(b) $A + (BC) = (A + B)\,(A + C)$

4. Identity Law

(a) $A + A = A$

(b) $AA = A$

5. Redundance Law

a) $A + AB = A$

b) $A\,(A + B) = A$

c) $AB + A\bar{B} = A$

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

e) $A + \bar{A}B = A + B$

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

6. De Morgan's Theorem

a) $(\overline{A + B}) = \bar{A}\bar{B}$

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

The laws of Boolean algebra are also true for more than two variables.

Let us show one use of this law to prove the expression $A + B.C = (A + B).(A + C)$

Proof:

$A + BC = A.1 + B.C \quad (since, A.1 = A)$

$= A.(1 + B) + B.C \quad (since, B + 1 = 1)$

$= A.(1 + C) + AB + BC \quad (since, A.A = A.1 = A)$

$= A.1 + AB + BC$

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

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

Proof of De Morgan's Theorem by truth table,

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

Inputs		Outputs					
A	B	$\bar{A}$	$\bar{B}$	$\overline{A + B}$	$\bar{A} \cdot \bar{B}$	$\overline{A \cdot B}$	$\bar{A} + \bar{B}$
0	0	1	1	1	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	0	0	0

$Column\ for\ \overline{A + B}\ and\ \bar{A}\bar{B}\ are\ same$

$Column\ for\ \overline{AB}\ and\ \bar{A} + \bar{B}\ are\ same$

2.3 Examples of Boolean Algebra:

1. $Simplify, \overline{(A + \bar{B})(C + \bar{D})}$

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

$$= \bar{A} \cdot \bar{\bar{B}} + \bar{C} \cdot \bar{\bar{D}}$$

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

There is another method of simplifying complex Boolean expression. In this method there are only three simple steps.

i. Complement entire Boolean expression.
ii. Change all ORs to ANDs and all ANDs to ORs.
iii. Complement each of the variables and get final expression.

By this method,

Step 1: $\overline{(A+\bar{B})(C+\bar{D})}$will be first complemented, we get $(A+\bar{B})(C+\bar{D})$

Step 2: Change all (+) to (.) and (.) to (+) , we get $A\bar{B}+C\bar{D}$

Step 3: complement each of the variable, we get $\bar{A}B+\overline{CD}$

The final simplified form of Boolean expression$\overline{(A+\bar{B})(C+\bar{D})}$is got at the third step.

And it is exactly equal to the results which have been got by applying De Morgan Theorem.

2. $Simplify, \overline{\overline{AB}+\bar{A}+AB}$

$\overline{\overline{AB}+\bar{A}+AB)} = \overline{\overline{AB}}.\bar{\bar{A}}.\overline{AB}$

$= AB.A.\overline{AB}$

$= 0$

By Second Method,

$\overline{\overline{AB}+\bar{A}+AB} = \overline{AB}+\bar{A}+AB = \overline{A+B}.\bar{A}.A+B = \overline{\bar{A}+\bar{B}}.A.\bar{A}+\bar{B} = 0$

3. $Simplify, AB+A\bar{B}C+B\bar{C}$

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

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

$= AB+AC+B\bar{C}\ Since\ B+\bar{B}=1$

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

$= ABC+AB\bar{C}+AC+B\bar{C}$

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

$= AC+B\bar{C}$

4. $Simplify, C + \overline{BC}$:

Expression	Rule(s) Used
$C + \overline{BC}$	Original Expression
$= C + (\bar{B} + \bar{C})$	DeMorgan's Law.
$= (C + \bar{C}) + \bar{B}$	Commutative, Associative Laws.
$= 1 + B$	Complement Law.
$= 1$	Identity Law.

5. $Simplify\ \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

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

$= \bar{A}\bar{C} + A\bar{C}$ Since $\bar{B} + B$ =1

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

$= \bar{C}$

6. $Simplify, (A + C)(AD + A\overline{D}) + AC + C$:

Expression	Rule(s) Used
$(A + C)(AD + A\overline{D}) + AC + C$	Original Expression
$= (A + C)A(D + \overline{D}) + AC + C$	Distributive.
$= (A + C)A + AC + C$	Complement, Identity.
$= A((A + C) + C) + C$	Commutative, Distributive.
$= A(A + C) + C$	Associative, Idempotent.
$= AA + AC + C$	Distributive.
$= A + (A + 1)C$	Idempotent, Identity, Distributive.
$= A + C$	Identity, twice.

7. $Simplify$: $\bar{A}(A + B) + (B + AA)(A + \bar{B})$:

Expression	Rule(s) Used
$\bar{A}(A + B) + (B + AA)(A + \bar{B})$	Original Expression
$= \bar{A}A + \bar{A}B + (B + A)A + (B + A)\bar{B}$	Idempotent (AA to A), then Distributive, used twice.
$= \bar{A}B + (B + A)A + (B + A)\bar{B}$	Complement, then Identity. (Strictly speaking, we also used the Commutative Law for each of these applications.)
$= \bar{A}B + BA + AA + B\bar{B} + A\bar{B}$	Distributive, two places.
$= \bar{A}B + BA + A + A\bar{B}$	Idempotent (for the A's), then Complement and Identity to remove BB.
$= \bar{A}B + AB + A1 + A\bar{B}$	Commutative, Identity; setting up for the next step.
$= \bar{A}B + A(B + 1 + \bar{B})$	Distributive.
$= \bar{A}B + A$	Identity, twice (depending how you count it).
$= A + \bar{A}B$	Commutative.
$= (A + \bar{A})(A + B)$	Distributive.
$= A + B$	Complement, Identity.

8. $Simplify$, $\overline{AB}(\bar{A} + B)(\bar{B} + B)$:

Expression	Rule(s) Used
$\overline{AB}(\bar{A} + B)(\bar{B} + B)$	Original Expression
$= \overline{AB}(\bar{A} + B)$	Complement law, Identity law.
$= (\bar{A} + \bar{B})(\overline{A} + B)$	DeMorgan's Law
$= \bar{A} + \bar{B}B$	Distributive law.
$= \bar{A}$	Complement, Identity.

9. Prove that $(A+B)(A\bar{C}+C)(\bar{B}+AC) = A\bar{B}C + AC$

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

$= A\bar{B}\bar{C} + A\bar{C}.AC + A\bar{B}C + .AC + AB\bar{C}.\bar{B} + AB\bar{C}.AC + BC.\bar{B} + BC.AC$

$= A\bar{B}\bar{C} + A\bar{B}C + AC + ABC$

$= A\bar{B}\bar{C} + AC(\bar{B}+1) + ABC$

$= A\bar{B}\bar{C} + AC + ABC$

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

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

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

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

$= AC + A\bar{B}$

2.4 Representation of Boolean function in truth table:

Boolean algebra deals with binary variables and logic operation. A Boolean function, which is described by an algebraic expression called Boolean expression and which consists of binary variables, the constants 0 and 1, and the logic operation symbols. Consider the following example.

$$\underbrace{F(A,B,C,D)}_{BooleanFunction} = \underbrace{A + \overline{BC} + ADC}_{BooleanExpression} \qquad Equation\ 1$$

The left side of the equation represents the output Y. So we can state equation 1 as

$$Y = A + \overline{BC} + ADC$$

Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result.It is possible to convert the switching equation into a truth table. For example, consider the following switching equation.

$$F(A,B,C) = A + BC$$

The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is given by Table (2.1). The number of rows in the truth table is $2^n$where n is the number of input variables ($n = 3$ for the given equation). Hence there are $2^3 = 8$ possible input combinations of inputs.

Table 2.1: Truth table for $F = A + BC$

Inputs			Output	
A	B	C	BC	$F = A + BC$
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

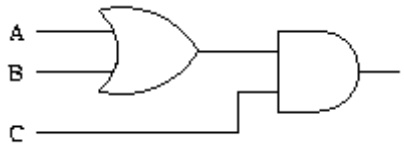
Represent the Boolean function $F(A, B, C) = \bar{A}B + \bar{B}C$as truth table in table 2.2.

Table 2.2: Truth table for $F = \bar{A}B + \bar{B}C$

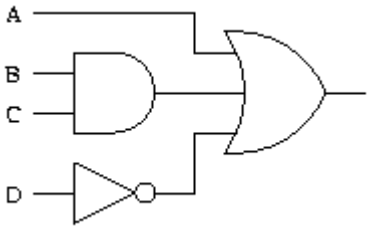
A	B	C	$\bar{A}$	$\bar{B}$	$\bar{A}B$	$\bar{B}C$	$\bar{A}B + \bar{B}C$
0	0	0	1	1	0	0	0
0	0	0	1	1	0	1	1
0	1	0	1	0	1	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0

2.5 Logic gates using Boolean Expressions:

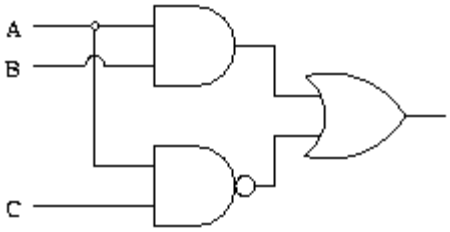
1. Draw a logic circuit for $(A + B)C$.



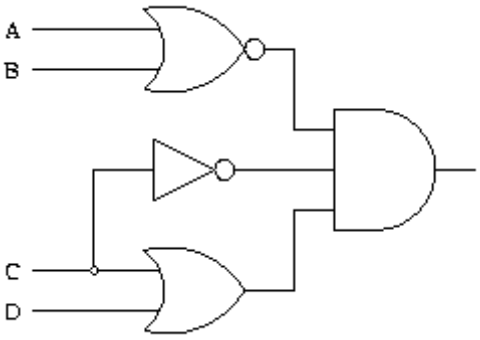
2. Draw a logic circuit for $A + BC + \bar{D}$.



3. Draw a logic circuit for $AB + \overline{AC}$.



4. Draw a logic circuit for $(\overline{A + B})\,(C + D)\,\bar{C}$.



2.6 Minterms:

In general, the unique algebraic expression for any Boolean function can be obtained from its truth table by using an OR operator to combined all minterms for which the function is equal to 1.

A minterm, denoted as m_i, where $0 \leq i < 2^n$, is a product (AND) of the n variables in which each variable is complemented if the value assigned to it is 0, and uncomplemented if it is 1.

1-minterms = minterms for which the function F = 1.

0-minterms = minterms for which the function F = 0.

Any Boolean function can be expressed as a sum (OR) of its 1- minterms.

A shorthand notation:

$$F(list\ of\ variables) = \sum(list\ of\ 1-minterm\ indices)$$

Example :

x	y	z	$Minterms$	F	F'
0	0	0	$m_0 = x'\ y'\ z'$	0	1
0	0	1	$m_1 = x'\ y'\ z$	0	1
0	1	0	$m_2 = x'\ y\ z'$	0	1
0	1	1	$m_3 = x'\ y\ z$	1	0
1	0	0	$m_4 = x\ y'\ z'$	0	1
1	0	1	$m_5 = x\ y'\ z$	1	0
1	1	0	$m_6 = x\ y\ z'$	1	0
1	1	1	$m_7 = x\ y\ z$	1	0

$$F = x'\ y\ z + x\ y'\ z + x\ y\ z' + x\ y\ z$$
$$= m_3 + m_5 + m_6 + m_7$$

or

$$F\ (x, y, z) = \Sigma(3, 5, 6, 7)$$

The inverse of the function can be expressed as a sum (OR) of its 0- minterms.

A shorthand notation:

$$F'(list\ of\ variables)\ =\ \Sigma(list\ of\ 0-minterm\ indices)$$

Example:

$$F'\ =\ x'\,y'\,z'\ +\ x'\,y'\,z\ +\ x'\,y\,z'\ +\ x\,y'\,z'$$
$$=\ m_0\ +\ m_1\ +\ m_2\ +\ m_4$$

Or

$$F'\,(x,y,z)\ =\ \Sigma(0,1,2,4)$$

Problem:

1. $Express\ the\ Boolean\ function\ F\ =\ x\ +\ y\,z\ as\ a\ sum\ of\ minterms.$

Solution:

This function has three variables: $x, y, and\ z$. Therefore All terms must have these three variables. Thus, we need to expand the first term by ANDing it with $(y\ +\ y')(z\ +\ z')$, and we expand the second term with $(x\ +\ x')$ to get

$$F\ =\ x\ +\ y\,z\ =\ x\,(y\ +\ y')\,(z\ +\ z')\ +\ (x\ +\ x')\,y\,z$$

$$=\ x\,y\,z\ +\ x\,y\,z'\ +\ x\,y'\,z\ +\ x\,y'\,z'\ +\ x\,y\,z\ +\ x'\,y\,z$$

$$=\ x'\,y\,z\ +\ x\,y'\,z'\ +\ x\,y'\,z\ +\ x\,y\,z'\ +\ x\,y\,z$$

$$=\ m3\ +\ m4\ +\ m5\ +\ m6\ +\ m7$$

$$=\ \Sigma(3,4,5,6,7)$$

x	y	z	$Minterms$	F
0	0	0	$m_0 = x'\,y'\,z'$	0
0	0	1	$m_1 = x'\,y'\,z$	0
0	1	0	$m_2 = x'\,y\,z'$	0
0	1	1	$m_3 = x'\,y\,z$	1
1	0	0	$m_4 = x\,y'\,z'$	1
1	0	1	$m_5 = x\,y'\,z$	1
1	1	0	$m_6 = x\,y\,z'$	1
1	1	1	$m_7 = x\,y\,z$	1

2. Suppose a function F is defined by the following truth table, then convert it into the sum of minterms

Solution:

Given Truth Table

A	B	C	m.t	F
0	0	0	m_0	0
0	0	1	m_1	1
0	1	0	m_2	1
0	1	1	m_3	0
1	0	0	m_4	1
1	0	1	m_5	0
1	1	0	m_6	0
1	1	1	m_7	1

Since $F = 1$ on rows 1, 2, 4, and 7, we obtain

$F = m_1 + m_2 + m_4 + m_7$

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

A compact notation is to write only the numbers of the minterms included in F, using the Greek letter capital sigma to indicate a sum:

$F = \sum(1,2,4,7)$

This form can be written down immediately by inspection of the truth table.

2.7 Maxterms:

A maxterm, denoted as $M_i, where\ 0 \leq i < 2^n$, is a sum (OR) of the $n$variables (literals) in which each variable is complemented if thevalue assigned to it is 1, and uncomplemented if it is 0.

1-maxterms = maxterms for which the function *F* = 1.

0-maxterms= maxterms for which the function *F* = 0.

Any Boolean function can be expressed as a product (AND) of its 0-maxterms.

A shorthand notation:

$$F(list\ of\ variables) = \prod(list\ of\ 0 - maxterm\ indices)$$

x	y	z	$Maxterms$	F	F'
0	0	0	$M_0 = x + y + z$	0	1
0	0	1	$M_1 = x + y + z'$	0	1
0	1	0	$M_2 = x + y' + z$	0	1
0	1	1	$M_3 = x + y' + z'$	1	0
1	0	0	$M_4 = x' + y + z$	0	1
1	0	1	$M_5 = x' + y + z'$	1	0
1	1	0	$M_6 = x' + y' + z$	1	0
1	1	1	$M_7 = x' + y' + z'$	1	0

Example:

$$F = (x + y + z) \cdot (x + y + z') \cdot (x + y' + z) \cdot (x' + y + z)$$
$$= M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

or

$$F(x, y, z) = \prod(0, 1, 2, 4)$$

The inverse of the function can be expressed as a product (AND) of its 1-maxterms.

A shorthand notation:

$$F(list\ of\ variables) = \prod(list\ of\ 1 - maxterm\ indices)$$

Example:

$$F' = (x + y' + z') \cdot (x' + y + z') \cdot (x' + y' + z) \cdot (x' + y' + z')$$
$$= M_3 \cdot M_5 \cdot M_6 \cdot M_7$$

or

$$F'(x, y, z) = \prod(3, 5, 6, 7)$$

Problem:

1. Express the Boolean function $F = xy + x'z$ in a product of maxterm form

Solution:

Convert the function into OR terms using the distributive law.

$F = xy + x'z$

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

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

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

The function has three variables $x, y \; and \; z$. In each OR term one variable is missing

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

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

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

Combining all terms and remove the terms that appear more than once, we get

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

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

$F = M_0 M_2 M_4 M_5$

$F(x, y, z) = \prod(0,2,4,5)$

2. Find the product of maxterms if $F(A, B, C) = \sum(1,4,5,6)$

Solution:

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

the complement is $F'(A, B, C) = \sum(0,2,3) = m_0 + m_2 + m_3$

Applying DeMorgan's theorem

$\bar{F}'(A, B, C) = \overline{m_0 + m_2 + m_3}$

$= \overline{m_0} \cdot \overline{m_2} \cdot \overline{m_3}$

$= M_0 \cdot M_2 \cdot M_3$

$F(A, B, C) = \prod(0,2,3)$

2.8 Canonical form :

Definition:

Any Boolean function that is expressed as a sum of minterms or as a product of maxterms is said to be in its canonical form.

To convert from one canonical form to its other equivalentform, interchange the symbols $\sum and \prod$, and list the index numbers that were excluded from the original form.

To convert from one canonical form to its dual, interchange the symbols Σ and Π, and list the index numbers from the original form, or use De Morgan's Law or the duality principle.

Example:

$$F = m_3 + m_5 + m_6 + m_7 = \Sigma(3,5,6,7) \quad \sum 1 - minterms$$
$$= x'yz + xy'z + xyz' + xyz$$

Equivalent

$$= M_0 \cdot M_1 \cdot M_2 \cdot M_4 = \prod(0,1,2,4) \quad \prod 0 - maxterms$$
$$= (x+y+z)\cdot(x+y+z')\cdot(x+y'+z)\cdot(x'+y+z)$$

Duals

Inverse

$$F' = m_0 + m_1 + m_2 + m_4 = \Sigma(0,1,2,4) \quad \sum 0 - minterms$$
$$= x'y'z' + x'y'z + x'yz' + xy'z'$$

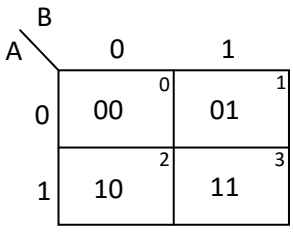
Equivalent

$$= M_3 \cdot M_5 \cdot M_6 \cdot M_7 = \prod(3,5,6,7) \quad \prod 0 - maxterms$$
$$= (x+y'+z')\cdot(x'+y+z')\cdot(x'+y'+z)\cdot(x'+y'+z')$$

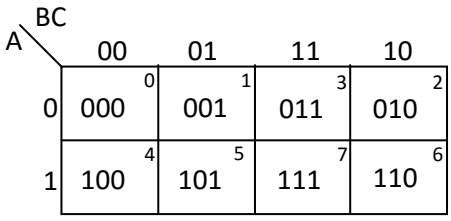
2.9 Karnaugh-map or K-map:

The Boolean theorems and the De-Morgan's theorems are useful in manipulating the logic expression. We can realize the logical expression using gates. The number of logic gates required for the realization of a logical expression should be reduced to a minimum possible value. One of the methods used to minimize the logical expression is K-map method. A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The Karnaugh map can also be described as a special arrangement of a truth table.

The K-map is a graphical device used to simplify a logical equation or to convert a truth table to its corresponding logic circuit in a simple, logical method. It is also known as Veitch diagram. A K-map is a diagram made up of squares and may be considered to be the graphic representation of the minterm canonical form. Each minterm is represented by a cell, and the cells are assembled in an orderly arrangement such that adjacent cell represent minterms which differ by one variable. The number of cells in a K-map depends upon the number of variables in the Boolean expression. Two variables map contain four cells, three variables map contain eight cells and n variables map contain 2^n cells. Each row and column of the map is assigned by 0's and 1's as shown in figure.



Two Variables K-map
With cell number



Three Variables K-map
with cell number

This method can be done in two different ways, as discussed below.

2.9.1 Sum of Products (SOP) Form

It is in the form of sum of three terms AB, AC, BC with each individual term is a product of two variables. Say A.B or A.C or B.C. Therefore such expressions are known as expression in SOP form. The sum and products in SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions. In SOP form, 0 represents a bar and 1 represents an unbar. SOP form is represented by $\sum$.

Boolean expression in SOP may or may not be in a standard form. First the expression is converted into SOP and then, 1's are marked in each cell corresponding to the minterm of expression and the remaining cells are marked with 0's.

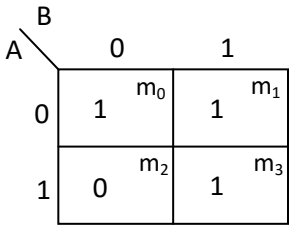
Examples of SOP:

1. K-map for the Boolean expression $Y(A,B,C) = \bar{A} + B$

In SOP form $\bar{A}\bar{B} + \bar{A}B + AB$

$\downarrow\downarrow \quad \downarrow\downarrow \quad \downarrow\downarrow$

$0\,0 \quad 0\,1 \quad 1\,1$



$\sum m(0,1,3)$

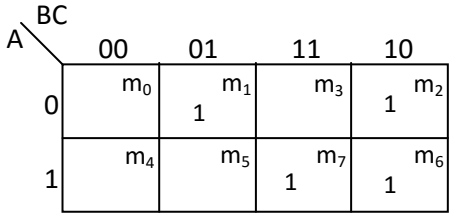
Result of $\bar{A}\bar{B} + \bar{A}B + AB$ is $\bar{A} + B$

2. K-map for the Boolean expression $Y(A,B,C) = AB + B\bar{C} + \bar{A}\bar{B}C$

In SOP form $ABC + AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C$

$\downarrow\downarrow\downarrow \quad \downarrow\downarrow\downarrow \quad \downarrow\downarrow\downarrow \quad \downarrow\downarrow\downarrow$

$1\,1\,1 \quad 1\,1\,0 \quad 0\,1\,0 \quad 0\,0\,1$



$\sum m(1,2,6,7)$

Result of $ABC + AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C$ is $AB + B\bar{C} + \bar{A}\bar{B}C$

2.9.2 Product of Sums (POS) Form

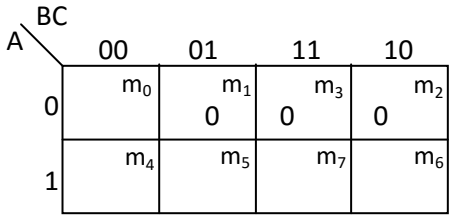
It is in the form of product of three terms (A+B), (B+C), or (A+C) with each term is in the form of a sum of two variables. Such expressions are said to be in the product of sums (POS) form. In POS form, 0 represents an unbar and 1 represents a bar. POS form is represented by $\prod$

Example of POS:

In POS form $(B+\bar{C})(\bar{A}+\bar{B})(\bar{B}+C)$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0$

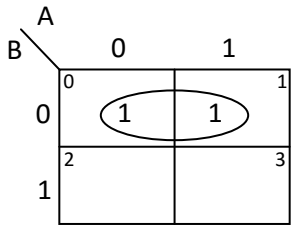


$\prod m(1,3,2)$

Result of $(B+\bar{C})(\bar{A}+\bar{B})(\bar{B}+C)$ is $(A+\bar{C})(A+\bar{B})$

2.10 Grouping the adjacent cells of Karnaugh map:

Adjacent cells*:*If two occupied cells of a Karnaugh are adjacent, horizontally or vertically (but not diagonally) then one variable is redundant. This has resulted by labeling the map as shown,



Consider the above map. The function plotted is $Y = f(AB) = \bar{A}\bar{B} + A\bar{B}$Using algebraic simplification, $Y = \bar{B}(\bar{A}+A) = \bar{B}$ by using the Boolean law$(A + \bar{A} = 1)$. Referring to the map we can encircle the adjacent cells and assume that $A\ and\ \bar{A}$ are not required.

i.e. adjacent cells satisfy the condition$A + \bar{A} = 1$.

The Karnaugh map uses the following rules for the simplification of expressions by grouping together adjacent cells containing ones

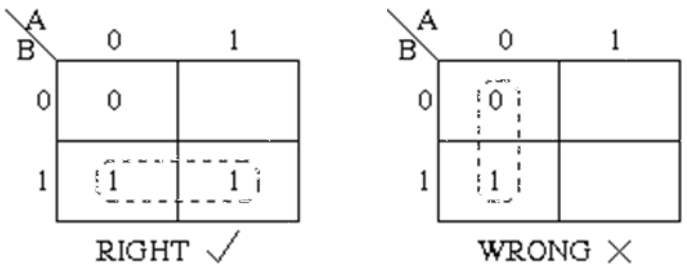
2.11 Rules for grouping cells in K-map:

1. Groups may not include any cell containing a zero
2. Groups may be horizontal or vertical, but not diagonal.
3. Groups must contain 1, 2, 4, 8, or in general 2^n cells. That is if n = 1, a group will contain two 1's since $2^1 = 2$. If n = 2, a group will contain four 1's since $2^2 = 4$.
4. Each group should be as large as possible.
5. Each cell containing a one must be in at least one group.
6. Groups may overlap.
7. Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
8. There should be as few groups as possible, as long as this does not contradict any of the previous rules.

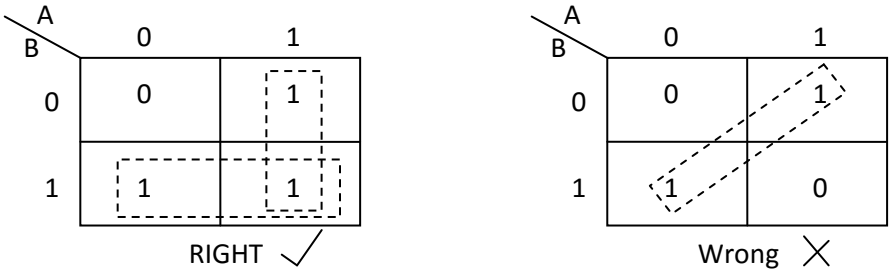
2.12 Examples for the above said rules:

In the following examples the grouping of cells in correct method is indicated as "RIGHT with a tick mark". For more understanding, the improper grouping is alsoindicated about the respective rule as "Wrong with a cross mark".

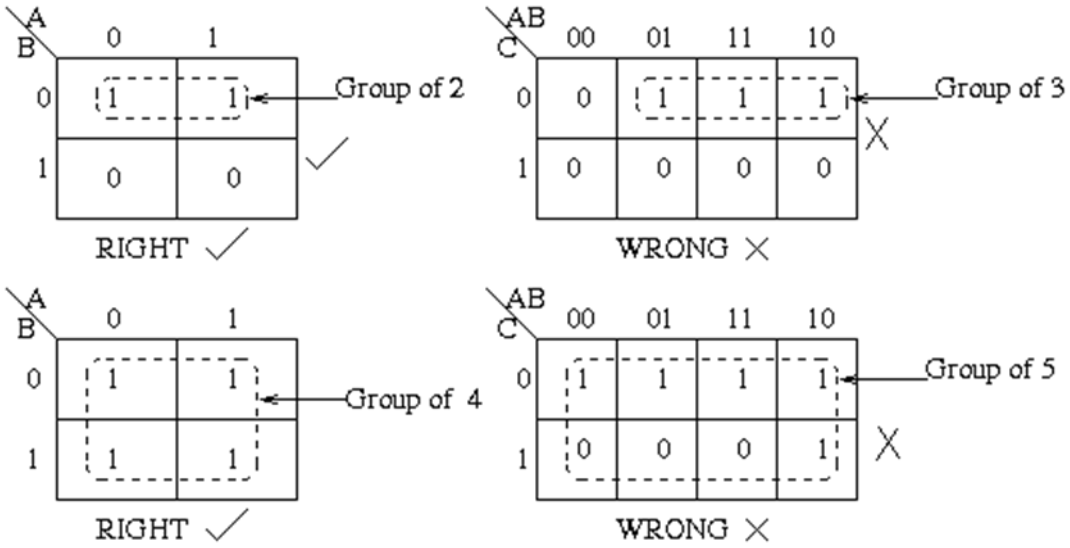
For Rule 1:



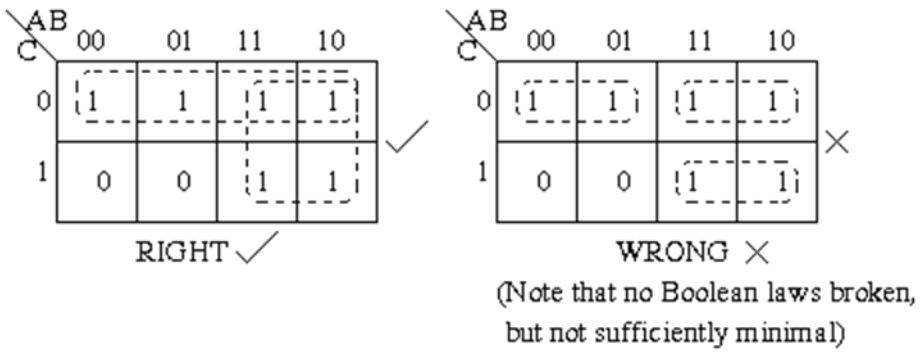
For Rule 2:



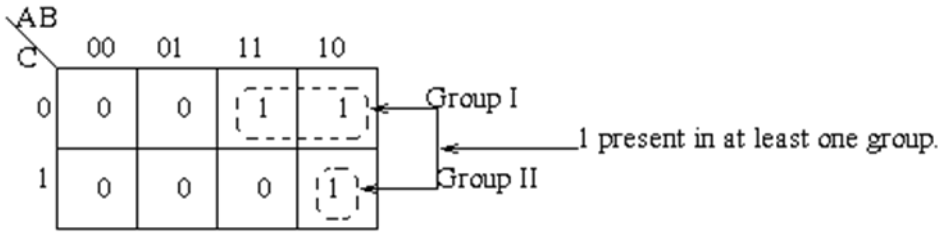
For Rule 3:



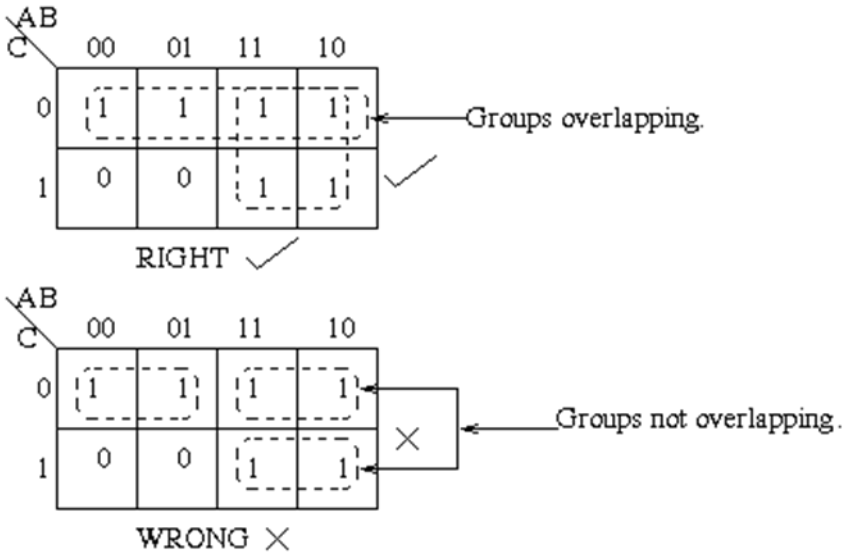
For Rule 4:



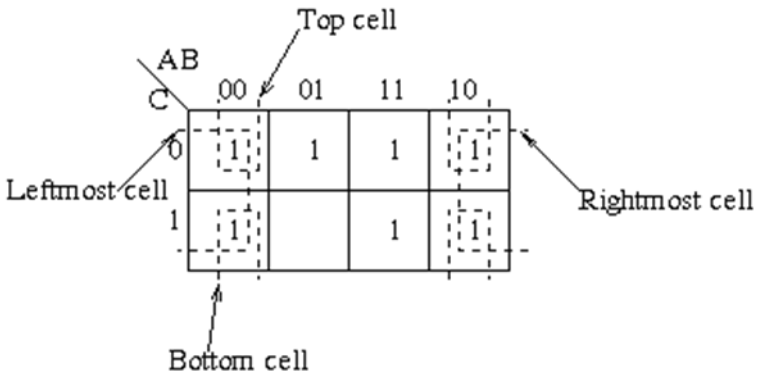
For Rule 5:



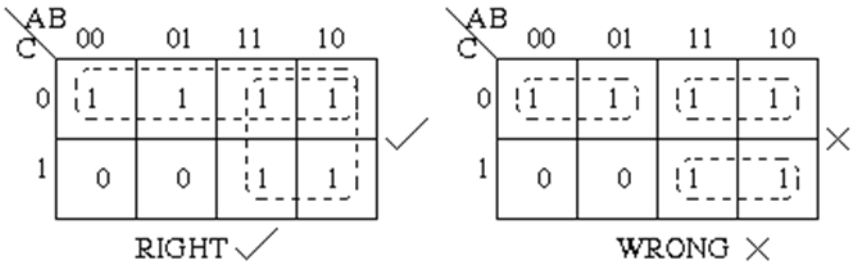
For Rule 6:



For Rule 7:



For Rule 8:



2.13Simplifying Boolean Expression using K Map

2.13.1 Minterm Solution of K Map:

The following are the steps to obtain simplified minterm solution using K-map.

Step 1: Initiate

Express the given expression in its canonical form

Step 2: Populate the K-map

Enter the value of 'one' for each product-term into the K-map cell, while filling others with zeros.

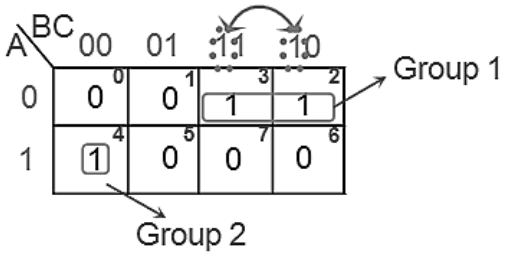
Step 3: Form Groups

Using the rules of grouping of cells form as many as possible larger groups

Step 4: Obtain Boolean Expression for Each Group

Express each group interms of input variables by looking at the common variables seenin cell-labeling.

For example in the figure shown below there are two groups with two and one number of 'ones' in them (Group 1 and Group 2, respectively). All the 'ones' in the Group 1 of the K-map are present in the row for which A = 0. Thus they contain the variable $\overline{A}$. Further these two 'ones' are present in adjacent columns which have only B term in common as indicated by the double headed arrow in the figure.



Hence the next term is B. This yields the product term corresponding to this group as $\overline{A}B$. Similarly the 'one' in the Group 2 of the K-map is present in the row for which A = 1. Further the variables corresponding to its column are $\overline{B}\,\overline{C}$. Thus one gets the overall product-term for this group as $A\overline{B}\overline{C}$.

Step 5: Obtain Boolean Expression for the Output

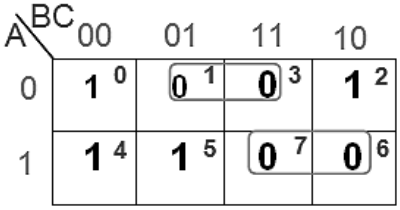
The product-terms obtained for individual groups are to be combined to form sum-of-product (SOP) form which yields the overall simplified Boolean expression. This means that for the K-map shown in Step 4, the overall simplified output expression is $\bar{A} + A\bar{B}\bar{C}$

2.13.2 Maxterm Solution of K Map

The method to be followed in order to obtain simplified maxterm solution using K-map is similar to that for minterm solution except minor changes listed below.

1. K-map cells are to be populated by 'zeros' for each sum-term of the expression instead of 'ones'.
2. Grouping is to be carried-on for 'zeros' and not for 'ones'.
3. Boolean expressions for each group are to be expressed as sum-terms and not as product-terms.
4. Sum-terms of all individual groups are to be combined to obtain the overall simplified Boolean expression in product-of-sums (POS) form.

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

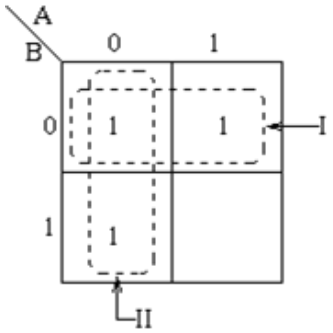


Simplified Expression is $Y = (A + \bar{C})(\bar{A} + \bar{B})$

Problems:

1. Draw Karnaugh map for the Boolean expression $Y = \bar{A}\bar{B} + A\bar{B} + \bar{A}B$:

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



Pairs of 1's are grouped as shown above, and the simplified answer is obtained by using the following steps: Two groups can be formed that the largest rectangular bands that can bemadeconsistoftwo1s.

The first group:

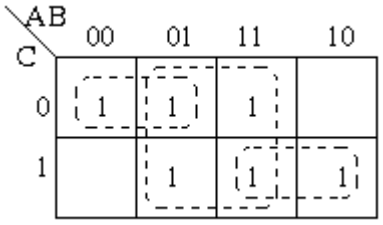
The first group labeledI, consists of two 1s which correspond to A = 0, B = 0 and A = 1, B = 0. (or) all squares that correspond to the area of the map where B = 0 contains 1s, independent of the value of A. So when B = 0 the output is 1. The expression of the output will contain the term $\bar{B}$

The second group:

The group labeledII corresponds to the area of the map where A = 0. The group can therefore be defined as$\bar{A}$. This implies that when A = 0 contains 1s, independent of the value of B.So when A = 0 the output is 1. The expression of the output is $\bar{A}$

Hence the simplified answer is $Y = \bar{A} + \bar{B}$

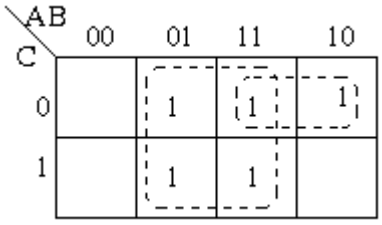
2. Draw K-map for the Expression $Y = \bar{A}\bar{B}\bar{C} + \bar{A}B + AB\bar{C} + AC$



$Y = B + AC + \bar{A}\bar{C}$

By using the rules of simplification and ringing of adjacent cells in order to make as many variables dismissed, the minimised result obtained is $Y = B + AC + \bar{A}\bar{C}$

3. Draw k-map for the expression$Y = \bar{A}B + B\bar{C} + BC + A\bar{B}\bar{C}$



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

By using the rules of simplification and ringing of adjacent cells in order to make as many variables redundant, the minimised result obtained is $Y = B + A\bar{C}$

4. Minimize the following expressions using K-map

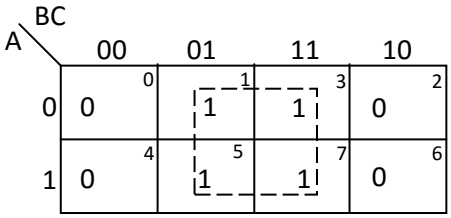
(i) $Y(A,B,C) = \sum m(1,3,5,7)$

(ii) $Y(A,B,C) = \sum m(0,1,4,5)$

(iii) $Y(A,B,C) = \sum m(0,2,4,6)$

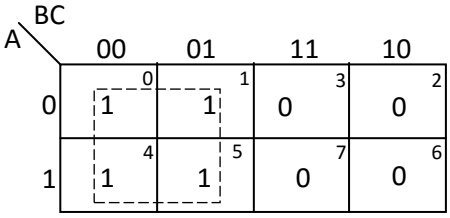
Solution:

(i) $Y(A,B,C) = \sum m(1,3,5,7)$



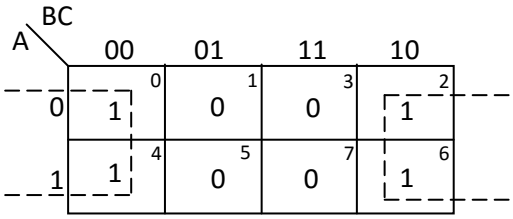
$Y(A,B,C) = C$

(ii) $Y(A,B,C) = \sum m(0,1,4,5)$



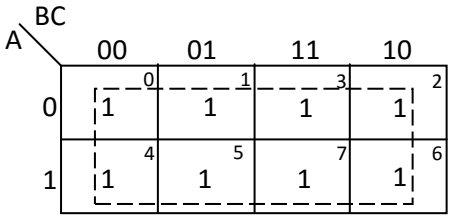
$Y(A,B,C) = \bar{B}$

(iii) $Y(A,B,C) = \sum m(0,2,4,6)$



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

5. Simplify the expression using K-map $Y(A,B,C) = \sum m(0,1,2,3,4,5,6,7)$



$Y(A,B,C) = 1$

6. Minimize the following expressions using K-map

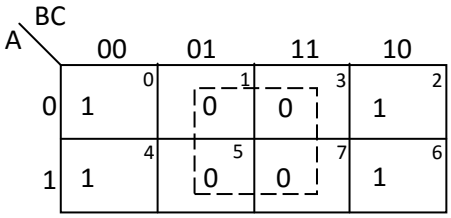
(i) $Y(A,B,C) = \prod M(1,3,5,7)$

(ii) $Y(A,B,C) = \prod M(0,1,4,5)$

(iii) $Y(A,B,C) = \prod M(0,2,4,6)$

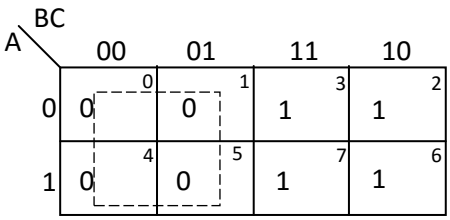
Solution:

(i) $Y(A,B,C) = \prod M(1,3,5,7)$



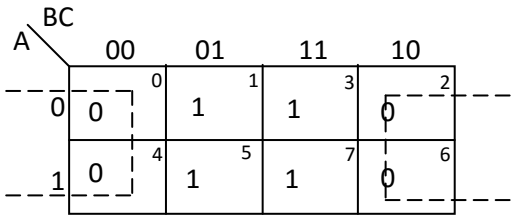
$Y(A,B,C) = \bar{C}$

(ii) $Y(A,B,C) = \prod M(1,3,5,7)$



$Y(A,B,C) = B$

(iii) $Y(A,B,C) = \prod M(0,2,4,6)$



$Y(A,B,C) = C$

Unit III Basic Logic Gates

AND, OR, NOT (symbol, truth table, circuit diagram, working) – NAND, NOR, EX-OR, EX-NOR(symbol, truth table)

3.1 Logic Gate:

A logic gate is an elementary building block from which all digital electronic circuits and microprocessor based systems are constructed.

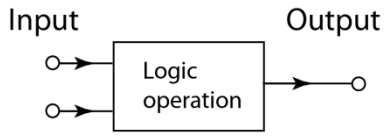


Figure 3.1 Block Diagram of Logic Gate

In digital logic design only two voltage levels or states are allowed and these states are generally referred to as Logic “1” and Logic “0”, High and Low, or True and False. These two states are represented in Boolean algebra and standard truth tables by the binary digits of “1” and “0” respectively. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels. A good example of a digital state is a simple light switch as it is either “ON” or “OFF” but not both at the same time

It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc. The relationship between the various digital states is given in table 3.1 as

Table 3.1: Digital States

Boolean Algebra	Boolean Logic	Voltage State
Logic “1”	True (T)	High (H)
Logic “0”	False (F)	Low (L)

The digital logicgates and digital logic systems use “Positive logic”, in which a logic level “0” or “LOW” is represented by a zero voltage, 0v or ground and a logic level “1” or “HIGH” is represented by a higher voltage such as +5 volts, with the switching from one voltage level to the other, from either a logic level “0” to “1” or “1” to “0” being made as quickly as possible to prevent any faulty operation of the logic circuit.

There also exists a complementary “Negative Logic” system in which the values and the rules of a logic “0” and a logic “1” are reversed.

Ideal Digital Logic Gate Voltage Levels is shown in figure 3.2

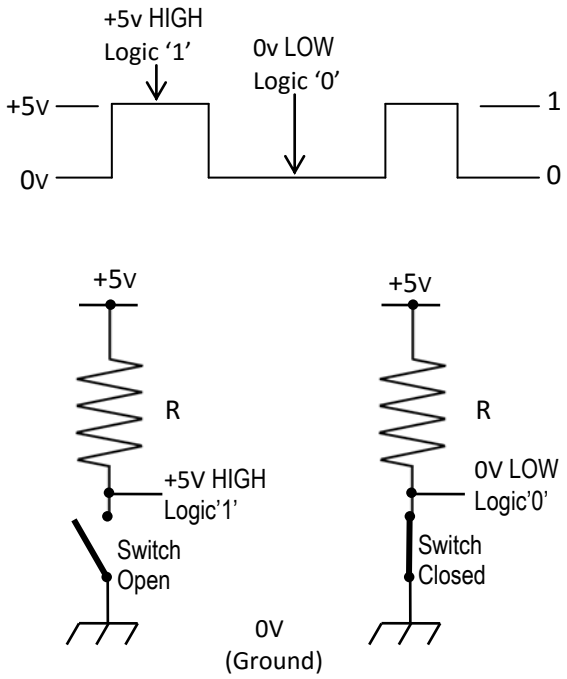


Figure 3.2: Digital Logic Gate Voltage Level

Where the opening or closing of the switch produces either a logic level “1” or a logic level “0” with the resistor R being known as a “pull-up” resistor.

3.2 OR gate:

OR gate performs logical OR(addition) operation which means outputs is logical 1 if at least one of the inputs is 1. An OR gate has two or any more numbers of inputs but only one output. Only if all of the inputs are only in low state or logical 0 the output is low or 0 and in all other inputs conditions the output will be high or logical 1.

The logical symbol of two input OR gate is

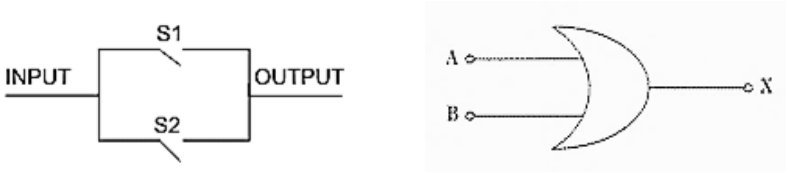


Figure 3.3: Logical Symbol of OR Gate

The logical expression is $X = A + B$

From above explanation the truth table of logical OR gate can be represented in table 3.2 as,

Table 3.2 Truth Table of OR Gate

Inputs		Output
A	B	$X = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

3.2.1 Diode OR Gate

A simple two inputs OR gate can be realized by using diode as shown in figure 3.4

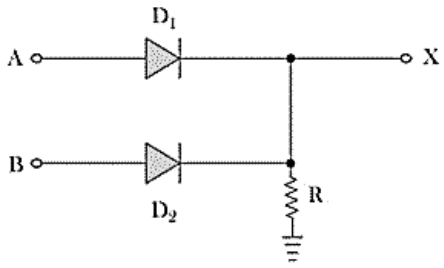


Figure 3.4: Diode OR Gate

In the above circuit (figure 3.4), if A and B are applied with 0V, there will be no voltage appears at X.

When any of the inputs is given with +5V, (figure 3.5) the respective diode becomes forward biased and behaves as ideally short circuited hence this +5 V will appear at output X. +5 V means logical 1. Actually entire 5V will not appear at X, around 0.6 to 0.7 V will drop across the diode as forward bias voltage and rest of the voltage i.e. 5 – 0.6 = +4.4 V or 5 – 0.7 = +4.3 V will appear at X. This 4.4 V or 4.3 V is practically considered as logical 1.

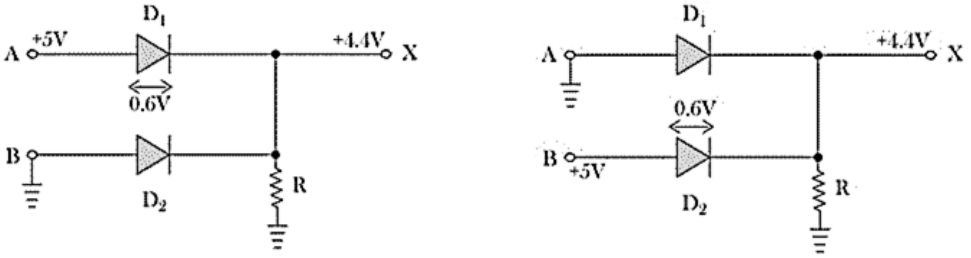


Figure 3.5: Diode OR Gate with any one input is +5v

If both of the inputs are given with +5 V, both diodes will be forward biased (figure 3.6). Hence, similarly 4.4 V will appear at X.

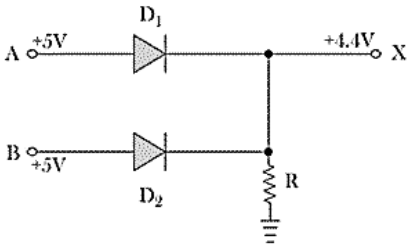


Figure 3.6: Diode OR Gate with both inputs are +5v

If both of the inputs A and B are grounded or given 0V, there will be no voltage appears at X and hence X is considered as logical 0.

3.2.2 Transistor OR Gate

The OR gate can also be realized by using transistor. In this case the OR gate is referred as transistor OR gate. Two inputs such OR gate is shown in figure 3.7

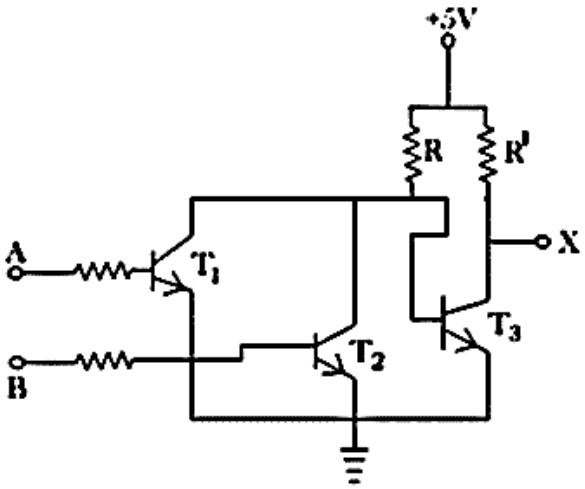


Figure 3.7: Transistor OR Gate

Now if A and B both are given with 0V, both of the transistor are in OFF condition, hence supply voltage + 5 V will not get path to the ground through either of the transistors, T_1 and T_2. As a result base of the transistor T_3 will get enough potential to make it ON. In that condition supply + 5 V will get path to the transistor T_3 is in ON condition it will behaves as ideally short circuited, hence entire supply voltage + 5 V will drop across resistor R' and X terminal (Node) will get 0V. In practice, transistor T_3 will not be ideal short circuited it will have some voltage drop across it which will be around 0.6 – 0.7 V. This voltage will appear at node X and this 0.6 or 0.7 volt is considered as logical 0. Now, if base terminal either of the transistors T_1 or T_2 or both are given with + 5 V, the respective transistor as both will be in ON condition. In that case supply voltage + 5 V will get path to ground through either of the transistors or both. As a result current starts flowing to the ground from supply through this path, and entire supply voltage will drop across resistor R. So, the base of transistor T_3 will not get sufficient potential to make the transistor T_3 ON. Hence entire supply voltage will appear at X and the X becomes at high logical state or logical 1.

3.3 AND Gate

AND Gate is a logical gate which is widely used having two or more inputs and a single output. This gate works or operates on logical multiplication rules. In AND gate if either of the inputs is low (0), then the output is also low, but if all the inputs are high (1) the output will also be high (1). An AND gate performs multiplication operation of binary digits 1 and 0. In multiplying 0 with 0 we will get 0, 1 with 0 or 0 with 1, we will get 0. We get 1 only when 1 is multiplied by 1.

In other words, an AND gate is a digital device which produces high output only when all inputs are high and produces low output at all other inputs conditions. High digital signal means logically 1 and low digital signal means logically 0. An AND gate may have any number of input probes but only one output probe.

A two input AND gate is logically represented in figure 3.8 as

Figure 3.8: Logical Symbol of AND Gate

Where A and B represent inputs and X represents the output of the gate. A, B and X either be 1 or 0 logically. The logical Expression of AND gate hence can be represented as $X = A \cdot B$All multiplication combination of A and B can be represented in tabular form (table 3.3) and is known as truth table.

Table 3.3 Truth Table of AND Gate

Inputs		Output
A	B	$X = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

3.3.1 Diode AND Gate

Normally an AND gate is designed by either diodes or transistors. While, diodes are used to design AND gate, it is called diode AND gate. The basic circuit of a diode AND gate is shown in figure 3.9

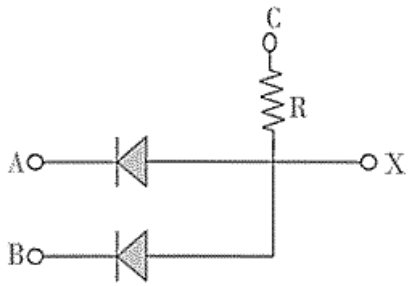


Figure 3.9: Diode AND Gate

In the above circuit (figure 3.9) we first apply +5V at C. Now if we apply +5V at A and B, both of the diodes are reversed biased (figure 3.10) and hence behave both diodes as OFF or open circuit. At this situation as both diodes are OFF, no current will flow through resistor R and voltage of C (+5V) will also appears at X. As the supply voltage +5V appears at X, the output of the circuit is considered as high or logical 1.

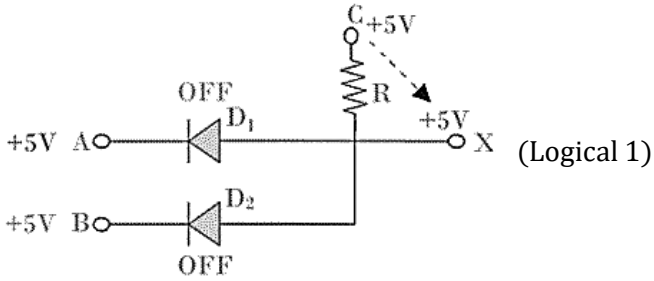


Figure 3.10: Diode AND Gate with both input +5v

Now, if either point A or B or both are applied with 0 Volt or they are grounded, respective diode will become forward biased (figure 3.11) and hence behaves as 'ON' or short circuited. At this condition, supply voltage +5V at point C will get path through either of diodes or both to the ground potential. As the current flowing from C to ground through resistor R, entire 5V will be dropped across the resistor and hence voltage at X will become low or logically zero.

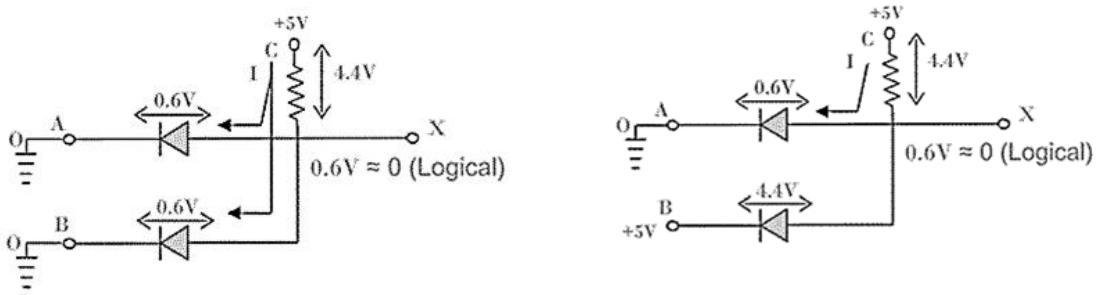


Figure 3.11: Diode AND Gate with any one input +5v

The diodes at forward biased condition do not behave as ideal short circuit; some voltage drop will be there across the forward biased diodes which are equal to forward bias voltage. This voltage drop will appear at X during low output condition, so practically low output will not be 0V it is rather 0.6 to 0.7V which is ideally considered as zero.

3.3.2 Transistor AND Gate

An AND logic gate can also be realized from transistor AND gate. The circuit diagram of transistor logic gate is shown in figure 3.12.

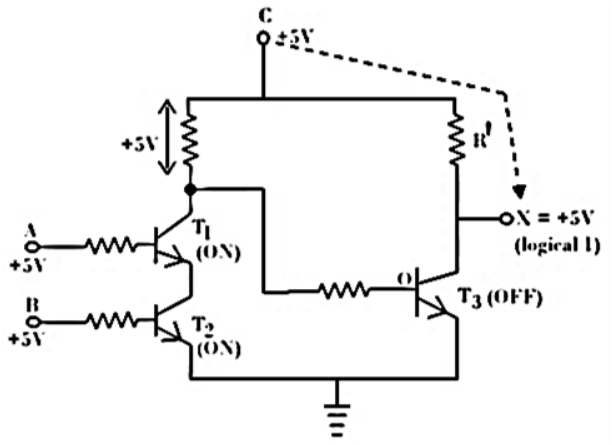


Figure 3.12: Transistor AND Gate

In the above circuit figure 3.12 when A or B or both A and B are grounded or at 0V potential transistor T_1 or T_2 or both T_1 and T_2 are in OFF condition respectively. This is because terminal A and B are base terminal of transistor T_1 and T_2 respectively. Zero base voltage makes a transistor OFF. As the path through $T_1$and T_2 is open circuited base of transistor T_3gets enough potential to makes T_3 ON. Current then starts flowing the supply to ground through T_3. As a result entire supply voltage will drop across R' and potential of terminal X will become low or logical zero.

If any of the transistors T_1 and T_2 is in OFF condition, same result will come at output X as both the transistors are in series. Now we will check what will be the logical value of X, if both A and B are at high logical value. If we apply +5V at both A and B i.e. at base of transistor T_1 and T_2 respectively. This makes both the transistor T_1 and T_2 are in ON condition. Enter supply voltage will drop across R and the base potential of the transistor T_3 will be zero and T_3 becomes in OFF condition. As a result the supply voltage +5V appears at X and X will become logically 1 or high.

3.4 NOT gate

NOT gate is a logical gate which only inverts the input digital signal. Therefore a **NOT gate** sometimes is referred as inverter. A NOT gate always have high or logical 1 output when its input is low or logical 0. On the other hand a logical NOT gate always have low or logical 0 output when input is high or logical 1.

The logical symbol of a NOT gate is shown in figure 3.13

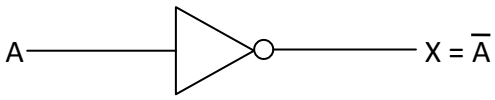


Figure 3.13: Logical Symbol of NOT Gate

If the input binary variable of a NOT gate is considered as A, then the output binary variable of the gate will be Ā. As the symbol of not operation is (-) bar.

The logical Expression of a NOT gate is$X = \bar{A}$

If the value of A is 1, then $\bar{A} = 0$ and in opposite if the value of A is 0 then $\bar{A} = 1$. The truthtable of a **NOT gate** hence can be represented as table 3.4,

Table 3.4 Truth Table of NOT Gate

A	$X = \bar{A}$
0	1
1	0

A NOT gate can easily be realized by using a simple bipolar transistor. The circuit of a NOT gate or transistor inverter is shown in figure 3.14,

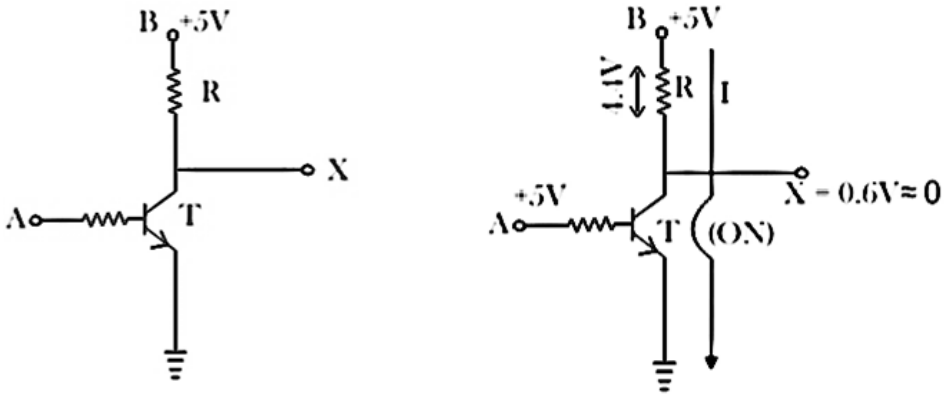


Figure 3.14: Transistor Circuit NOT Gate and with input +5v

Let us examine the above simple circuit (figure 3.14) by applying high input variable, i.e. +5V.At that condition the transistor T gets enough base potential to make the transistor T 'ON'.As soon as the transistor becomes ON, the supply voltage (+5V) at B will get a path to the earth through the resistor R. At ON condition the transistor will behave short circuited ideally, hence entire supply voltage will drop across resistor R and no voltage will appear at X and hence the output of the inverter or NOT gate will be zero.

In actual case, there will be some voltage drop across collector and emitter even at ON condition, of transistor. This collector-emitter voltage is about 0.6V. So, at the above said input condition, entire supply voltage +5V will not drop across resistor instead it will be 5 – 0.6 = 4.4V. So, 0.6V is practically considered as logical zero or low.

Now let us examine the condition, Where, input A = 0V i.e. base terminal of the transistor is given with 0V or grounded (figure 3.15).

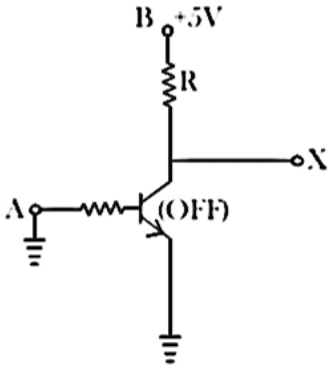


Figure 3.15: Transistor Circuit NOT Gate and with input 0v

At that condition, as the base of the transistor is at 0 potential, the transistor T will be in OFF condition and hence, the supply voltage will not get any path to the earth and entire supply voltage will appear at output terminal of the NOT gate high or logical 1, when input terminal A is low or logical zero.

3.5 NAND gates

AND, NOT and OR gates are the basic gates; we can create any logic gate or any Boolean expression by combining them. Now NAND and NORgates have the particular property that any one of them can create any logical Boolean expression if designed in a proper way. Now we will look at the operation of each gate separately as universal gates.
When output of an AND gate is inverted through a NOT gate, the operation is called NAND operation. The logic gate which performs this NAND operation is called NAND gate.

ie.,ANOT gate followed by an AND gate makes a NAND gate.

The basis logical construction of the NAND gate is shown in figure 3.16

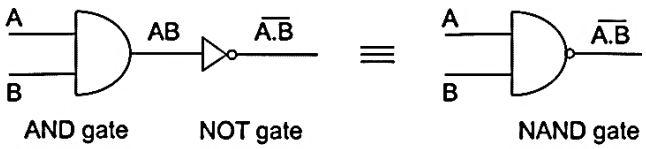


Figure 3.16: Logical Symbol of NAND Gate

The symbol of NAND gate is similar to AND gate but one bubble is drawn at the output point of the AND gate, in the case of NAND gate. NAND gate actually means “not AND gate” which means, the output of this gate is just reverse of that of a similar AND gate.

We know that the output of the AND gate is only high or 1, when all the inputs are high or 1. In all other cases, the output of AND gate is low or 0. In the case NAND, the case is a just opposite, here, the output is only low or 0 when and only when all inputs of the gate are 1 and in all other cases, the output of NAND gate is high or 1. Hence, truth table of a NAND gate can be written like, Just reverse of the truth table of AND gate which is given in table 3.5

Table 3.5 Truth Table of NAND Gate

Inputs		Output
A	B	$X = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

The logical Expression of NAND gate is $X = \overline{A \cdot B}$

Like AND gate a NAND gate can also be more than two inputs, like 3, 4, input NAND gate. An NAND gate is also referred as universal logic gate as all the binary operations can be realized by using only NAND gates. There are three basic binary operations, AND, OR and NOT. By these three basic operations, one can realize all complex binary operations. Now, we will show all these three binary operations can be realized by using only NAND gates.

3.5.1 Realizing NOT Gate Using NAND Gate

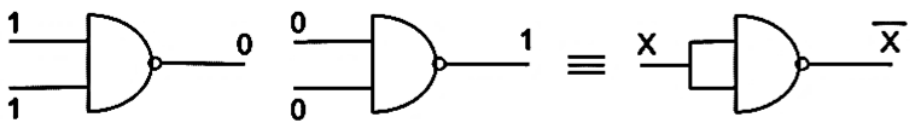


Figure 3.17: NOT Operation using NAND Gate

When, both inputs of a two inputs NAND gate are zero, the output is 1 and both inputs of the NAND gate are 1, the output is 0. Hence a NOT gate can very easily be realized

from NAND gate just by applying common inputs to the NAND gate. This is done by short circuiting all the inputs terminals of a NAND gate (figure 3.17). Where, x is either 1 or 0.

3.5.2 Realizing AND Gate Using NAND Gate

A NAND gate is a NOT gate followed by an AND gate, so if we can cancel the effect of NOT gate in a NAND gate it will become an AND gate. Hence, a NOT gate followed by a NAND gate realizes an AND gate. In this case we use the NOT gate which is realized from NAND gate and the logic circuit is shown in figure 3.18

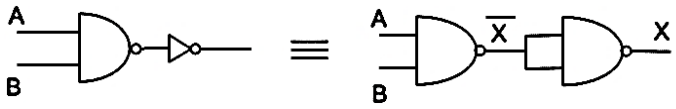


Figure 3.18: AND Operation using NAND Gate

$$X = A \cdot B$$

3.5.3 Realizing OR Gate from NAND Gate

From De Morgan Theorem we know, $X = \overline{\bar{A}.\bar{B}} = \bar{\bar{A}} + \bar{\bar{B}} = A + B$

The above equation is a logical OR operation.

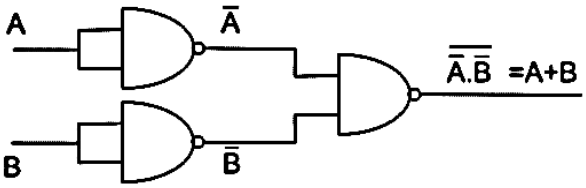


Figure 3.19: OR Operation using NAND Gate

The above logic equation can be represented by gates as shown in figure 3.19, where inputs first inverted then passed through a third NAND gate. The truth table of such circuit is given in table 3.6

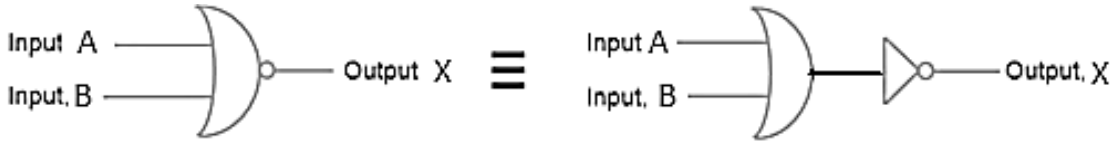
Table 3.6 Truth Table of OR by NAND Gates

Inputs		Output
A	B	$X = \overline{\bar{A} \cdot \bar{B}} = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Now, we have proved that all three basic binary operations can be realized by using only NAND gates. Hence, any other simple or complex binary operation must also be realized by using only NAND gates and hence it is justified to call an NAND gates as universal gates

3.6 NOR Gate

NOR gate is a result of combining NOT gate with an OR gate (figure 3.20). Thus its output is the negation of OR gate output which implies that it has high output only if all of its inputs are low. However for any other combination of inputs, the output will be low as shown by the truth table in table 3.7.



.

Figure 3.20: Logical Symbol of NOR Gate

Table 3.7 Truth Table of NOR Gate

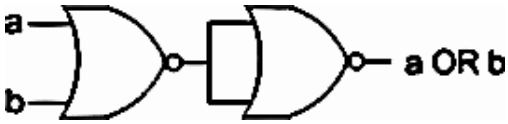
Inputs		Output
A	B	$\boldsymbol{X = \overline{A + B}}$
0	0	1
0	1	0
1	0	0
1	1	0

Logical expression for NOR gate is

$$\boldsymbol{X = \overline{A + B} = \overline{A}\,\overline{B}}$$

3.6.1 NOR gate as universal gate

We have seen how NAND gate can be used to make all the three basic gates by using that alone. Now we will discuss the same in case of NOR gate



.

Figure 3.21: OR Operation Using NOR Gate

The above diagram figure 3.21 is of an OR gate made by only using NOR gates. The output of this gate is exactly similar to that of a single OR gate. As we can see the circuit arrangement of OR gate using NOR gates is similar to that of AND gate using NAND gates.

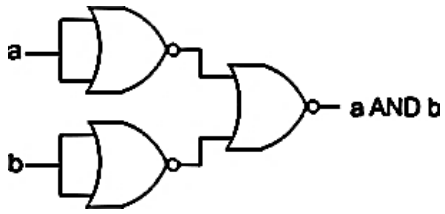


Figure 3.22: AND Operation Using NOR Gate

The above diagram figure 3.22 as the name suggests is of AND gate using only NOR gate, again we can see that the circuit diagram of AND gate using only NOR gate is exactly similar to that of OR gate using only NAND gates. Now we will finally see how a NOT gate can be made by using only NOR gates.

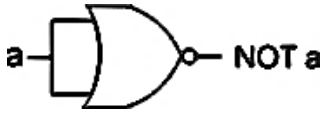


Figure 3.23: NOT Operation Using NOR Gate

The above diagram figure 3.23 is of a NOT gate made by using a NOR gate. The circuit diagram is similar to that of NOT gate made by using only NAND gate. So, from the above discussion it is clear that all the three basic gates (AND, OR, NOT) can be made by only using NOR gate. And thus, it can be aptly termed as Universal Gate.

3.7 XOR gate

Modulo sum of two variables in binary system is like this,

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=0 \longrightarrow Carry\ 1$$

The gate performs this modulo sum operation without including carry is known as X OR gate. An X OR gate is normally two inputs logic gate where, output is only logical 1 when only one input is logical 1. When both inputs are equal, that is either both are 1 or both are 0, the output will be logical 0. This is the reason an XOR gate also called anti-coincidence

gate or inequality detector. This gate is called as XOR or exclusive OR gate because, its output is only 1 when one of its input is exclusively 1.

The truth table of XOR gate is given in table 3.8

Table 3.8 Truth Table of XOR Gate

Inputs		Output
A	B	$X = \mathrm{A} \oplus \mathrm{B}$
0	0	0
0	1	1
1	0	1
1	1	0

The binary operation of above truth table is known as exclusive OR operation and it is represented as, A$\oplus$ B. The symbol of exclusive OR operation is represented by a plus ring surrounded by a circle $\oplus$.

3.7.1 Realization of Two Inputs XOR Gate

The above expression, A$\oplus$ B can be simplified as,

Let us prove the above expression. In first case consider, A = 0 and B = 0.

$$A \oplus B = 0 \oplus 0 = 0.\bar{0} + \bar{0}.0 = 0.1 + 1.0 = 0$$

In second case consider, A = 0 and B = 1.

$$A \oplus B = 0 \oplus 1 = 0.\bar{1} + \bar{0}.1 = 0.0 + 1.1 = 1$$

In third case consider, A = 1 and B = 0.

$$A \oplus B = 1 \oplus 0 = 1.\bar{0} + \bar{1}.0 = 1.1 + 0.0 = 1$$

In fourth case consider, A = 1 and B = 1.

$$A \oplus B = 1 \oplus 1 = 1.\bar{1} + \bar{1}.1 = 1.0 + 0.1 = 0$$

So it is proved that, the Boolean expression for A $\oplus$ B is $A\bar{B} + \bar{A}B$, as this Boolean expression satisfied all output states respect to inputs conditions, of an XOR gate. From this Boolean expression one can easily realize the logical circuit of an XOR gate.

Logical Symbol of XOR Gate is shown in figure 3.24

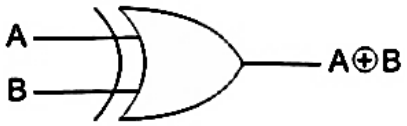


Figure 3.24 LogicalSymbol of XOR Gate

An XOR gate is logically represented in figure 3.25 as

,

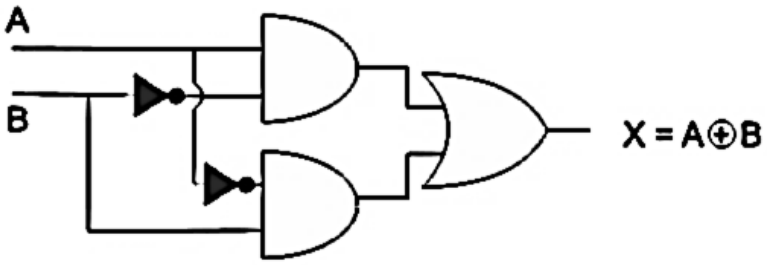


Figure 3.25 Logical Circuit diagram of XOR Gate

3.8 XNOR Gate

XNOR gate is a NOT gate followed by an XOR gate. As we know that XOR operation of inputs A and B is A⊕ B, therefore XNOR operation those inputs will be $\overline{(A \oplus B)}$. That means, output of XOR gate is inverted in XNOR gate. In XOR operation, the output is only 1 when only one input is 1. The output is logical 0 when both inputs are same that means they are either 1 or 0. But in the case of XNOR gate, the output is 0 when only one input is 0 and the output is 1 when both inputs are same that is either both of them are 0 or 1.

The truth table of the XNOR gate is given in table 3.9

Table 3.9 Truth Table of XNOR Gate

Inputs		Output
A	B	$X = \mathrm{A} \odot \mathrm{B}$
0	0	1
0	1	0
1	0	0
1	1	1

The logical XNOR operation is represented by ⊙. That is a dot surrounded by circle. The expression of XNOR operation between variable A and B is represented as. Now again, the truth table is satisfied by the equation

The logical expression is

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

The symbol of XNOR gate is shown in figure 3.26

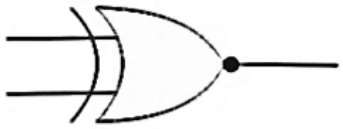


Figure 3.26 Logical Symbol of XNOR Gate

$$When, A = 0, \ and \ B = 0 \ , AB + \overline{A}\ \overline{B} = 0.0 + \overline{0}.\ \overline{0} = 0.0 + 1.1 = 1.$$

$$When, A = 0, \ and \ B = 1, AB + \overline{A}\ \overline{B} = 0.1 + \overline{0}\ .\overline{1} = 0.1 + 1.0 = 0.$$

$$When, A = 1, \ and \ B = 0, AB + \overline{A}\ \overline{B} = 1.0 + \overline{1}\ .\overline{0} = 1.0 + 0.1 = 0.$$

$$When, A = 1, \ and \ B = 1, AB + \overline{A}\ \overline{B} = 1.1 + \overline{1}\ .\overline{1} = 1.1 + 0.0 = 1.$$

Hence, it is proved that A ⊙ B = $AB + \bar{A}\bar{B}$. The same can be proved by using K-map also.

The expression of XNOR operation can be realized by using two NOT gates, two AND gates and one OR gate is shown in figure 3.27

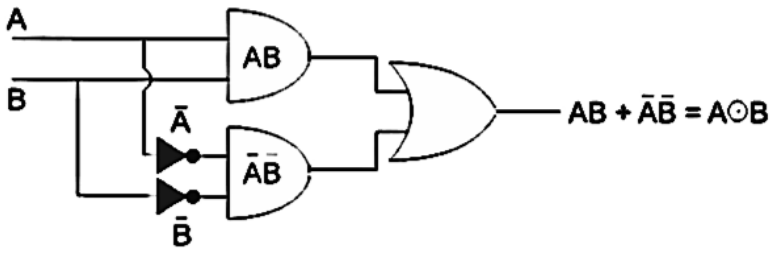


Figure 3.27 Logical Circuit diagram of XNOR Gate

Unit IV Combinational Circuit

Half adder- full adder- half subtractor- full subtractor- binary adder- BCD adder-decoder-encoder-multiplexer-demultiplexer.

Binary adder is one of the basic combinational logic circuits. The outputs of a combinational logic circuit depend on the present input only. In other words, outputs of combinational logic circuit do not depend upon any previously applied inputs. It does not require any memory like component. Binary adder is one of the basic combinational logic circuits as present state of input variables.

4.1 Half Adder

Before designing a binary adder, let us know some basic rules of binary addition. The most basic binary addition is addition of two single bit binary numbers that is addition of two binary digits. The binary digits are 0 and 1.Hence, there must be four possible combinations of binary addition of two binary bits

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

In the above list, first three binary operations result in one bit but fourth one result in two bits. In one bit binary addition, if augend and addend are 1, the sum will have two digits. The higher significant bit (HSB) or Left side bit is called carry and the least significant bit (LSB) or right side bit of the result is called sum bit. The logical circuit performs this one bit binary addition is called half adder.

4.1.1 Design of Half Adder

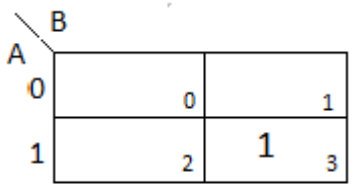
For designing a half adder logic circuit, we first have to draw the truth table for two input variables i.e. the augend and addend bits, two outputs variables carry and sum bits. In first three binary additions, there is no carry hence the carry in these cases are considered as 0

Table 4.1: Truth Table for Half Adder

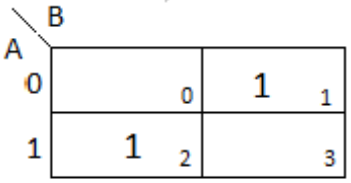
Inputs		Outputs	
Augend(A)	Addend(B)	Carry(C)	Sum(S)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

4.1.2 K-map for Half Adder

Now from this truth table we can draw K-map for carries and sums separately.



K-map for Carry



K-map for Sum

For above K-maps we get, $Carry\ C = AB \ \ \& \ Sum\ S = A\bar{B} + \bar{A}B$

Although from truth table 4.1 it is clearly seen that carry (C) column signifies AND operation and sum (S) column signifies XOR operation between input variables but till we went through K-map as it is general practice to do so for more complex binary logic operations. Hence, the logical design of Half Adder would be as figure 4.1

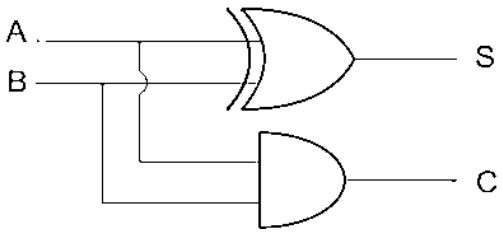


Figure 4.1: Half Adder Circuit

4.2 Full Adder

Before knowing about full adder, let us know what is full addition? For that let us consider the example

$$\begin{array}{r} 1\,1\,0\,1 \\ +\ 0\,1\,1\,1 \\ \hline 1\,0\,1\,0\,0 \end{array}$$

There, are two four bits binary numbers 1101 and 0111 which we have to add. The process of binary addition is like follows,

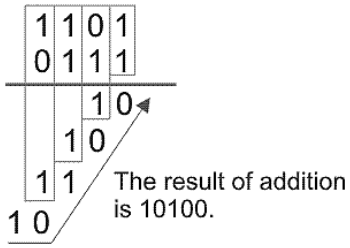
1. We have to add first least significant bits (LSB) of both 4bits binary numbers first and

$$\begin{array}{r} 1\,1\,0\,\boxed{1} \\ 0\,1\,1\,\boxed{1} \\ \hline 1\,0 \end{array}$$

this will result a two bits binary number. Here, LSB of 1101 and 0111 are 1,

Hence 1 + 1 = 10. The LSB of 10 is 0 and higher significant bit (HSB) is 1.

2. The LSB of the result is sum and to be put at the least significant position of the final result of the sum, and HSB of the two bits results will be carry and to be added with next higher significant bit of two 4bits augend and addend are 0 and 1 and the carry of previous result i.e. 1 to be added with 0 and 1. $0 + 1 + 1 = 10$
3. After this addition, that is next higher than least significant bit of bits of both binary augend and addend and it is previous carry we get another two bits result. This also has carry and sum. Here also we will write sum at final result and add the carry to the next higher significant bits of augend and addend. This will continue up to most significant bit of augend and addend.



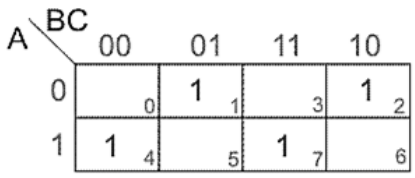
4.2.1Full Adder

Full adder is a conditional circuit which performs full binary addition that means it adds two bits and a carry and outputs a sum bit and a carry bit. Any bit of augend can either be 1 or 0 and we can represent with variable A, similarly any bit of addend we represent with variable B. The carry after addition of same significant bit of augend and addend can represent by C. Hence truth table for all combinations of A, B and C is as follows,

Table 4.2: Truth Table for Full Adder

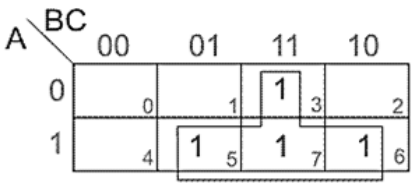
Augend (A)	Addend (B)	Carry (C)	Sum (S)	Final Carry C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the above table 4.2, we can draw K-map for sum (s) and final carry (C_{out}). Hence, from K-maps, the logical expression and logical diagram figure 4.2 is found as follow.



K-map for Sum (S)

Sum $= A\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C}$



K-map for Carry (C_{out})

$C_{out} = AC + BC + AB$

$S = A\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C}$

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

$S = C\left(\overline{\bar{A}B + A\bar{B}}\right) + \bar{C}(\bar{A}B + A\bar{B})$

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

$C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

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

$C_{out} = (A \oplus B)C + AB$

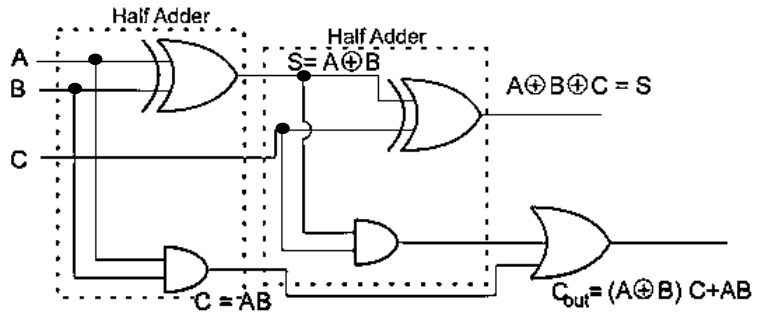


Figure 4.2: Full Adder Circuit

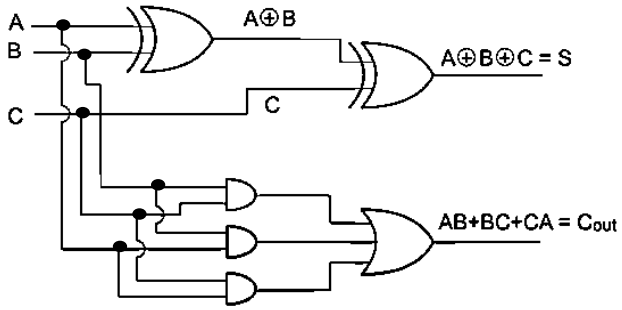


Figure 4.2a: Full Adder Circuit

4.3 Half Subtractor

Half subtractor is a combinational circuit which performs subtraction of single bit binary numbers. The subtraction combinations of two single bit binary numbers can be,

$0 - 0 = 0$

$0 - 1 = 1\ with\ borrow\ 1$

$1 - 0 = 1$

$1 - 1 = 0$

The circuit of logical half subtractor is

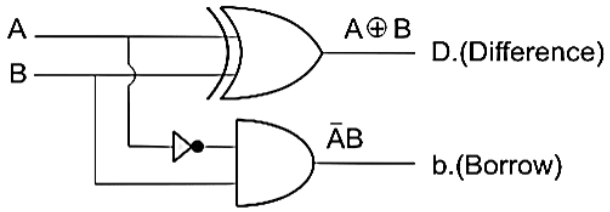


Figure 4.3: Half Subtractor Circuit

The truth table with all differences (D) and borrow (b) is

Table 4.3: Truth Table for Half Subtractor

Minuend (A)	Subtrahend (B)	Difference (D)	Borrow (b)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Hence, from truth table it is found that, The logical expression using logic gates can be represented as.

$$D = A \oplus B \ \ and \ \ b = \bar{A}B$$

4.4 Full Subtractor

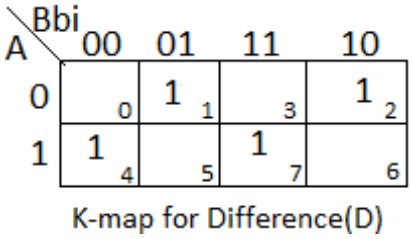
This is not practical to perform subtraction only between two single bit binary numbers. Instead binary numbers are always multi-bits. The subtraction of two binary numbers is performed bit by bit from right (LSB) to left (MSB). During subtraction of same significant bit of minuend and subtrahend, there may be one borrow bit along with difference bit. This borrow bit (either 0 or 1) is to be added to the next higher significant bit of minuend and then next corresponding bit of subtrahend to be subtracted from this. It will continue up to MSB. The combinational logic circuit performs this operation is called full subtractor. Hence, full subtractor is similar to half subtractor but inputs in full subtractor are three instead of two.

Two inputs are for the minuend and subtrahend bits and third input is for borrowed which comes from previous bits subtraction. The outputs of full adder are similar to that of half adder, these are difference (D) and borrow (b). The combination of minuend bit (A), subtrahend bit (B) and input borrow (bi) and their respective differences (D) and output borrows (b) are represented as a truth tablein table 4.4

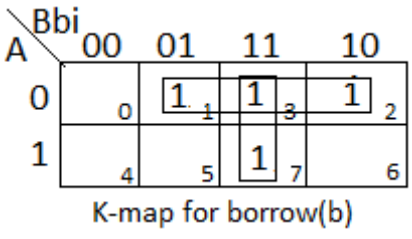
Table 4.4: Truth Table of Full Subtractor

Minuend (A)	Subtrahend (B)	Input borrow (bi)	Difference (D)	Borrow (b)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Then draw K-map for Difference and borrow. And from K-map the logical expression and logical circuit are obtained.



K-map for Difference(D)



K-map for borrow(b)

$$D = A\bar{B}\bar{bi} + \bar{A}\bar{B}bi + ABbi + \bar{A}B\bar{bi}$$

$$b = \bar{A}\bar{B}bi + \bar{A}Bbi + \bar{A}B\bar{bi} + ABbi$$

$$D = A\bar{B}\bar{bi} + ABbi + \bar{A}B\bar{bi}$$

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

$$D = \bar{bi}(A\bar{B} + \bar{A}B) + bi\overline{(A\bar{B} + \bar{A}B)}$$

$$D = bi \oplus A \oplus B$$

$$D = A \oplus B \oplus bi$$

$$b = \bar{A}\bar{B}bi + \bar{A}Bbi + \bar{A}B\bar{bi} + ABbi$$

$$b = \bar{A}B\left(bi + \bar{bi}\right) + (AB + \bar{A}\bar{B})bi$$

$$b = \bar{A}B + (\overline{A \otimes B})bi$$

The circuit of logical full subtractor is

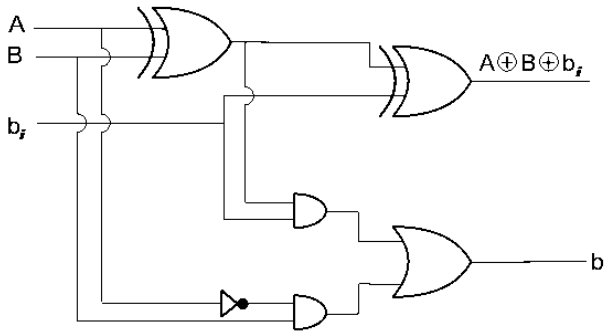


Figure 4.4: Full Subtractor Circuit

4.5 Binary Parallel Adder

A full binary adder performs addition of any single bit of one binary number, same significant or same position bit of another binary numbers and carry comes from result of addition of previous right side bits of both binary numbers. But a single full adder cannot add more than one bits binary number instantly. This can be done only by connecting as many full adders as the number of bits of the binary numbers whose addition is to be performed. This parallel combination of full adders which performs addition of specific bits binary numbers is called binary parallel adder. For adding two 4 bit binary numbers we have to connect 4 full adders to make 4 bit parallel adder. The inter connection of 4 full adder in 4bit parallel adder is shown in figure 4.5.

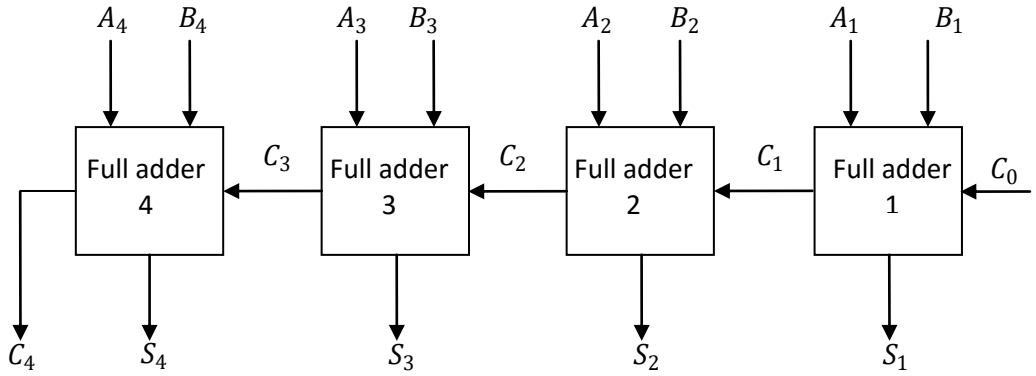


Figure 4.5: Binary parallel Adder

Let us study the explanation of the above circuit by taking an example of addition of two 4 bit binary numbers. Let us add 1011 with 1101.

$$\begin{array}{r} 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ \hline 1\,1\,0\,0\,0 \end{array}$$

$Here, \quad A_1 = 1, \quad A_2 = 1, \quad A_3 = 0, \quad A_4 = 1$

$B_1 = 1, \quad B_2 = 0, \quad B_3 = 1, \quad B_4 = 1$ As there is no previous carry C_0=0.

$Now, \quad C_0 + A_1 + B_1 = 0 + 1 + 1 = 10 \rightarrow S_1 = 0, \quad C_1 = 1$

$$C_1 + A_2 + B_2 = 1 + 1 + 0 = 10 \rightarrow S_2 = 0, \quad C_2 = 1$$

$$C_2 + A_3 + B_3 = 1 + 0 + 1 = 10 \rightarrow S_3 = 0, \quad C_3 = 1$$

$$C_3 + A_4 + B_4 = 1 + 1 + 1 = 10 \rightarrow S_4 = 1, \quad C_4 = 1$$

Therefore, final result of the addition would be $C_4S_4S_3S_2S_1 = 1100$ The 1 bit, 2 bits and 4 bits parallel adder ICs are available in market.

4.6 Binary Subtractor

To study about binary subtractor, first discuss the method of subtracting two multi-bit binary numbers.

```
  110011
- 100101
--------
  001110
```

For above subtraction the following general rules are used,

0 – 0 = 0

0 – 1 = 1 with borrow 1

1 – 0 = 1

1 – 1 = 0

and borrow 1 which to be added to next higher significant bit of first binary number. Then same positioned bit of second binary number would be subtracted from that. But there are other methods by which two binary numbers can be subtracted confidently. One of these is 2's complement method of subtraction. Here, first binary number (from which another binary number to be subtracted) is kept as it is. Then each bit of second binary numbers (which to be subtracted) is complemented. Then 1 is added to LSB of complemented second binary number. This results 2's complement of second binary number. Now finally we add first binary number with 2's complement of the second binary number and we get final result of subtraction

In the previous example, First binary number was 110011 and second binary number was 100101. Complement or 1's complement of 100101 is 011010. Now by adding 1 with LSB of this 1's complement number we get,

```
  011010
+      1
--------
  11011
```

```
  110011
  011011
-------
1001110
```

Now by adding first number, 110011 and 2's complement of second number i.e. 11011. We get, 1001110 .Hence, 4 bit binary subtractor can be drawn like figure 4.6

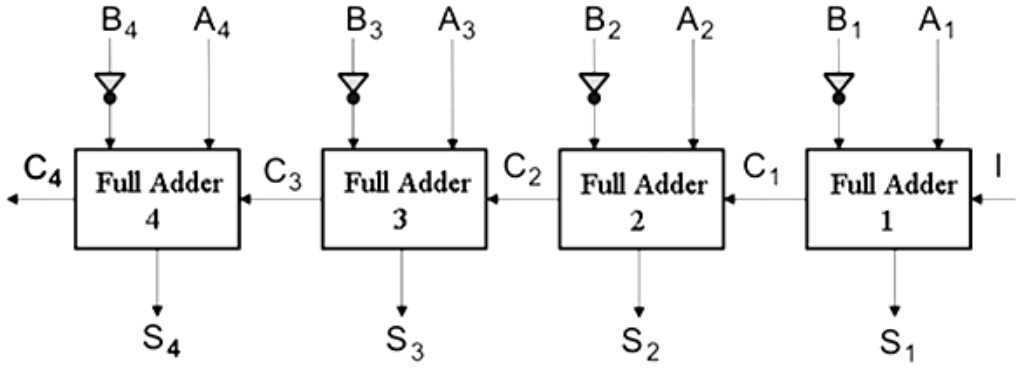


Figure 4.6: Binary Subtractor Circuit

Here, A_4, A_3, A_2, A_1 is minuend and B_4, B_3, B_2, B_1 is subtrahend. S_4, S_3, S_2, S_1 is result of subtraction where C4 is final carry which is ignored.

4.7 Binary Adder Subtractor

We have already designed 4 bits binary parallel adder and 4 bit binary subtractor. We have also seen that both circuits are more or less same except in subtractor the subtrahend bit inputs are inverted with input borrow bit at LSB is 1.

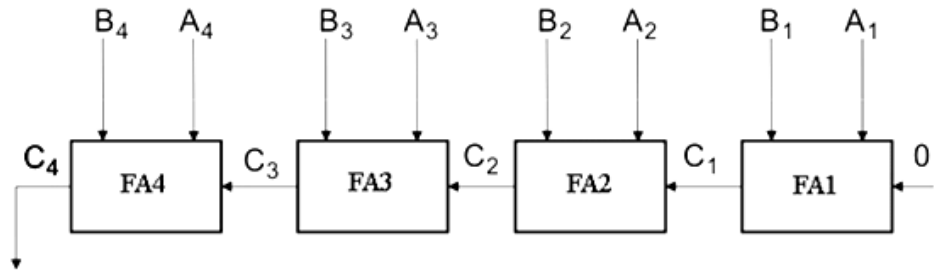


Figure 4.7: Four bit Full adder

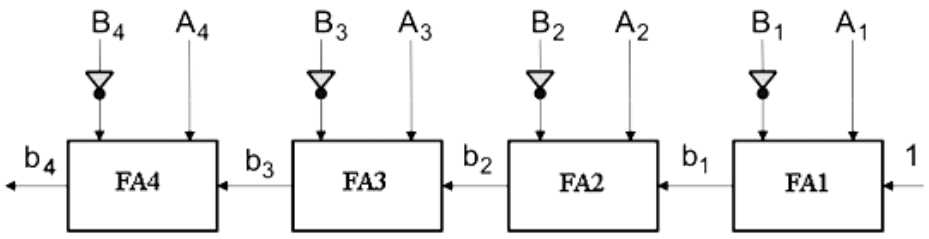


Figure 4.8: Four bit Full Subtractor

In the above 4 bit full adder circuit, third input to LSB Adder (FA1) is 1. In addition to that, in full subtractor subtrahend bits, i.e. B_1, B_2, B_3 and B_4 are inverted. We can combine these two circuits (Adder and Subtractor) in one circuit by controlling B_1, B_2, B_3 and B_4 terminals and third input of LSB adder unit (FAI). We know that, So, we can use XOR gate at each input B_1, B_2, B_3 and B_4 with control input M (either 1 or 0). Now, if M = 1, B_1, B_2, B_3 and B_4 will be complemented. At the same time if third input of FA1 is 1, the circuit becomes subtractor. So, M = 1 is also to be fed to the third input of FA1 in subtractor.

$$1 \oplus B = 1.\bar{B} + 0.B = \bar{B}$$

$$0 \oplus B = 0.\bar{B} + 1.B = B$$

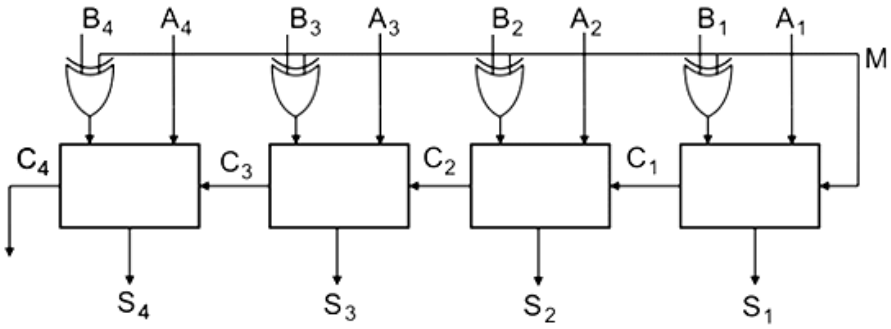


Figure 4.9: Four bit Full adder and Subtractor

4.8 BCD Addition

Like other number system in BCD arithmetical operation may be required. BCD is a numerical code which has several rules for addition. The rules are given below in three steps with an example to make the idea of BCD Addition clear.

At first the given number are to be added using the rules of binary. For example,

Case 1:	Case 2:
1010	0001
+ 0101	+ 0101
1111	0110

In second step we have to judge the result of addition. Here two cases are shown to describe the rules of BCD Addition**.** In case 1 the result of addition of two binary number is greater than 9, which is not valid for BCD number. But the result of addition in case 2 is less than 9, which is valid for BCD numbers.

If the four bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 whose binary equivalent is $(0110)_2$ to the result of addition. Then the resultant that we would get will be a valid binary coded number. In case 1 the result was $(1111)_2$, which is greater than 9 so we have to add 6 or $(0110)_2$ to it.$(1111)_2 + (0110)_2 = 0001\ 0101 = 15$

As you can see the result is valid in BCD. But in case 2 the result was already valid BCD, so there is no need to add 6. This is how BCD Addition could be. Now a question may arrive that why 6 is being added to the addition result in case BCD Addition instead of any other numbers. It is done to skip the six invalid states of binary coded decimal i.e from 10 to 15 and again return to the BCD codes. Now the idea of BCD Addition can be cleared from two more examples.

Example:1

Let, 0101 is added with 0110.

$$\begin{array}{rl} & 0101 \\ + & 0110 \\ \hline & 1011 \rightarrow \text{Invalid BCD number} \\ + & 0110 \rightarrow \text{Add 6} \\ \hline 0001 & 0001 \rightarrow \text{Valid BCD number} \end{array}$$

To verify it

We have $(0101)_2 \rightarrow (5)_{10}$& $(0110)_2 \rightarrow (6)_{10}$

The sum is $(5)_{10} + (6)_{10} = (11)_{10}$

Example:2

Now let 0001 0001 is added to 0010 0110.

$$\begin{array}{rl} & 0001\ 0001 \\ + & 0010\ 0110 \\ \hline & 0011\ 0111 \longrightarrow \text{Valid BCD number} \end{array}$$

$(0001\ 0001)_{BCD} \rightarrow (11)_{10}$, $(0010\ 0110)_{BCD} \rightarrow (26)_{10}$

The sum is $(11)_{10} + (26)_{10} = (37)_{10}$

$(0011\ 0111)_{BCD} \rightarrow (37)_{10}$

So no need to add 6 as because both $(0011)_2 = (3)_{10}$ and $(0111)_2 = (7)_{10}$ are less than $(9)_{10}$. This is the process of BCD Addition.

4.9 Encoder

When we insert any character or symbol to a digital system, through key board, it is needed to be encoded in machine readable farm. Digital systems like computer etc, cannot read the characters or symbol directly. The system reads and computes any characters, numbers and symbols in their digital form. An encoder does the job that means, it converts different human readable characters or symbol to their equivalent digital format. An encoder is basically multi inputs and multi outputs digital logic circuit, which has as many inputs as the number of character to be encoded and as many outputs as the number of bits in encoded form of characters. Suppose we have to design an encoder which will encode 10 characters (from 0 to 9). The encoded form of each character would be 4 bit binary equivalent. Then the encoder will have 10 numbers of input lines and each for one character. There will be four output lines to represent 4 bit encoded form of each input character.

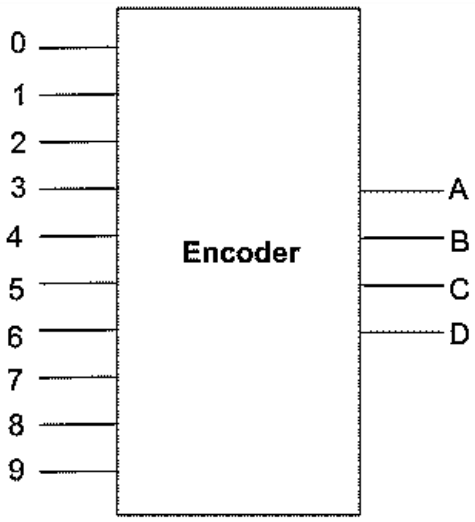


Figure 4.10: Encoder Block Diagram

Similarly for encoding M numbers of characters in N bit format, we need M input N output Decimal to Binary Encoder.

In encoder normally, the input of which encoding to be done, is made high, other all inputs remain low at that time. That means a digital encoder works on active high input. To understand about a digital encoder let us design the above decimal to binary encodes. The

Truth table for 10 inputs 4 output encoder would be

Table 4.5: Truth Table for Decimal to Binary Encoder

Inputs	Decimal	Binary			
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	1	1

From the truth table it is found, that output A would be high at D_8 , D_9

so it can be written as $A = D_8 + D_9$

Similarly, $B = D_4 + D_5 + D_6 + D_7$

$$C = D_2 + D_3 + D_6 + D_7 + D_9$$

$$D = D_1 + D_3 + D_5 + D_7 + D_9$$

From the above equations the logic circuit drawn as follows. This circuit can also be considered as decimal to BCD encoder

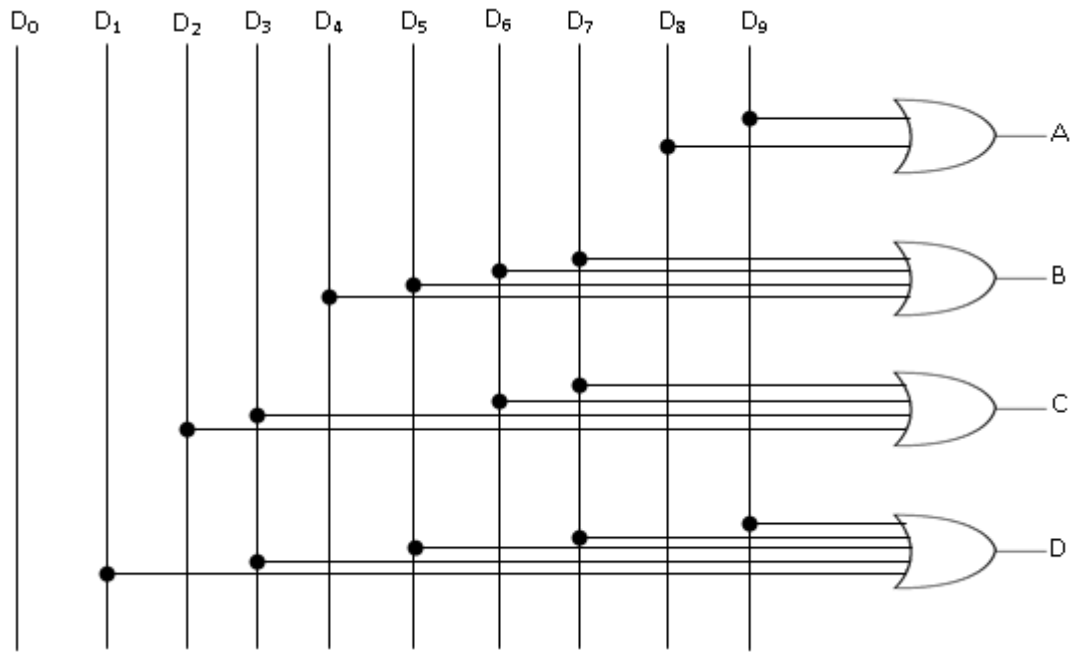


Figure 4.11: Decimal to BCD Encoder Circuit

4.9.1 Octal to Binary Encoder

The octal numbers system has base of 8. Hence the number of digits used in octal system is 8 and the octal digits are 0 to 7. Hence, there will be eight input line in a basic Octal to binary encoder. As binary equivalent of numbers 0 to 7 can be represented by only three binary bits, there will be three output lines to represent bits of binary equivalent of octal number. The truth table in table 4.6, logical relations between inputs and outputs and the corresponding logic circuit figure 4.12 are shown as follows,

Table 4.6: Truth Table for Octal to Binary

Inputs	Octal	Binary		
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_3 + D_5 + D_7$

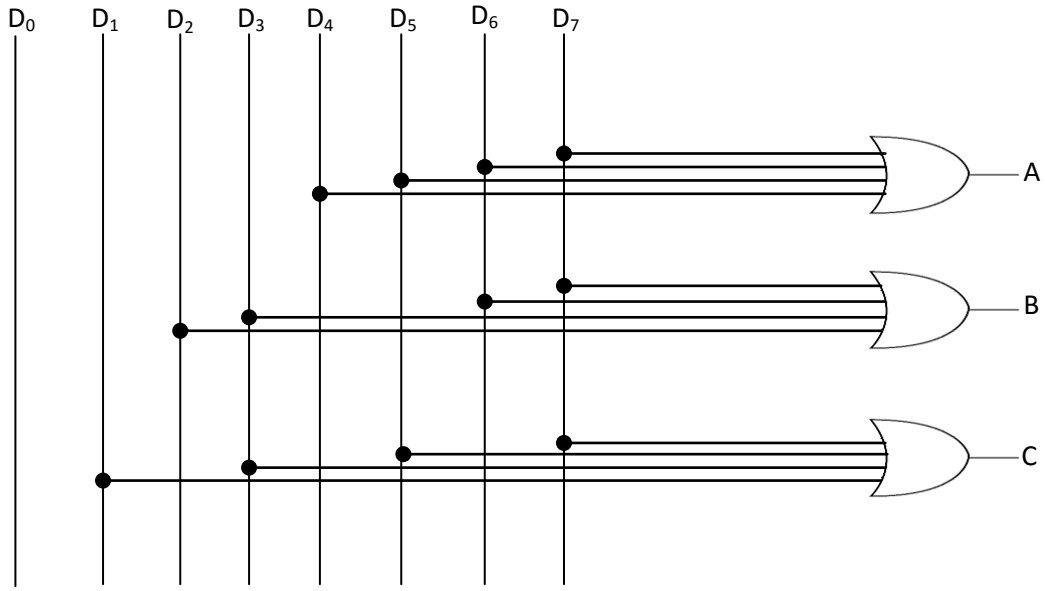


Figure 4.12: Octal to BCD Encoder Circuit

4.10 Binary Decoder

Decoder is a combinational circuit with n input lines and $2n$ output lines. In functionality, a binary decoder converts a definite sequence of input bits into a specific pattern as decided by the user based on the requirement. Figure4.13 shows a binary decoder with one enable pin and 3 input lines which further results in 8 lines at its output.

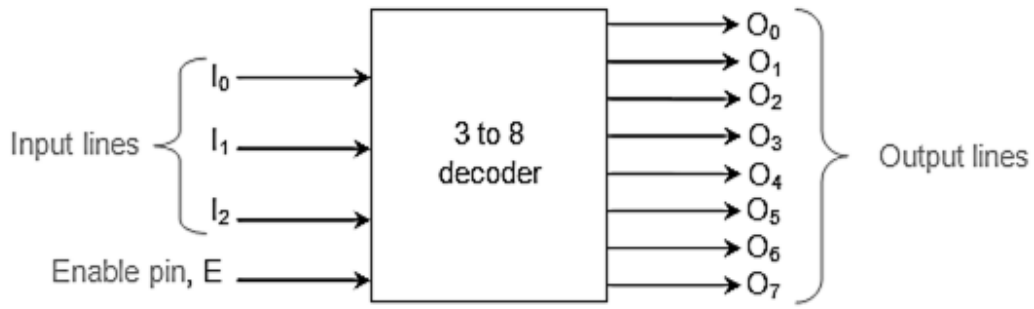


Figure 4.13: Decoder Block Diagram

The output sequence of a decoder for a particular input pattern is realized using its truth table. Table 4.7 shows the truth table for the decoder of Figure 4.13 which shows that when the enable is low, all the output lines are low, no matter what the input sequence be. This indicates the OFF state of the decoder which can also be considered to be its reset state. Thus one has to drive high on the enable pin to realize the functionality of the decoder.

Table 4.7: Truth table for 3 to 8 decoder

Enable Pin	Input Lines			Output Lines							
E	I_2	I_1	I_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0
X denotes don't care condition											

Table 4.7 shows that for the input sequence$I_2I_1I_0 = 000$, the output pin O_0 of the decoder is high while all other bits (O_7 down to O_1) remain low. Likewise, for the input sequence of 001, only O_1 is high. Similar observation shows that only one output line is high for any given input bit pattern i.e. O_2 is high for 010, O_3 is high for 011, O_4 is high for 100, O_5 is high for 101, O_6 is high for 110 and O_7 is high for 111. Thus the Boolean equations for the outputs of the 3 to 8 decodershown in Figure 4.13 are given by

$$O_0 = E\,\bar{I}_2\bar{I}_1\bar{I}_0 \quad (1)$$
$$O_1 = E\,\bar{I}_2\bar{I}_1 I_0 \quad (2)$$
$$O_2 = E\,\bar{I}_2 I_1\bar{I}_0 \quad (3)$$
$$O_3 = E\,\bar{I}_2 I_1 I_0 \quad (4)$$
$$O_4 = E\,I_2\bar{I}_1\bar{I}_0 \quad (5)$$
$$O_5 = E\,I_2\bar{I}_1 I_0 \quad (6)$$
$$O_6 = E\,I_2 I_1\bar{I}_0 \quad (7)$$
$$O_7 = E\,I_2 I_1 I_0 \quad (8)$$

Equations (1) to (8) show that the decoder of Figure 4.13 can be designed using AND gate and NOT gate as shown by Figure 4.14. This is due to the fact that the output lines are nothing but the logical 'and' of either input or its negation with the enablesignal. The analogy presented here for 3 to 8 decoder holds good for any n to $2n$ decoder. However the output bit pattern need not be the same as the one explained. These kinds of decoders are used in the applications such as data multiplexing, seven segment display and so on.

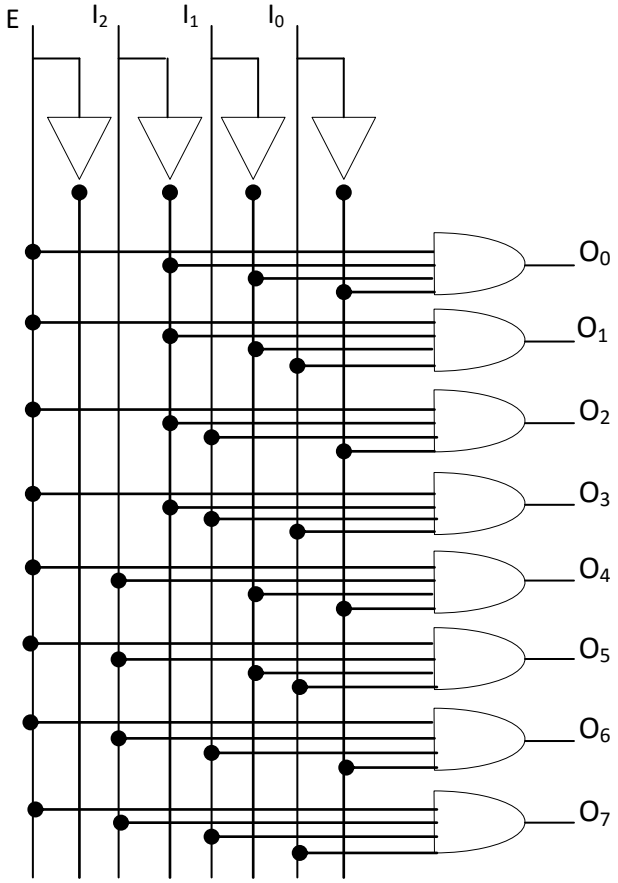


Figure 4.14: Binary Decoder Circuit using Basic Gates

4.11 Mutliplexer:

A multiplexer is a circuit that accepts many input but give only one output. A de-multiplexer function exactly in the reverse of a multiplexer, that is a de-multiplexer accepts only one input and gives many outputs. Generally multiplexer and de-multiplexer are used together, because of the communication systems are bi directional.

Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a signal output. An simple example of an non electronic circuit of a multiplexer is a single pole multi-position switch.Multi-position switches are widely

used in many electronics circuits. However circuits that operate at high speed require the multiplexer to be automatically selected. A mechanical switch cannot perform this task satisfactorily. Therefore, multiplexer used to perform high speed switching are constructed of electronic components.

Multiplexer handle two type of data that is analog and digital. For analog application, multiplexer are built of relays and transistor switches. For digital application, they are built from standard logic gates.

The multiplexer used for digital applications, also called digital multiplexer, is a circuit with many input but only one output. By applying control signals, we can steer any input to the output. Few types of multiplexer are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.

Following figure shows the general idea of a multiplexer with n input signal, m control signals and one output signal.

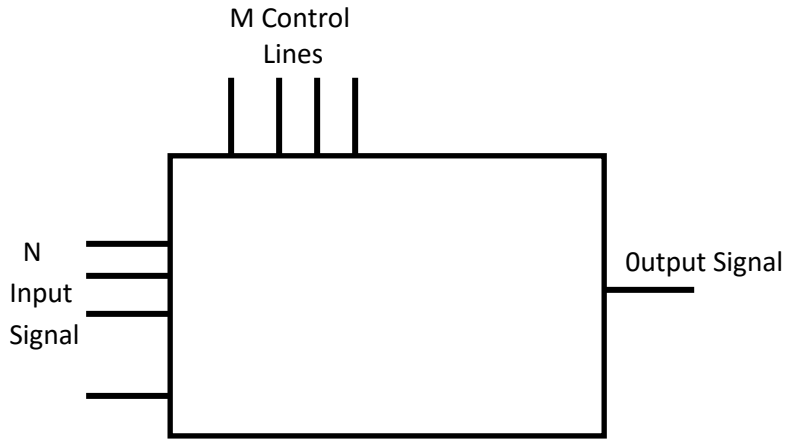


Figure 4.15: Multiplexer Pin Diagram

4.11.1 Understanding 4-to-1 Multiplexer:

The 4-to-1 multiplexer has 4 input bit, 2 control bits, and 1 output bit. The four input bits are D_0,D_1,D_2 and D_3 only one of this is transmitted to the output y. The output depends on the value of AB which is the control inputs. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in fig. when AB = 00, the upper AND gate is enabled while all other AND gates are disabled. Therefore, data bit D_0 is transmitted to the output, giving

$$Y = D_o.$$

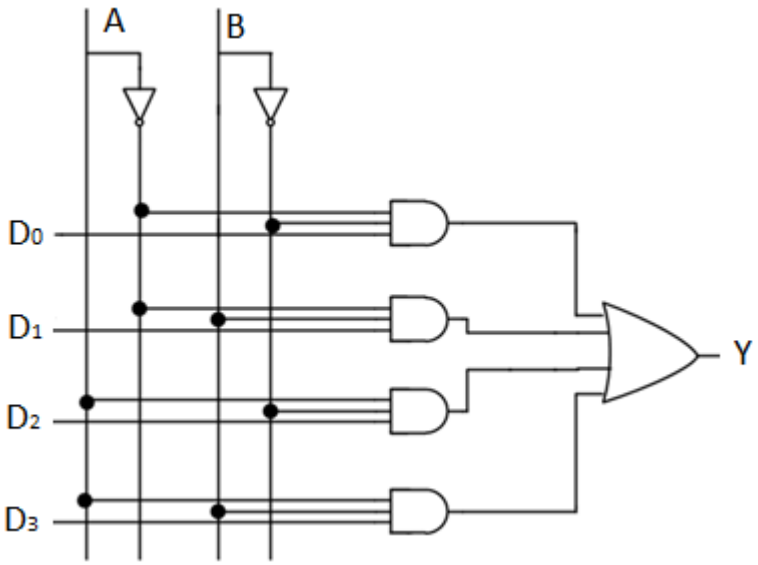


Figure 4.16: Multiplexer Circuit Diagram

If the control input is changed to AB =11, all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and $Y = D_3$.

An example of 4-to-1 multiplexer is IC 74153 in which the output is same as the input.

Another example of 4-to-1 multiplexer is 45352 in which the output is the compliment of the input.

Example of 16-to-1 line multiplexer is IC74150.

4.11.2Applications of Multiplexer:

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers –

Communication system – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.

Telephone network **–** In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.

Computer memory – Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.

Transmission from the computer system of a satellite – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

4.12 Demultiplexer:

Demultiplexer means one to many. A demultiplexer is a circuit with one input and many output. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to 2, 1-to-4, 1-to-8 and 1-to 16 demultiplexer.

Figure 4.17 illustrates the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.

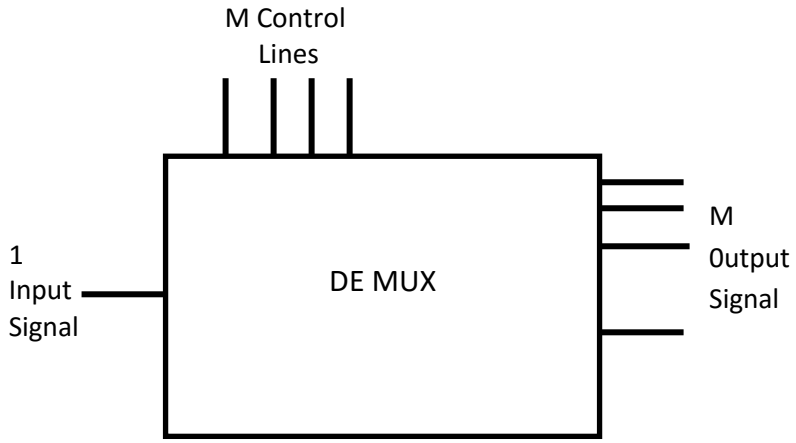


Figure 4.17: DeMultiplexer Pin Diagram

4.12.1 Understanding 1- to-4 Demultiplexer:

The 1-to-4 demultiplexer has 1 input bit, 2 control bit, and 4 output bits. An example of 1-to-4 demultiplexer is IC 74155. The 1-to-4 demultiplexer is shown in figure below-

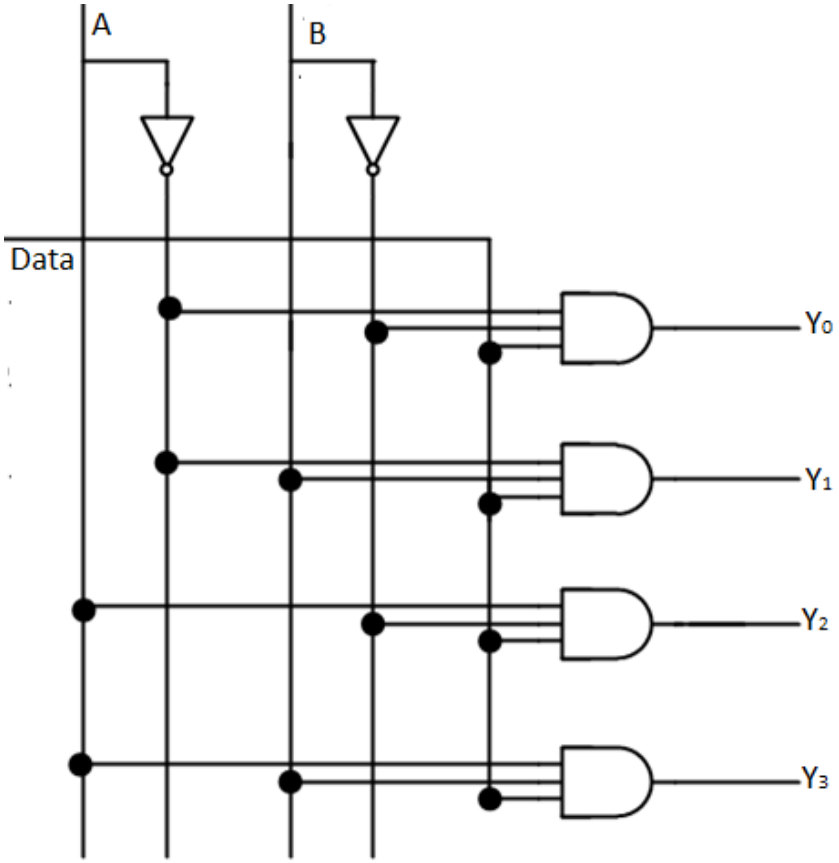


Figure 4.18: DeMultiplexerCircuit Diagram

The input bit is labeled as Data D. This data bit is transmitted to the data bit of the output lines. This depends on the value of AB, the control input.

When AB = 01, the upper second AND gate is enabled while other AND gates are disabled. Therefore, only data bit D is transmitted to the output, giving Y1 = Data.

If D is low, Y1 is low. IF D is high,Y1 is high. The value of Y1 depends upon the value of D. All other outputs are in low state.

If the control input is changed to AB = 10, all the gates are disabled except the third AND gate from the top. Then, D is transmitted only to the Y2 output, and Y2 = Data.

Example of 1-to-16 demultiplexer is IC 74154 it has 1 input bit, 4 control bits and 16 output bit.

4.12.2Applications of Demultiplexer:

Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system where multiplexer are used. Most of the communication system are bidirectional i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.

Communication System – Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process makes the transmission easier. The demultiplexer receives the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.

ALU (Arithmetic Logic Unit) – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.

Serial to parallel converter – A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

R S , J K, D, T flip flops – master slave flip flop- IC 555 timer – astable- multi-vibrator – mono stable multivibrator.

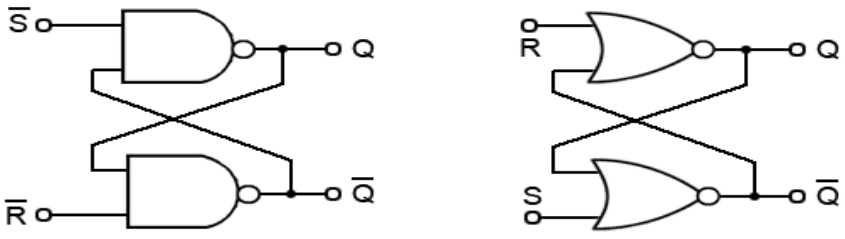
5.1 Flip Flop

A digital computer needs devices which can store information. A flip flop is a binary storage device. It can store binary bit either 0 or 1. It has two stable states HIGH and LOW i.e. 1 and 0. It has the property to remain in one state indefinitely until it is directed by an input signal to switch over to the other state. It is also called bistablemultivibrator.

The basic formation of flip flop is to store data. They can be used to keep a record or what value of variable (input, output or intermediate). Flip flop are also used to exercise control over the functionality of a digital circuit i.e. change the operation of a circuit depending on the state of one or more flip flops. These devices are mainly used in situations which require one or more of these three. Operations; storage and sequencing.

5.2 Latch R S Flip Flop

The RS (Reset Set) flip flop is the simplest flip flop of all and easiest to understand. It is basically a device which has two outputs one output being the inverse or complement of the other, and two inputs. A pulse on one of the inputs takes on to a particular logic state. The outputs will then remain in this state until a similar pulse is applied to the other input. The two inputs are called the Set and Reset input (sometimes called the preset and clear inputs). Such flip flop can be made simply by cross coupling two inverting gates either NAND or NOR gate could be used Figure 5.1(a) shows on RS flip flop using NAND gate and Figure 5.1(b) shows the same circuit using NOR gate.



5.1(a) **5.1(b)**

Figure 5.1: Latch RS Flip Flop Using NAND and NOR Gates

To describe the circuit of Figure 5.1(a), assume that initially both R and S are at the logic 1 state and that output is at the logic 0 state.

Now, if $Q = 0$and $R = 1$, then these are the states of inputs of gate B, therefore the outputs of gate B is at 1 (making it the inverse of Q i.e. 0). The output of gate B is connected to an input of gate A so if S = 1, both inputs of gate A are at the logic 1 state. This means that the output of gate A must be 0 (as was originally specified). In other words, the 0 state at Q is continuously disabling gate B so that any change in R has no effect. Also the 1 state at$\bar{Q}$is continuously enabling gate A so that any change S will be transmitted to Q. The above conditions constitute one of the stable states of the device referred to as the Reset state since $Q = 0$.

Now suppose that the RS flip flop in the Reset state, the S input goes to 0. The output of gate A i.e. Q will go to 1 and with Q = 1 and R = 1, the output of gates $B(\bar{Q})$ will go to 0 with $(\bar{Q})$now 0 gate A is disabled keeping Q at 1. Consequently, when S returns to the 1 state it has no effect on the flip flop whereas a change in R will cause a change in the output of gate B. The above conditions constitute the other stable state of the device, called the Set state since $Q = 1$. Note that the change of the state of S from 1 to 0 has caused the flip flop to change from the Reset state to the Set state.

There is another input condition which has not yet been considered. That is when both the R and S inputs are taken to the logic state 0. When this happens both Q and $\bar{Q}$will be forced to 1 and will remain so far as long as R and S are kept at 0. However when both inputs return to 1 there is no way of knowing whether the flip flop will latch in the Reset state or the Set state. The condition is said to be indeterminate because of this indeterminate state great care must be taken when using RS flip flop to ensure that both inputs are not instructed simultaneously

Table 5.1: The Truth Table for the NAND RS flip flop

Initial Conditions	Inputs(pulsed)		Final Output	
Q	S	R	Q	$\overline{Q}$
1	0	0	indeterminate	
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0
0	0	0	indeterminate	
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1

or more simply shown in Table 5.2

Table 5.2: Simple NAND RS Flip Flop Truth Table

S	R	Q
0	0	indeterminate
0	1	Set (1)
1	0	Reset(0)
1	1	No change

When NOR gate are used the R and S inputs are transposed compared with the NAND version. Also the stable state when R and S are both 0. A change of state is effected by pulsing the appropriate input to the 1 state. The indeterminate state is now when both R and S are simultaneously at logic 1. Table 5.3 shows this operation.

Table 5.3: NOR Gate RS Flip Flop Truth Table

S	R	Q
0	0	No change
0	1	Reset(0)
1	0	Set (1)
1	1	indeterminate

5.3 Clocked RS Flip Flop

The RS latch flip flop required the direct input but no clock. It is very useful to add clock to control precisely the time at which the flip flop changes the state of its output.

In the clocked RS flip flop the appropriate levels applied to their inputs are blocked till the receipt of a pulse from another source called clock. The flip flop changes state only when clock pulse is applied depending upon the inputs. The basic circuit is shown in Figure 5.2. This circuit is formed by adding two NAND gates at inputs to the RS flip flop. In addition to control inputs Set (S) and Reset (R), there is a clock input (C) also.

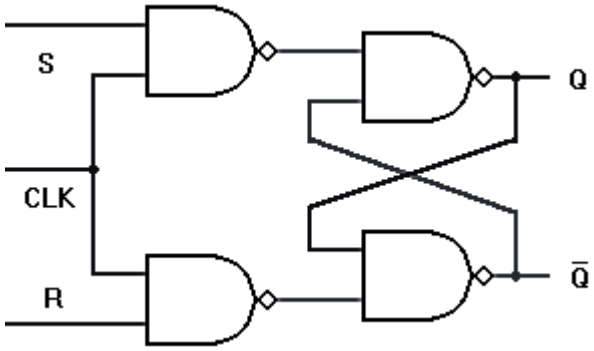


Figure 5.2: Clocked RS Flip Flop

Table 5.4: The Truth Table for the Clocked R-S Flip Flop

Initial Conditions	Inputs(pulsed)		Final output
$\boldsymbol{Q}$	$\boldsymbol{S}$	$\boldsymbol{R}$	$\boldsymbol{Q\ (t+1)}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	indeterminate

The excitation table 5.5 for RS flip flop is very simply derived as given below

Table 5.5: Excitation Table for RS Flip Flop

S	R	Q
0	0	No change
0	1	Reset(0)
1	0	Set (1)
1	1	indeterminate

5.4 D Flip Flop

A D type (Data or delay flip flop) has a single data input in addition to the clock input as shown in Figure 5.3.

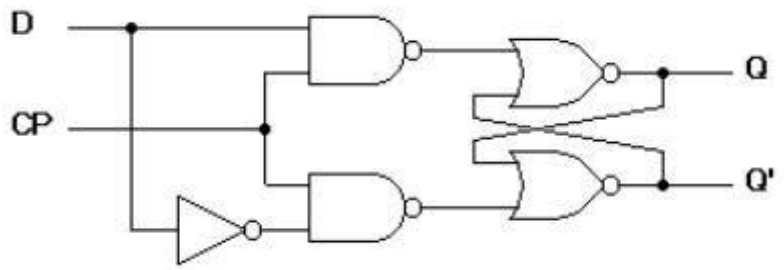


Figure 5.3: D Flip Flop

Basically, such type of flip flop is a modification of clocked RS flip flop gates from a basic Latch flip flop and NOR gates modify it in to a clock RS flip flop. The D input goes directly to S input and its complement through NOT gate, is applied to the R input.

This kind of flip flop prevents the value of D from reaching the output until a clock pulse occurs. The action of circuit is straight forward as follows.

When the clock is low, both NAND gates are disabled;therefore D can change values without affecting the value of$Q$. On the other hand, when the clock is high, both NAND gates are enabled. In this case, Q is forced equal to D when the clock again goes low, Q retains or stores the last value of D. The truth table for such a flip flop is as given below in table 5.6.

Table 5.6: Truth Table for D Flip Flop

S	R	$Q\ (t\ +1)$
0	0	0
0	1	1
1	0	0
1	1	1

The excitation table 5.7 for D flip flop is very simply derived given as under.

Table 5.7: Excitation Table for D Flip Flop

S	Q
0	0
0	1

5.5 JK Flip Flop

One of the most useful and versatile flip flop is the JK flip flop the unique features of a JK flip flop are:

1. If the J and K input are both at 1 and the clock pulse is applied, then the output will change state, regardless of its previous condition.

2. If both J and K inputs are at 0 and the clock pulse is applied there will be no change in the output. There is no indeterminate condition, in the operation of JK flip flop i.e. it has no ambiguous state. The circuit diagram for a JK flip flop is shown in Figure 5.4.

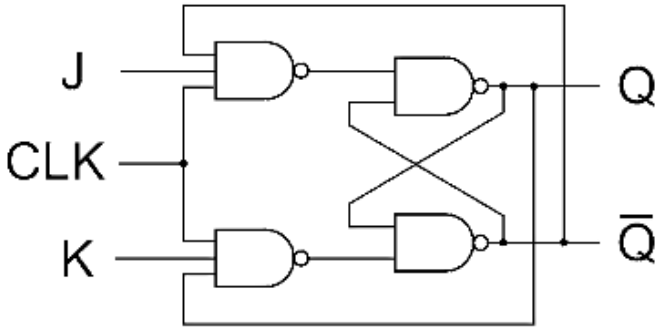


Figure 5.4: JK Flip Flop

$When\ J\ =\ 0\ and\ K\ =\ 0$

These J and K inputs disable the NAND gates, therefore clock pulse have no effect on the flip flop. In other words, Q returns it last value.

$When\ J\ =\ 0\ and\ K\ =\ 1,$

The upper NAND gate is disabled the lower NAND gate is enabled if Q is 1 therefore, flip flop will be reset ($Q\ =\ 0$, $\bar{Q}\ =1$)if not already in that state.

$When\ J\ =\ 1\ and\ K\ =\ 0$

The lower NAND gate is disabled and the upper NAND gate is enabled if $\bar{Q}$is at 1, As a result we will be able to set the flip flop ($Q\ =\ 1$, $\bar{Q}$= 0) if not already set

$When\ J\ =\ 1\ and\ K\ =\ 1$

If Q = 0 the lower NAND gate is disabled the upper NAND gate is enabled. This will set the flip flop and hence Q will be 1. On the other hand if Q = 1, the lower NAND gate is enabled and flip flop will be reset and hence Q will be 0. In other words , when J and K are both high, the clock pulses cause the JK flip flop to toggle. Truth table 5.8 for JK flip flop is shown

Table 5.8: The truth table for the JK flip flop

Initial Conditions	Inputs(pulsed)		Final Output
$\boldsymbol{Q}$	$\boldsymbol{S}$	$\boldsymbol{R}$	$\boldsymbol{Q(t+1)}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

The excitation table 5.9 for JK flip flop is very simply derived as given in table

Table 5.9: Excitation table for JK Flip Flop

S	R	Q
0	0	No change
0	1	0
1	0	0
1	1	Toggle

5.6 T Flip Flop

A method of avoiding the indeterminate state found in the working of RS flip flop is to provide only one input (the T input) such, flip flop acts as a toggle switch. Toggle means to change in the previous stage i.e. switch to opposite state. It can be constructed from clocked RS flip flop be incorporating feedback from output to input as shown in Figure 5.5.

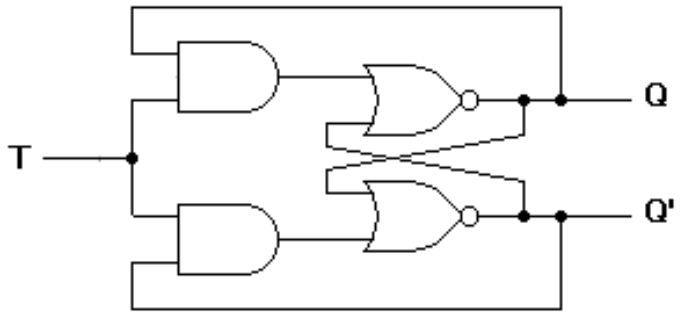


Figure 5.5: T Flip Flop

Such a flip flop is also called toggle flip flop. In such a flip flop a train of extremely narrow triggers drive the T input.Each time one of these triggers arrives, the output of the flip flop changes the stage. For instance Q equals 0 just before the trigger. Then the upper AND gate is enable and the lower AND gate is disabled. When the trigger arrives, it results in a high S input. This sets the Q output to 1. When the next trigger appears at the point T, the lower AND gate is enabled and the trigger passes through to the R input this forces the flip flop to reset.

Since each incoming trigger is alternately changed into the set and reset inputs the flip flop toggles. It takes two triggers to produce one cycle of the output waveform. This means the

output has half the frequency of the input stated another way, a T flip flop divides the input frequency by two. Thus such a circuit is also called a divide by two circuit.

A disadvantage of the toggle flip flop is that the state of the flip flop after a trigger pulse has been applied is only known if the previous state is known. The truth table for a T flip flop is as given table 5.10.

Table 5.10: Truth table for T Flip Flop

Q_n	T	$Q_n + 1$
0	0	0
0	1	1
1	0	1
1	1	0

The excitation table 5.11 for T flip flop is very simply derived as shown.

Table 5.11: Excitation table for T Flip Flop

T	Q
0	Q_n
1	$\bar{Q}_n$

Generally T flip flop ICs are not available. It can be constructed using JK, RS or D flip flop. Figure 5.6 shows the relation of T flip flop using JK flip flop

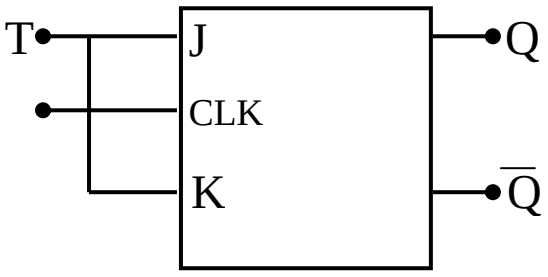


Figure 5.6: T Flip Flop using J K Flip Flop

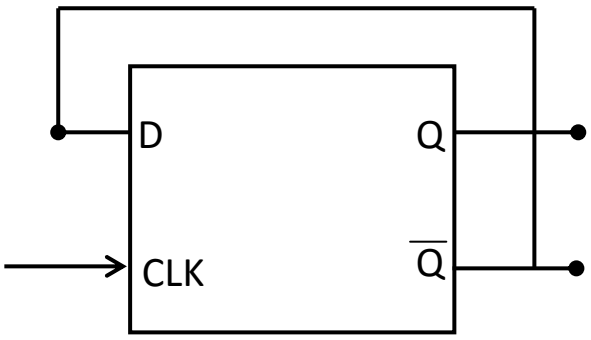


Figure 5.7: D-type Flip Flop connected as toggle stage

A D type flip flop may be modified by external connection as a Ttype stage as shown in Figure 5.7. Since the Q logic is used as Dinput the opposite of the Q output is transferred into the stage each clock pulse. Thus the stage having $Q = 0$transistors$\bar{Q} = 1$, providing a toggle action, if the stage had $Q = 1$ the clock pulse would result in $Q = 0$ being transferred, again providing the toggle operation. The Dtype flip flop connected as in Figure 5.7 will thus operate as a Ttype stage, complementing each clock pulse.

5.7 Master Slave Flip Flop

Figure 5.8 shows the schematic diagram of master slave JK flip flop

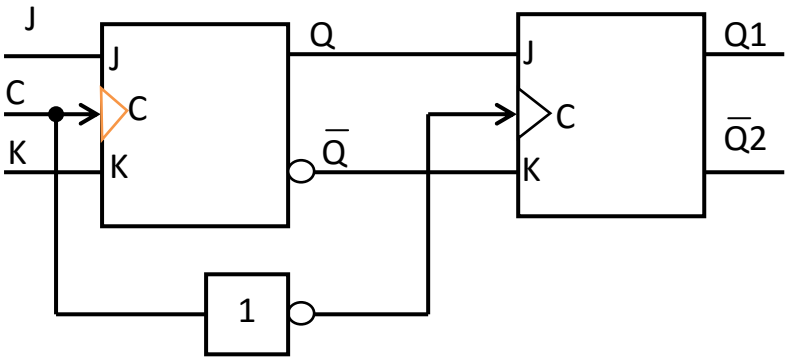


Figure 5.8: Master Slave JK Flip Flop (Block diagram)

A master slave flip flop contains two clocked flip flops. The first is called master and the second slave. When the clock is high the master is active. The output of the master is set or reset according to the state of the input. As the slave is inactive during this period its output remains in the previous state. When clock becomes low the output of the slave flip flop changes because it becomes active during low clock period. The final output of master slave flip flop is the output of the slave flip flop. So the output of master slave flip flop is available at the end of a clock pulse.

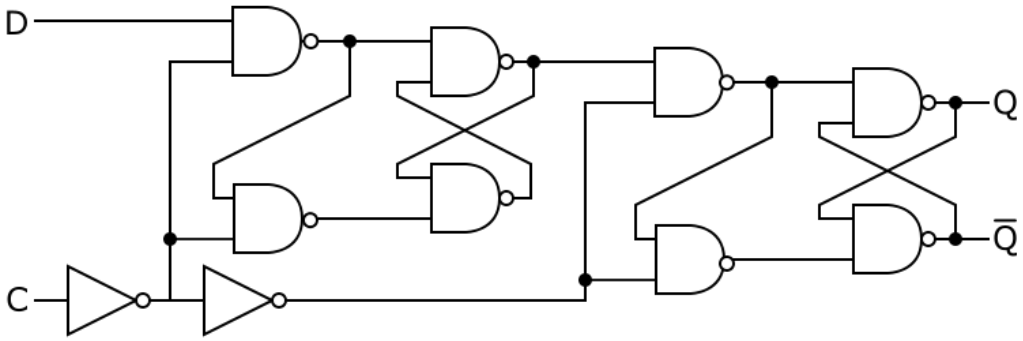


Figure 8(a): Master Slave Flip Flop

5.8 555 IC Timer

The 555 is the most popular integrated circuit (chip) introduced in 1971 by American company Signetics.The 555 timer IC is used in a variety of timer, pulse generation, and oscillator applications. The 555 can also be used to provide time delays, as an oscillator, and as a flip-flop element. Derivatives provide up to four timing circuits in one package.The 555 is still in widespread use due to its low price, ease of use, and stability. It is now made in the original bipolar and also in low-power CMOS types. The standard 555 package includes 25 transistors, 2 diodes and 15 resistors on a silicon chip installed in an 8-pin mini dual-in-line package

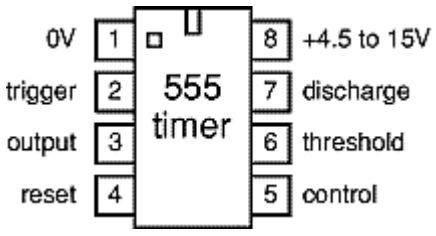


Figure 5.9: 555 PIN Arrangements

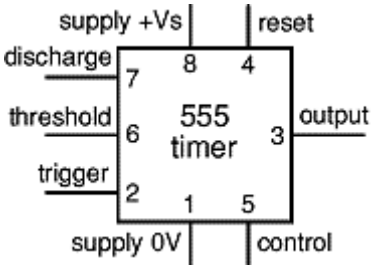


Figure 5.10: 555 Circuit Symbol

The circuit symbol pins are arranged to suit the circuit: for example pin 8 at the top for the +Vs supply, pin 3 output on the right. Usually just the pin numbers are used and they are not labeled with their function.

The connection of the pins for a DIP package is as follows:

Pin 1: GND⟹ Ground reference voltage, low level (0 V)

Pin 2: TRIG⟹ The OUT pin goes high and a timing interval starts when this input falls below 1/2 of CTRL voltage (which is typically $1/3V_{CC}$, CTRL being $2/3\ V_{CC}$ by default if CTRL is left open). More simply we can say that OUT will be high as long as the trigger is kept at low voltage. Output of the timer totally depends upon the amplitude of the external trigger voltage applied to this pin

Pin 3: OUT⟹ This output is driven to approximately 1.7 V below $+V_{CC}$, or to GND.

Pin 4: RESET⟹ A timing interval may be reset by driving this input to GND, but the timing does not begin again until RESET rises above approximately 0.7 volts. Overrides TRIG which overrides THR

Pin 5: CTRL⟹ Provides "control" access to the internal voltage divider (by default, $2/3\ V_{CC}$).Pin 5 is also sometimes called the CONTROL VOLTAGE pin. By applying a voltage to the CONTROL VOLTAGE input one can alter the timing characteristics of the device. In most applications, the CONTROL VOLTAGE input is not used. It is usual to connect a 10 nF capacitor between pin 5 and 0 V to prevent interference. The CONTROL VOLTAGE input can be used to build an astablemultivibrator with a frequency-modulated output.

Pin 6: THR⟹ The timing (OUT high) interval ends when the voltage at THR ("threshold") is greater than that at CTRL ($2/3\ V_{CC}$ if CTRL is open).

Pin 7: DIS⟹ Open collector output which may discharge a capacitor between intervals. In phase with output.

Pin 8: V_{CC}⟹ Positive supply voltage, which is usually between 3 and 15 V depending on the variation.

The IC 555 has three operating modes:

Bistable mode or Schmitt trigger – The 555 IC can operate as a flip-flop, if the DIS pin is not connected and no capacitor is used. Uses include bounce-free latched switches.

Monostablemode –In this mode, the 555 IC functions as a "one-shot" pulse generator. Applications include timers, missing pulse detection, bounce-free switches, touch switches, frequency divider, capacitance measurement, pulse-width modulation (PWM) etc.

Astable (free-running) mode –The 555 IC can operate as an electronic oscillator. Uses include LED and lamp flashers, pulse generation, logic clocks, tone generation, security alarms, pulse position modulation etc. The 555 IC can be used as a simple ADC, converting an analog value to a pulse length (e.g., selecting a thermistor as timing resistor allows the use of the 555 IC in a temperature sensor and the period of the output pulse is determined by the temperature). The use of a microprocessor-based circuit can then convert the pulse period to temperature, linearize it and even provide calibration means.

5.9 555 AstableMultivibrator

The 555 timer IC can be used with a few simple components to build an astable circuit which produces a 'square wave'. This is a digital waveform with sharp transitions between low (0V) and high (+Vs), the durations of the low and high states may be different. The circuit is called an astable because it is not stable in any state: the output is continually changing between 'low' and 'high'.

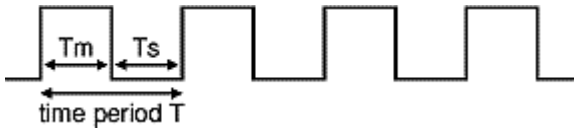


Figure 5.11: 555 astable output, a square wave (Tm and Ts may be different)

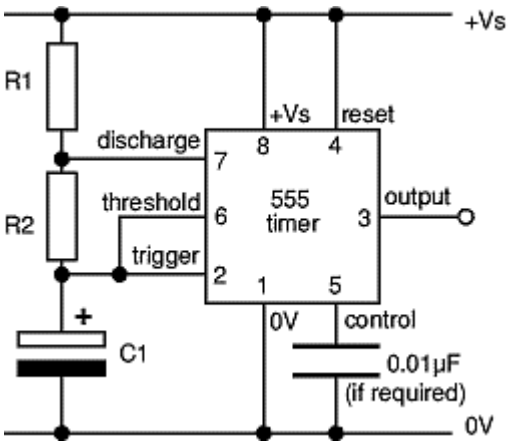


Figure 5.12: 555 AstableMultivibratorcircuit

Time period and frequency

The time period (T) of the square wave is the time for one complete cycle, but it is often better to consider frequency (f) which is the number of cycles per second.

$$T = 0.7 \times (R1 + 2R2) \times C1$$

$$f = \frac{1.4}{(R1 + 2R2) \times C1}$$

T = time period in seconds (s)

f = frequency in hertz (Hz)

$R1$ = resistance in ohms (ohm)

$R2$ = resistance in ohms (ohm)

$C1$ = capacitance in farads (F)

Mark and Space times:

The time period can be split into two parts:

Time period, $T = Tm + Ts$

Mark time (output high), $Tm = 0.7 \times (R1 + R2) \times C1$

Space time (output low), $Ts = 0.7 \times R2 \times C1$

It is important to note that Tm must be greater than Ts since $R1$ cannot be $0\ ohm$ (the minimum is $1\ Kohm$).Many circuits require $Tm\ and\ Ts$ being about equal. This is achieved if $R2$ is much larger than $R1$.

Choosing $R1, R2\ and\ C1$

$R1\ and\ R2$should be in the range $1kohm\ to\ 1Mohm$. It is best to choose $C1$ first because capacitors are available in just a few values.Choose $C1$ to suit the frequency range you require (use the table as a guide).

555 Astable frequencies			
C1	R2 - 10k R1 - 1k	R2 - 100k R1 - 10k	R2 - 1M R1 - 100k
0.001µF	68kHz	6.8kHz	680Hz
0.01µF	6.8kHz	680Hz	68Hz
0.1µF	680Hz	68Hz	6.8Hz
1µF	68Hz	6.8Hz	0.68Hz
10µF	6.8Hz	0.68Hz (41 per min.)	0.068Hz (4 per min.)

Choose $R2$ to give the frequency (f) you require. Assume that R1 is much smaller than R2 (so that Tm and Ts are almost equal), then you can use:

If $R1 << R2$ use

$$R2 = \frac{0.7}{f \times C1}$$

Choose R1 to be about a tenth of $R2$ (the minimum is 1kohm) unless you want the mark time Tm to be significantly longer than the space time Ts.If you need a variable resistor it is best to make it R2. Beware that if R1 is variable it must have a fixed resistor of at least 1kohm in series (this is not required for R2)

Duty cycle

The duty cycle of an astable circuit is the proportion of the complete cycle for which the output is high (the mark time). It is usually given as a percentage. For a standard 555 astable circuit the mark time (Tm) must be greater than the space time (Ts), so the duty cycle must be at least 50%:

$$Duty\ cycle = \frac{Tm}{\mathrm{Tm} + \mathrm{Ts}} = \frac{R1 + R2}{\mathrm{R1} + 2\mathrm{R2}}$$

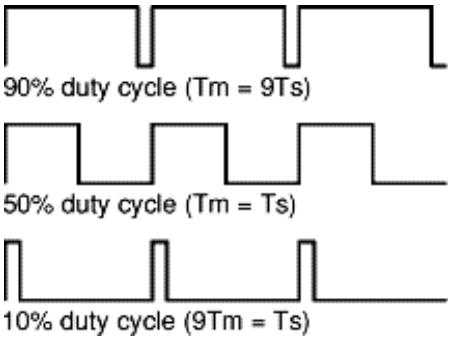


Figure 5.13: Duty Cycle

Duty cycle of less than 50%

To achieve a duty cycle of less than 50% a signal diode (such as 1N4148) can be added in parallel with R2 as shown in the diagram. This bypasses R2 during the charging (mark) part of the cycle so that Tm depends only on R1 and C1:

555 astable with diode (for duty cycle < 50%)

Tm = 0.7 × R1 × C1 (ignoring 0.7V across diode)

Ts = 0.7 × R2 × C1 (unchanged)

T = Tm + Ts = 0.7 × (R1 + R2) × C1

$$Duty\ cycle\ with\ diode\ = \frac{Tm}{\mathrm{Tm}\ +\ \mathrm{Ts}} = \frac{R1}{\mathrm{R1}\ +\ \mathrm{R2}}$$

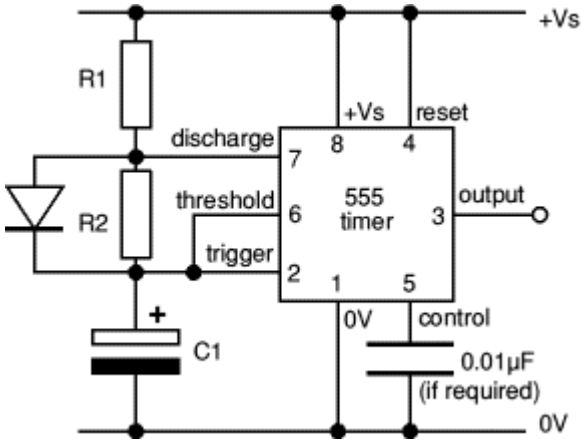


Figure 5.14: 555 Astablecircuit with diode across R2 Tm can be less than Ts so the duty cycle can be less than 50%

AstableMultivibrator Operation

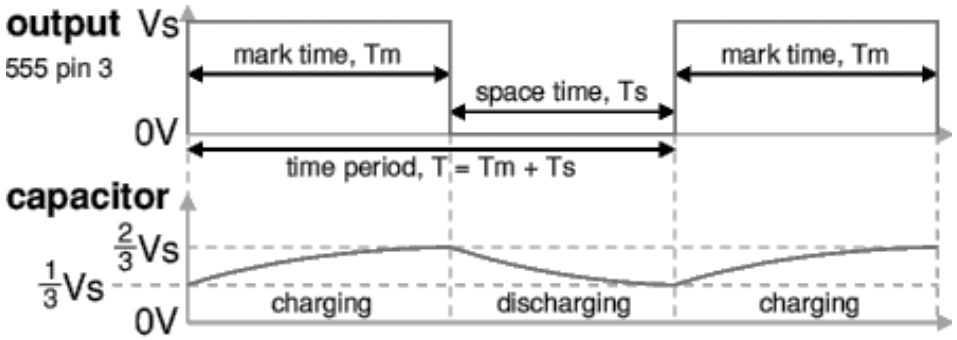


Figure 5.15: 555 AstableMultivibratorOperations

With the output high (+Vs) the capacitor C1 is charged by current flowing through R1 and R2. The threshold and trigger inputs monitor the capacitor voltage and when it reaches $^2/_3$Vs (threshold voltage) the output becomes low and the discharge pin is connected to 0V.

The capacitor now discharges with current flowing through R2 into the discharge pin. When the voltage falls to $^1/_3$Vs (trigger voltage) the output becomes high again and the discharge pin is disconnected, allowing the capacitor to start charging again.

This cycle repeats continuously unless the reset input is connected to 0V which forces the output low while reset is 0V.

An astable can be used to provide the clock signal for circuits such as counters.

A low frequency astable (< 10Hz) can be used to flash an LED on and off, higher frequency flashes are too fast to be seen clearly. Driving a loudspeaker or piezo transducer with a low frequency of less than 20Hz will produce a series of 'clicks' (one for each low/high transition) and this can be used to make a simple metronome.

An audio frequency astable (20Hz to 20kHz) can be used to produce a sound from a loudspeaker or piezo transducer. The sound is suitable for buzzes and beeps. The natural (resonant) frequency of most piezo transducers is about 3 kHz and this will make them produce a particularly loud sound.

5.10 555 MonoStableMultivibrator

The 555 timer IC can be used with a few simple components to build a monostable circuit which produces a single output pulse when triggered. It is called a *mono*stable because it is stable in just *one* state: 'output low'. The 'output high' state is temporary.

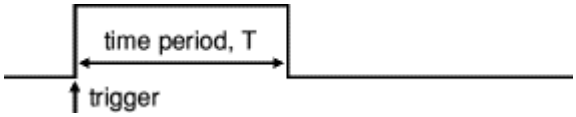


Figure 5.16: 555 mono stable output, a single pulse

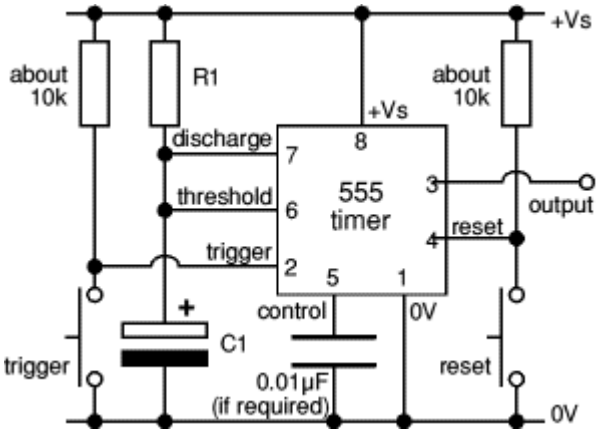


Figure 5.17: 555 monostablecircuit with manual trigger

Monostable Time Period

The duration of the pulse is called the time period (T) and this is determined by resistor R1 and capacitor C1:

$$Time\ period, T\ =\ 1.1\ \times\ R1\ \times\ C1$$

T = time period in seconds (s)

R1 = resistance in ohms (ohm)

C1 = capacitance in farads (F)

The maximum reliable time period is about 10 minutes.

The constant 1.1 is added because the capacitor charges to $2/3\ =\ 67\%$so it is a bit longer than the time constant (R1 × C1) which is the time taken to charge to 63%.

Choosing R1 and C1

Choose C1 first because there are relatively few values available. Choose R1 to give the required time period. R1 should be in the range 1kohm to 1Mohm, so use a fixed resistor of at least 1kohm in series if R1 is variable. The electrolytic capacitors do not have accurate values (errors of least 20% are common) and they tend to leak charge which increases the time period (especially if you are using a high value resistor).

MonostableMultivibrator Operation

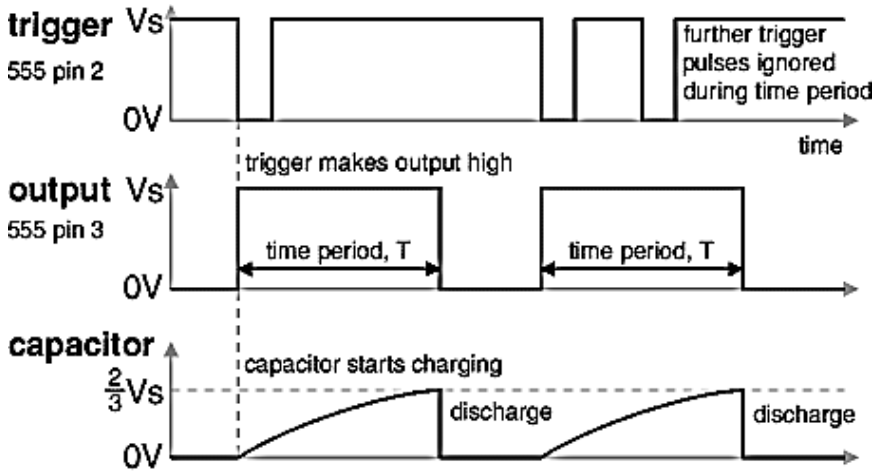


Figure 5.18: 555 monostable Operation

The timing period is triggered (started) when the trigger input (pin 2) is less than $^{1}/_{3}$ Vs, this makes the output high (+Vs) and the capacitor C1 starts to charge through resistor R1. Once the time period has started further trigger pulses are ignored.

The threshold input (pin 6) monitors the voltage across C1 and when this reaches $^{2}/_{3}$ Vs the time period is over and the output becomes low. At the same time discharge (pin 7) is connected internally to 0V, discharging the capacitor ready for the next trigger.

The reset input (pin 4) overrides all other inputs and the timing may be cancelled at any time by connecting reset to 0V, this instantly makes the output low and discharges the capacitor. If the reset function is not required the reset pin should be connected to +Vs directly with wire or with a resistor of about 10k .

It may be useful if a monostable circuit is reset or triggered automatically when the power supply is connected or switched on. This is achieved by using a capacitor instead of (or in addition to) a push switch as shown in the figure.

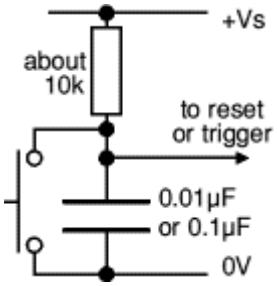


Figure5.19 : For Automatic Trigger