MCA I

Computer Organization and Architecture

TMC 102

By

Jaishankar Bhatt

Assistant Professor

Graphic Era Deemed to be University

Dehradun

Unit 1

Topic Name – Number System

Table of Contents

- Structure of Syllabus

- Number System

- Decimal Number System

- Binary Number System

- Octal Number System

- Hexadecimal Number System

- References

Structure of the Syllabus

The Whole syllabus of Computer Organization and Architecture can be divided in two parts. The first part covers Digital Electronics principle like number systems, logic gates, K map, various combinational and sequential circuits etc. This all is covered in unit 1.

The second part of the syllabus completely focuses on the computer organization and architecture of a basic computer. It covers basic micro operations, instruction cycle, instruction formats , addressing modes, I/O organization, memory systems and parallel processing concepts. All these concepts are divided into unit 2, 3, 4 and 5.

Number Systems: -

The technique to represent and work with numbers is called number system. Decimal number system is the most common number system. Other popular number systems include binary number system, octal number system and hexadecimal number system etc.

Base of a number system: - A number base indicates how many different digits are available when using a particular numbering system. Base is also known as Radix. For example, decimal is number base 10, which means it uses ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

We can represent a any number system like as **(Number)$_{Base}$**

For example a Decimal number can be represented as $(458)_{10}$, $(7582)_{10}$ etc.

Types of Number system: -

Commonly used number systems are Decimal, Binary, Octal and Hexadecimal.

1 Decimal number system (Base 10 number system): - Decimal number system has base 10 because it uses ten digits from 0 to 9. This means that any numerical quantity can be represented using these 10 digits.

Examples: -$(2468)_{10}$, $(243)_{10}$, $(1024)_{10}$, $(0024)_{10}$, etc.

Decimal numbers obey the positional weight principle. Each position of the number represents a specific weight.

Position	10^4	10^3	10^2	10^1	10^0
Positional Weight	10000	1000	100	10	1

2 Binary Number System (Base 2 number system): - Binary number is extensively used in digital electronics since it uses only two symbols 1 & 0. These two digits represent the two voltage levels 'high' or 'low' in digital electronics. The base (radix) of the binary number system is 2. All digital systems operate on binary number system.

Examples: - $(101)_2$, $(01101)_2$, $(110110)_2$, $(101001)_2$

Each binary digit is called a Bit. The right most bit is called Most Significant Bit (MSB) and the left most bit is called Least Significant Bit (LSB). Binary numbers obey the positional weight principle. Each position of the number represents a specific weight.

MSB
↓

Position	2^4	2^3	2^2	2^1	2^0	← LSB
Positional Weight	16	8	4	2	1	

3 Octal Number System (Base 8 number system): - In the Octal number system the base is 8 and it uses numbers from 0 to 7 (i.e. 0,1,2,3,4,5,6,7) to represent a numbers. Octal numbers are commonly used in computer applications.

Examples: - $(754)_8$, $(245)_8$, $(1345)_8$, $(726)_8$ etc

Octal numbers also obey the positional weight principle. Each position of the number represents a specific weight.

Position	8^4	8^3	8^2	8^1	8^0
Positional Weight	4096	512	64	8	1

4 Hexadecimal Number System (Base 16 number system): - In this number system total 16 numbers (i.e. 0 to 15) are used. The first 10 numbers (0 to 9) are used in numeric form and the last six numbers are used in alphabetic form (A to F). So the total 16 numbers are represented as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. A is used in place of 10, and B is used for 11 and so on.

$$10 \to A, 11 \to B, 12 \to C, 13 \to D, 14 \to E, 15 \to F$$

Examples: - $(45AF)_{16}$, $(9CE)_{16}$, $(7D4A)_{16}$, $(2BE)_{16}$ etc.

Position	16^4	16^3	16^2	16^1	16^0
Positional Weight	65536	4096	256	16	1

Check Your Self

1. The digital systems usually operate on _____ system.

a) Binary   b) decimal   c) octal   d) hexadecimal

2. The number 14 is represented in hexadecimal number system as.

a) A   b) D   c) F   d) E

3. After counting 0, 1, 10, 11, the next binary number is

a) 12   b) 100   c) 101   d) 110

4. The positional weight of most significant bit in binary number system is

a) 0   b) 2   c) 1   d) None

5. The base of the hexadecimal number system is

a) 8   b) 2   c) 16   d) 10

Unit 1

Topic Name – Conversion of Number System

Table of Contents

- Conversion of Number System

- Decimal to Binary

- Decimal to Octal

- Decimal to Hexadecimal

- Binary to Decimal

- Octal to Decimal

- Hexadecimal to Decimal

- References

Conversion of Number System

We can convert any number system to any other number system by following certain rules. Conversion can be done among the following number systems

1. Decimal Number System

2. Binary Number System

3. Octal Number System

4. Hexadecimal Number System

In a very simple way, we can perform six conversions very easily as follows.

1. **Decimal to any base** (i.e. Decimal to Binary, Decimal to Octal, Decimal to Hex)

2. **Any base to decimal** (i.e. Binary to Decimal, Octal to Decimal, Hex to Decimal)

In first case i.e. decimal to any base, same process or steps are followed for conversion. In second case i.e. any base to decimal, same procedure is followed for conversion only radix or base changes accordingly..

1. Decimal to any base (i.e. Decimal to Binary, Decimal to Octal, Decimal to Hex)

In this conversion, decimal is the source number system and target number system may be binary or octal or hexadecimal. In this conversion procedure a same technique is used for all three conversion.

In decimal to any base conversion system we will divide the decimal number (source number system) by the base of the target number system and the remainder will give the target number system.

For example in decimal to binary conversion, source number system is decimal and target number system is binary. Decimal number is divided by 2 repeatedly, until the remainder is zero. Every time the decimal number is divided by 2 the reminder is note down in right side. Whenever a number is divided by two, the reminder may be 0 or 1 not anything else. The remainder in the reverse order, give the binary equivalent of the given decimal number.

Example 1: - Convert decimal number $(65)_{10}$ into binary number.

OR

$(65)_{10} \longrightarrow (?)_2$

Solution: -

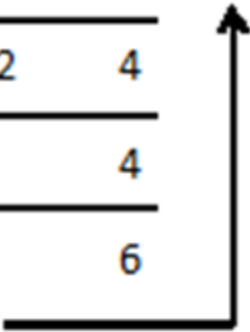| 2	65	
2	32	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

$(65)_{10} \longrightarrow (1000001)_2$

Example 2: - Convert decimal number $(2980)_{10}$ into Octal number.

OR

$(2980)_{10}$ $\longrightarrow$ $(?)_8$

Solution: -

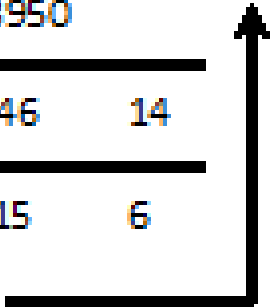| 8	2980	
8	372	4
8	46	4
	5	6

$(2980)_{10}$ $\longrightarrow$ $(5644)_8$

Example 3: - Convert decimal number $(3950)_{10}$ into hexadecimal number.

OR

$(3950)_{10}$ ⟶ $(?)_{16}$

Solution: -

16	3950	
16	246	14
	15	6

15 is represented as F and 14 is represented as E in hexadecimal.

So $(3950)_{10}$ ⟶ $(F6E)_{16}$

2. Any base to decimal (i.e. Binary to Decimal, Octal to Decimal, Hex to Decimal)

In this type of conversion source number system may be binary or octal or hexadecimal and the target number system is decimal. The procedure of conversion is same only base changes accordingly.

In a any base to decimal conversion, decimal number is sum of all digits multiplied individually by their positional weight.

Binary to decimal conversion: - d is the binary digit and $d_{n-1} ... d_3 d_2 d_1 d_0$ is a group of binary digits.

Decimal number = $d_{n-1} \times 2^{n-1} .... + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$

Example: - $111001_2 \longrightarrow (?)_{10}$

$= (111001)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 32 + 16 + 8 + 0 + 0 + 1$

$= (57)_{10}$

Octal to decimal conversion: - d is the octal digit. $d_{n-1} \ldots d_3\, d_2\, d_1\, d_0$ is a group of octal digits.

Decimal number = $d_{n-1} \times 8^{n-1} \ldots\ldots\ldots + d_2 \times 8^2 + d_1 \times 8^1 + d_0 \times 8^0$

Example 1: - ($2754)_8 \longrightarrow (?)_{10}$

Solution: -

$$2\ 7\ 5\ 4 = 2 \times 8^3 + 7 \times 8^2 + 5 \times 8^1 + 4 \times 8^0$$

$$= 1024\ +\ 448\ + 40\ \ \ \ + 4$$

$$= (1516)_{10}$$

Example 1: - ($1034)_8 \longrightarrow (?)_{10}$

$$1034 = 1 \times 8^3 + 0 \times 8^2 + 3 \times 8^1 + 4 \times 8^0$$

$$= 512\ +\ 0\ + 24\ \ \ \ + 4$$

$$= (540)_{10}$$

Hexadecimal to decimal conversion: - d is the hexadecimal digit and $d_{n-1} \ldots d_3 d_2 d_1 d_0$ is a group of hexadecimal digits.

Decimal number = $d_{n-1} \times 16^{n-1} \ldots \ldots \ldots + d_2 \times 16^2 + d_1 \times 16^1 + d_0 \times 16^0$

Example 1: - $(2A5)_{16} \longrightarrow (?)_{10}$

Solution: -

$$2A5 = 2 \times 16^2 + 10 \times 16^1 + 5 \times 16^0$$

$$= 512 + 160 + 5$$

$$= (677)_{10}$$

Example 2: - $(A37E)_{16} \longrightarrow (?)_{10}$

Solution: -

$$A37E = A \times 16^3 + 3 \times 16^2 + 7 \times 16^1 + E \times 16^0$$

$$= 10 \times 16^3 + 3 \times 16^2 + 7 \times 16^1 + 14 \times 16^0$$

$$= 40960 + 768 + 112 + 14$$

$$= (41854)_{10}$$

Check your Progress: - Convert the following number systems into decimal number.

(1) ($10110)_2$ $\longrightarrow$ $(?)_{10}$

(2) ($011100)_2$ $\longrightarrow$ $(?)_{10}$

(3) ($1100)_2$ $\longrightarrow$ $(?)_{10}$

(4) ($6548)_8$ $\longrightarrow$ $(?)_{10}$

(5) ($2617)_8$ $\longrightarrow$ $(?)_{10}$

(6) ($547)_8$ $\longrightarrow$ $(?)_{10}$

(7) ($A2B)_{16}$ $\longrightarrow$ $(?)_{10}$

(8) ($45E)_{16}$ $\longrightarrow$ $(?)_{10}$

(9) ($82C)_{16}$ $\longrightarrow$ $(?)_{10}$

Unit 1

Topic Name – Number System Conversion Part 2

Table of Contents

- Octal to Binary Conversion

- Binary to Octal Conversion

- Hexadecimal to Binary Conversion

- Binary to Hexadecimal Conversion

- Octal to Hexadecimal Conversion

- Hexadecimal to Octal Conversion

- References

Decimal Equivalent of Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	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. Octal to Binary Conversion: - The base or (radix) of the octal number system is 8. This number system uses 8 different symbols 0,1,2,3,4,5,6,7. Each significant position in an octal no. system has a positional weight. These weights are given in the ascending powers of 8, i.e. 8^0, 8^1, 8^2,....etc. respectively.

The binary equivalent of an octal no. must be a group of 3 bits, since the base of octal no. system is $8 = 2^3$.

<u>Octal numbers can be converted into equivalent binary numbers by replacing each octal digit by its 3 bit equivalent binary.</u>

Example 1: convert $(3702)_8$ to a binary number.

$(3702)_8 =$ 3   7   0   2

                011   111   000   010

The number $(3702)_8 = (011111000010)_2$

                $= (11111000010)_2$

Example 2: convert $(456)_8$ to a binary number.

$(456)_8 =$ 4   5   6

               100   101   110

The number $(456)_8 = (100\ 101\ 110)_2$

Example 3: Convert $(103)_8$ to a binary number.

$(103)_8 =$ 1   0   3

               001   000   011

The number $(103)_8 = (001\ 000\ 011)_2$

Example 4: convert $(6352)_8$ to a binary number.

$(6352)_8 =$ 6   3   5   2

               110   011   101   010

The number $(6352)_8 = (110\ 011101010)_2$

2. Binary to Octal Conversion: - To convert from binary to octal, divide the whole binary string into groups of 3 bits. Then starting from LSB and moving towards MSB, convert each group of 3 bits into the corresponding octal digit.

Example 1: -

Convert $(10101011111)_2$ to an octal number.

Solution: **0**10   101   011   111

                              2   5   3   7

Therefore $(10101011111)_2 = (2537)_8$

Example 2: - Convert $(11011101010110)_2$ to an octal number.

Solution: **0**11   011   101   010   110

                              3   3   5   2   6

Therefore $(11011101010110)_2 = (33526)_8$

3. Hexadecimal to Binary Conversion: - The base of hexadecimal number system is 16 and it uses 16 symbols, namely, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The symbol A to F represents the decimal 10 to 15 respectively.

Since its base is 16= 2^4, every 4 bit binary digit combination can be represented by one hexadecimal digit. Each significant position in an hexadecimal number has a positional weight. These weights are given in the ascending powers of 16. i.e., 16^0, 16^1, 16^2, …. etc.

In Hexadecimal to Binary Conversion, each hexadecimal digit is converted to its four bit binary equivalent. For example

Example 1: - Convert $(2DAC)^{16}$ to Binary number system

$$(2DAC)_{16} = \qquad 2 \qquad D \qquad A \qquad C$$
$$\qquad\qquad 0010 \qquad 1101 \qquad 1010 \qquad 1100$$
$$(2DAC)_{16} = (10110110101100)_2$$

Example 2: - Convert $(7A5F)_{16}$ to Binary number system

$(7A5F)_{16} =$ 7   A   5   F

             0111   1010   0101   1111

$(7A5F)_{16} = (011110100101\ 1111\ )_2$

4. Binary to hexadecimal Conversion: To convert from binary to Hexadecimal, divide the whole binary string into groups of 4 bits. Then starting from right and moving towards left, convert each group of 4 bits into the corresponding Hexadecimal digit.

Example 1: - Convert $(011110100101\ 1111\ )_2$ to Hexadecimal number system

$(0111101001011111\ )_2 =$ 0111   1010   0101   1111

                    7   A   5   F

$(011110100101\ 1111\ )_2 = (7A5F)_{16}$

Example 1: - Convert $(1001110100100110\ )_2$ to Hexadecimal number system

$(1001110111000110)_2 =$ 1001   1101   1100   0110

                  9   D   C   6

$(1001110111000110\ )_2 = (9DC6)_{16}$

5. Octal to Hexadecimal Conversion: - To convert an octal no. to hexadecimal the easiest method is to first convert the octal number to its 3 bit binary equivalent and then the binary number to hexadecimal.

Example 1. : - Convert $(756)_8$ to hexadecimal number

$$(756)_8 = 7 \qquad\qquad 5 \qquad\qquad 6$$

$$\qquad\quad = \;\; 111 \qquad 101 \qquad 110$$

Now we have a binary number 111101110. Make groups of 4 binary bits from right side.

$$(756)_8 = \;\; 0001 \qquad 1110 \qquad 1110$$

$$\qquad\quad = \;\; 1 \qquad\qquad E \qquad\qquad E$$

$$(756)_8 \;\; = (1EE)_{16}$$

6. Hexadecimal to Octal Conversion: - To convert a hexadecimal number to octal, first write the 4 bit binary equivalent of each hexadecimal digit, then make group of 3 binary bits from LSB to MSB and write the octal equivalent of each binary group.

For Example: Convert $(A6F)_{16}$ to Octal no.

$$(A6F)_{16} = A \qquad\qquad 6 \qquad\qquad F$$

$$= 1010 \qquad\qquad 0110 \qquad 1111$$

Now we have $(101001101111)_2$. For octal conversion, make groups of 3 bits from right side.

$$= 101 \qquad\quad 001 \qquad 101 \qquad 111$$

$$= \ 5 \qquad\qquad 1 \qquad\quad 5 \qquad\quad 7$$

$$= (5157)_8$$

$$(A6F)_{16} = (5157)_8$$

Unit 1

Topic Name – Codes

Table of Contents

- Codes Definition

- ASCII Code

- EBDIC CODE

- ASCII VS EBCDIC

- BCD Code

- Access 3 Code

- Gray Code

- References

Codes: - Various binary codes are used to represent the data. A code is a symbolic representation of discrete information, which may be present in the form of numbers or letters. The code can be classified into two types of codes, i.e. numeric codes and alphanumeric codes. Alphanumeric codes represent alphanumeric information, i.e. letters of the alphabet, and decimal numbers. An example of alphanumeric code is the ASCII code (American Standard Code for Information Interchange). Numeric code represents numeric information, i.e. only numbers as a series of 0s and 1s. An example of the numeric code is the Binary coded decimal (BCD) code which is used to represent decimal digits.

Codes are also used for error detection and error correction purpose. Since when digital signals are transmitted from one location to another location, errors may occur due to electrical noise.

ASCII code: - ASCII, stands for American Standard Code for Information Interchange. It's a 7-bit character code where every single bit represents a unique character.

American Standard Code for Information Interchange, is a character encoding standard for electronic communication. ASCII is made up of 128 symbols in the character set. These symbols consist of letters (both uppercase and lowercase), numbers, punctuation marks, special characters and control characters. Each symbol in the character set can be represented by a Decimal value ranging from 0 to 127, as well as equivalent Hexadecimal and Octal values.

EBCDIC code: - Extended binary coded decimal interchange code (EBCDIC) is an 8-bit binary code for numeric and alphanumeric characters, it was developed by IBM, that uses a unique eight-bit binary code for each number and alphabetic character as well as punctuation marks and accented letters and non-alphabetic characters. It was developed and used by IBM. It is a coding representation in which symbols, letters and numbers are presented in binary language.

The **main difference** between ASCII and EBCDIC is that the **ASCII uses seven bits to represent a character while the EBCDIC uses eight bits to represent a character.**

Difference Between ASCII and EBCDIC

Definition

ASCII is a character encoding standard for electronic communication. EBCDIC is an eight-bit character encoding used mainly on IBM mainframe and IBM midrange computer operating systems.

Stands for

ASCII stands for American Standard Code for Information Interchange. EBCDIC stands for Extended Binary Coded Decimal Interchange Code.

Number of Characters

Also, ASCII represents 128 (2^7) characters while EBCDIC represents 256 (2^8) characters.

Efficiency

Moreover, the same character in ASCII requires 7 bits, but EBCDIC required 8 bits. Therefore, ASCII is more efficient than EBCDIC.

Number of bits Represent a Character

Further, ASCII uses 7 bits to represent a character. EBCDIC uses 8 bits to represent a character.

Compatibility

Additionally, ASCII is compatible with modern encodings such as Unicode. It is possible to open ASCII files with Unicode. On the other hand, EBCDIC is not compatible with modern encodings such as Unicode.

Conclusion

ASCII and EBCDIC are two character encoding standards. The main difference between ASCII and EBCDIC is that the ASCII uses seven bits to represent a character while the EBCDIC uses eight bits to represent a character.

BCD Code: - A Binary Coded Decimal (BCD) is one in which, decimal digits are encoded by their natural binary equivalents-one at a time-into groups of four bits. The disadvantage of BCD code is that it requires more no. of bits to code a decimal number. But the ease of conversion between the BCD code to decimal number and vice versa is the main advantage of this code, which makes it useful and popular code for input, output operations in digital systems. To convert any decimal number in to BCD ,each decimal digit should be replaced by the appropriate 4-bit code.

For Example

Convert $(3906)_{10}$ to BCD.

$\qquad$ $(3\ 9\ 0\ 6)_{10}$ = $\qquad$ 0011 $\qquad$ 1001 $\qquad$ 0000 $\qquad$ 0110

Therefore $(3906)_{10}$ = $(11100100000110)_2$

Note: Zeros are added to ensure that each digit is represented by four bits. The leading zeros can be suppressed on the most significant digit.

Excess -3 or XS-3 code : - <u>The excess -3 or XS-3 code is a non-weight code.</u> This means each position within a binary number isn't assigned affixed value. This code is a self complementing code, which means 1's complement of the coded number yields 9's complement of the number itself. As the name indicates, XS-3 code represents a decimal number, in binary form, as a number greater than 3.

An XS-3 code is obtained by adding 3 to a decimal number, then convert it into its equivalent BCD code.

Example 1: - Convert $(463)_{10}$ into its XS-3 code.

Sol. Given decimal number   4   6   3

 Add 3 to each digit of given number.   +3   +3   +3
                                        _____
                                          7   9   6

Converting the above sum into its BCD code,

we have Sum   7   9   6

          BCD   0111   1001   0110

Hence, the XS-3 code for $(463)_{10}$ is 0111 1001 0110

Example 2: - Convert $(574)_{10}$ into its XS-3 code.

Sol. Given decimal number   5   7   4

Add 3 to each digit of given number.   +3   +3   +3
                                      _____
                                       8   10   7

Converting the above sum into its BCD code,

we have Sum   8   10   7

        BCD   1000   1010   0111

Hence, the XS-3 code for $(574)_{10}$ is $(100010100111)_2$

Gray Code: - The gray code is not a weighted code and is not suitable for arithmetic operations. Gray code has some applications in analog to digital converters, as well as being used for error correction in digital communication. In computers, we need to convert binary to gray and gray to binary. The conversion of this can be done by using two rules namely binary to gray conversion and gray to binary conversion.

Example1 : Convert Binary number $(11101)_2$ into gray code.

Sol.
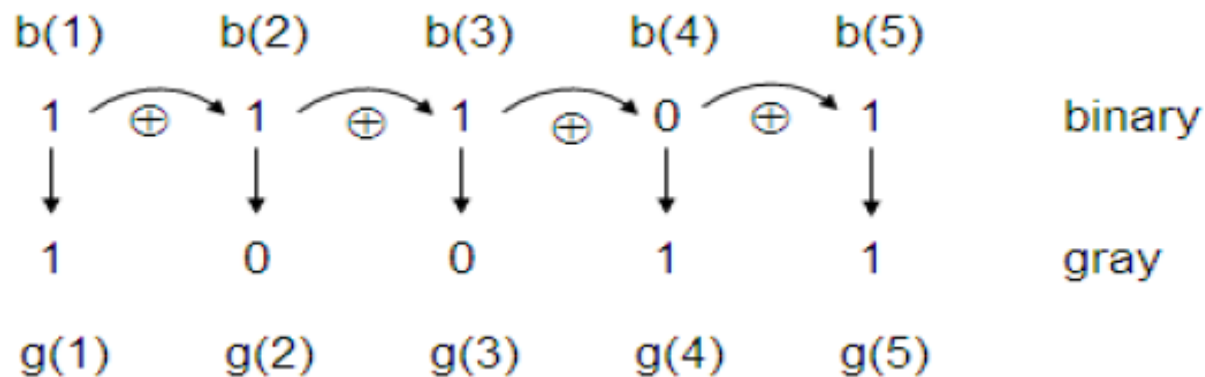
$g(1) = b(1),$ $\qquad\qquad$ $g(2) = b(1)$ XOR $b(2),$ $\qquad$ $g(3) = b(2)$ XOR $b(3),$

$g(4) = b(3)$ XOR $b(4)$ $\qquad$ $g(5) = b(4)$ XOR $b(5)$

Answer : $(111\,0\,1)_2 = (1\,0\,0\,1\,1)_{Gray}$

Example1 : Convert gray code $(10011)_{Gray}$ into binary code.

Sol: -
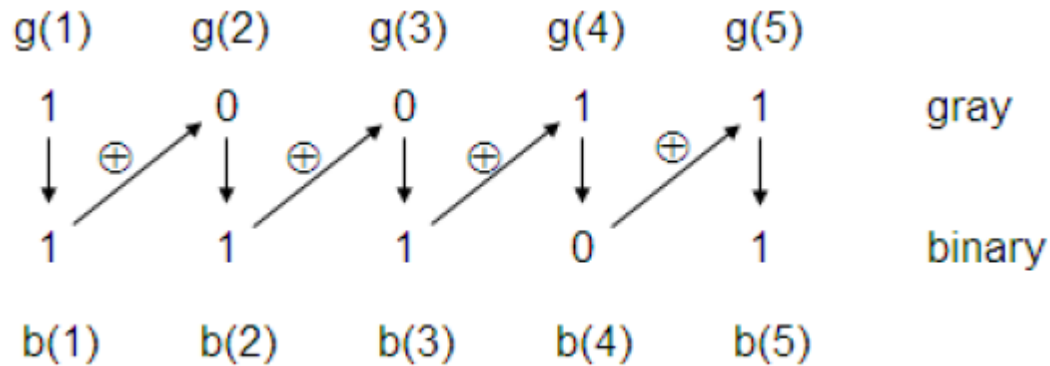
b(1) = g(1),   b(2) = b(1) XOR g(2),   b(3) = b(2) XOR g(3)

b(4) = b(3) XOR g(4),   b(5) = b(4) XOR g(5)

Answer: $(10011)_{Gray}$ = $(11101)_2$

Check your Progress –

Convert Following binary number into gray code

1. $(100110)_2 = (?)_{Gray}$

2. $(011001)_2 = (?)_{Gray}$

3. $(110011)_2 = (?)_{Gray}$

4. $(011001)_2 = (?)_{Gray}$

5. $(100101)_2 = (?)_{Gray}$

Convert Following gray codes into binary number.

1. $(110100)_{Gray} = (?)_2$

2. $(1110001)_{Gray} = (?)_2$

3. $(1000100)_{Gray} = (?)_2$

4. $(0011010)_{Gray} = (?)_2$

5. $(1010001)_{Gray} = (?)_2$

Unit 1

Topic Name – Complements

Table of Contents

- r's complements

- (r-1)'s complements

- 1's complements

- 2's complements

- Binary Addition

- Binary Subtraction using 2's Complements

- References

Complements: - Complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations. For each radix-r system (radix r represents base of number system) there are two types of complements.

S.N.	Complement	Description
1	Radix Complement	The radix complement is referred to as the r's complement. (10's complements and 2's complements)
2	Diminished Radix Complement	The diminished radix complement is referred to as the (r-1)'s complement. (9's complements and 1's complements)

1. r's complements or radix complements: - If N is a number to base r with integer part of n digits, the r's complements of N is defined as $r^n - N$.

Where r = Base of the number.

n = number of integer digits

N = Given positive number

Example1: - Find the 10's complements of decimal number$(5252)_{10}$

Solution: - $(5252)_{10} = 10^4 - 5252$

$= 10000 - 5252$

$= 4748$

Example2: - Find the 10's complements of decimal number$(0.5373)_{10}$

Solution: - $(0.5373)_{10} = 10^0 - 0.5373$

$= 1 - 0.5373$

$= 0.4627$

Example3: - Find the 10's complements of decimal number$(1423)_{10}$

Solution: - $(1423)_{10} = 10^4 - 1423$

$$= 10000 - 1423$$

$$= 8577$$

2. (r-1)'s complements or radix complements: - If N is a number to base r with integer part of n digits and m fraction digits, the (r-1)'s complements is defined as $[(r^n - 1) - N]$ or $(r^n - r^{-m} - N)$.

Where r = Base of the number.

  n = number of integer digits

  m = number of fraction digits

  N = Given positive number

Example 1: - Find the 9's complements of $(5252)_{10}$

$(5252)10 = 10^4 - 1 - 5252$ $[(r^n - 1) - N]$

  $= 9999 - 5252$

  $= 4747$

Example 2: - Find the 9's complements of $(0.5373)_{10}$

$(0.5373)_{10}$ $= 10^0 - 10^{-4} - 0.5373$ $[(r^n - 1) -N]$

$= 1 - 0.0001 - 0.5373$

$= 0.9999 - 0.5373$

$= 0.4626$

1's and 2's Complement: - There are two complement forms used in digital systems to represent signed numbers. These are 1's complement and 2's complement form. These two forms are used to represent negative numbers. Most digital computers do subtraction by the 2's complement method. <u>The advantage of performing subtraction by the complement method is reduction in the hardware. Instead of having separate digital circuits for addition and subtraction, only adding circuits are needed .i.e. subtraction is also performed by adders only.</u>

1's Complement: - To take the 1's complement of a binary no., simply change all 1's to 0s and all 0's to 1s .

For Example

The 1's complement of 110010 is 001101

2's Complement: - Similarly the 2's complement of a binary no. can be obtained by adding 1 to its 1's complement **.**

Example: - Find the 2's complement of $(10110)_2$

Solution: - First find the 1's complement of the given binary number.

1's complement of 10110 = 01001

For obtaining 2's complement of 10110, add 1 to 01001 (1's complement of 10110)

i.e, 01001 + 1= 01010

Binary Addition: - Binary addition is performed with the two binary bits. The outcome of a binary addition is Sum (S) and carry (C). The binary addition between two bits A and B is performed according to the following truth table.

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

Example1: - Add binary number $(1010)_2$ and $(1010)_2$

Solution:-

		1		1	
1	1	0	1	0	
	1	0	1	0	
Sum **=**	0	1	0	0	

Example1: - Add binary number $(1110)_2$ and $(0011)_2$

Solution:-

		1	1		
	1	1	1	1	0
		0	0	1	1
Sum =		0	0	0	1

Example3: - Add binary number $(1110)_2$ and $(0111)_2$

Solution:-

		1	1		
	1	1	1	1	0
		0	1	1	1
Sum =		0	1	0	1

Binary Subtraction using 2's complements: - Binary subtraction can be done with the help of complements. Subtraction using 2's complements can be done as follows.

- Take the 2's complement of the subtrahend.
- Add it to the minuend.

Example 1: - Subtract $(0111)_2 - (0101)_2$

Step 1: Find the 1's complement of subtrahend (i.e. 0101)

      1's complement of 0101 = 1010

         Add binary 1 in 1010 = 1 0 1 0

$$\underline{\qquad\qquad + \ 1\qquad}$$

                         1 0 1 1   (Add this number with minuend)

Step2 : -   1 1   1   1   (Carry generated)

          0 1 1 1   (Decimal 7)

          $\underline{1 \ \ 0 \ \ 1 \ \ 1}$ (Decimal -5)

          0   0 1 0   (Decimal 2)

Example 2: - Subtract $(0111)_2 - (0100)_2$

Step 1: Find the 1's complement of subtrahend (i.e. 0100)

1's complement of 0100 = 1011

$$1 \quad 1 \qquad \text{(Carry generated)}$$

Add binary 1 in 1010 = 1   0   1   1

$$+ \ 1$$
_____

1   1   0   0   (Add this number with minuend)

Step2 : -

1   1   (Carry generated)

0   1   1   1   (Decimal 7)

1   1   0   0   (Decimal -4)
_____

0   0   1   1   (Decimal 3)

Topic Name – Booth's Algorithm

Table of Contents

- Shift Micro-operation

- Arithmetic Shift left

- Arithmetic Shift right

- Booth's Algorithm

- References

Arithmetic shift: -An arithmetic shift is a micro-operation that shifts a signed binary number to the left or right. Two types of arithmetic shift are as follows.

1. A **Left Arithmetic Shift** of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded. It is identical to Left Logical Shift.

2. A **Right Arithmetic Shift** of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with the value of the previous (now shifted one position to the right) MSB.

Left Arithmetic Shift

MSB   LSB

| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | ← 0

| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

Right Arithmetic Shift

MSB   LSB

| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
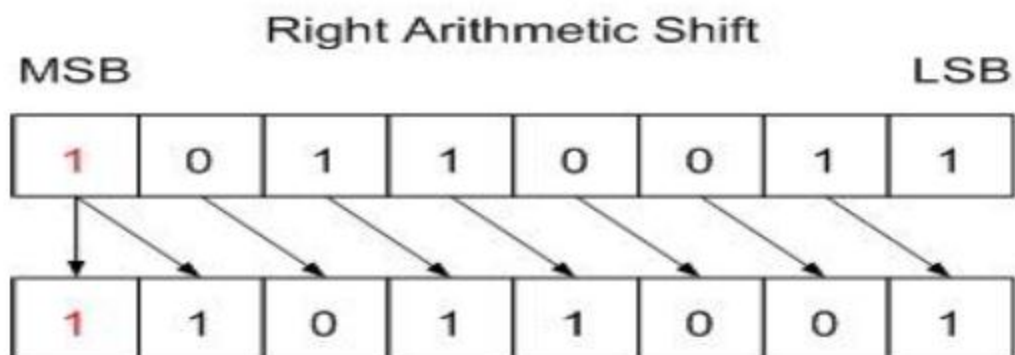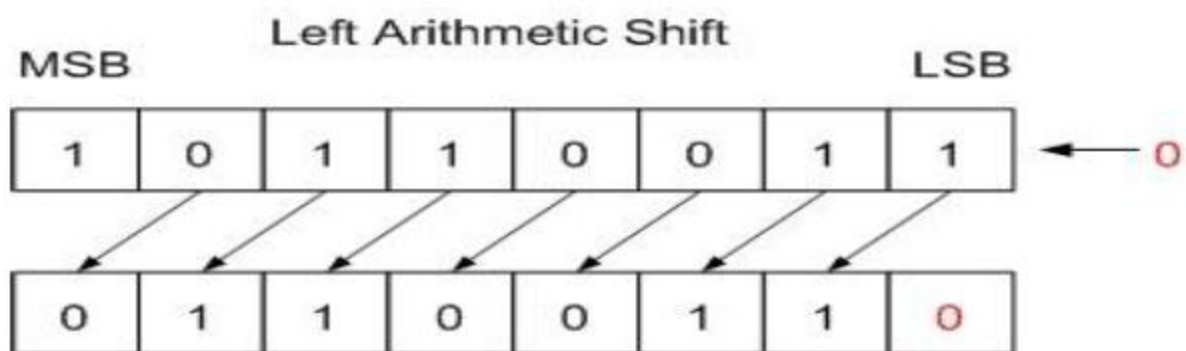
Booths Multiplication Algorithm:- The multiplier and multiplicand are placed in the Q and M registers, respectively. There is also a 1-bit register placed logically to the right of the least significant bit (Q_0) of the Q register and designated Q_1. The results of the multiplication will appear in the A and Q registers. A and Q_1 are initialized to 0. As before, control logic scans the bits of the multiplier one at a time. Now, as each bit is examined, the bit to its right is also examined. If the two bits are the same (1–1 or 0–0), then all of the bits of the A, Q, and Q_1 registers are shifted to the right 1 bit. If the two bits differ, then the multiplicand is added to or subtracted from the A register, depending on whether the two bits are 0–1 or 1–0. Following the addition or subtraction, the right shift occurs. In either case, the right shift is such that the leftmost bit of A, namely A_{n-1}, not only is shifted into A_{n-2}, but also remains in A_{n-1}. This is required to preserve the sign of the number in A and Q. It is known as an **arithmetic shift right**, because it preserves the sign bit.
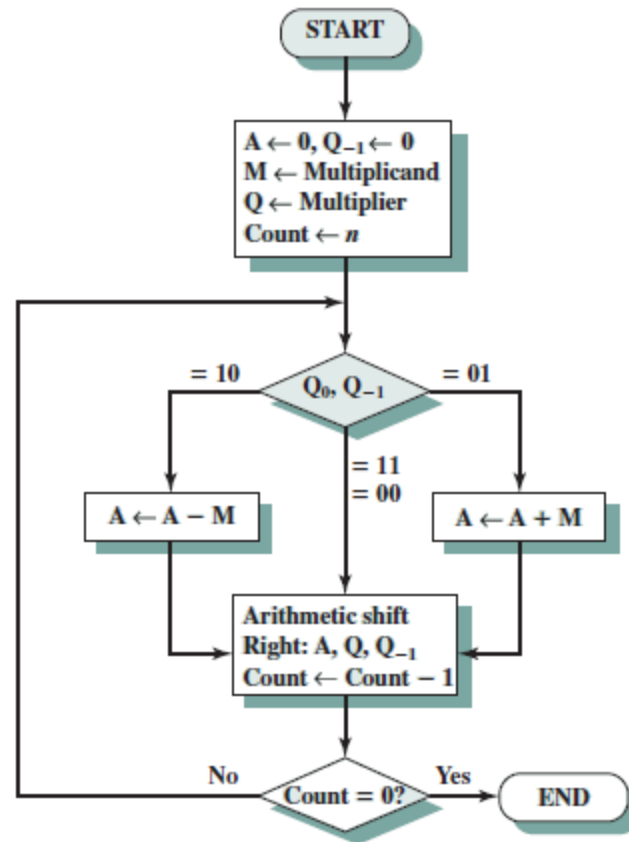
Flow Chart of Booth's multiplication algorithm

Example: Multiply -7 X -3 using Booth's algorithm.

Solution: Multiplicand = -7, First find the 2's complement of 7

$(7)_{10} = (0111)_2$

1's complement of 0111 = 1000
2's complement of 0111 = 1000

```
                    +   1
                  -----------
                     1001     (-7)
                  ------------
```

Multiplier = -3
$(3)_{10} = (0011)_2$

1's complement of 0111 = 1100
2's complement of 0111 = 1100

```
                    +   1
                  -----------
                     1101    (-3)
                  --------------
```

Place Multiplicand (-7) in register M, and multiplier (-3) in register Q.

Initialize register A and Q_1 to 0.

A	Q	Q_1	C	Steps
0000	1101	0	4	Initial Values
0111	1101	0	4	A ← A-M
0011	1110	1	3	Arithmetic Shift Right AQQ$_1$
1100	1110	1	3	A ← A+M
1110	0111	0	2	Arithmetic Shift Right AQQ$_1$
0101	0111	0	2	A ← A-M
0010	1011	1	1	Arithmetic Shift Right AQQ$_1$
1 0101 1		0	0	Arithmetic Shift Right AQQ$_1$

Answer is (-7X-3 = 21) = (00010101)$_2$ and (21) in decimal

Example: Multiply 10 X 10 using Booth's algorithm.

Solution: Multiplicand = Decimal 10

$(10)_{10} = (01010)_2$ In 5 bits

1's complement of 01010 = 10101
2's complement of 01010 = 10101
                              +   1

                            10110   (-10)

Place Multiplicand 10 (01010) in register M, and multiplier 10 (01010) in A register. Initialize register A and Q_1 to 0.

A	Q	Q_1	C	Steps
00000	01010	0	5	Initial Values
00000	00101	0	4	Arithmetic Shift Right AQQ_1
10110	00101	0	4	A<-A-M
11011	00010	1	3	Arithmetic Shift Right AQQ_1
00101	00010	1	3	A<-A+M
00010	10001	0	2	Arithmetic Shift Right AQQ_1
11000	10001	0	2	A<- A-M
11100	01000	1	1	Arithmetic Shift Right AQQ_1
00110	01000	1	1	A<- A+M
00011	00100	0	0	Arithmetic Shift Right AQQ_1

Answer is (10X10 = 100) = $(0001100100)_2$ and (100) in decimal

Unit 1

Topic Name – Logic Gate

Table of Contents

- Logic Gates

- AND Gate

- OR Gate

- NOT Gate

- Buffer

- NAND Gate

- NOR Gate

- XOR Gate

- XNOR Gate

- References

Logic Gates: - Binary information is represented in digital computers by physical quantities called signals. Electrical signals such as voltages exist throughout the computer in either one of two recognizable states. The two states represent a binary variable that can be equal to 1 or 0.

Binary logic deals with binary variables and with operations that assume a logical meaning. It is used to describe, in algebraic or tabular form, the manipulation and processing of binary information. The manipulation of binary information is done by logic circuits called gates. Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied. A variety of logic gates are commonly used in digital computer systems. Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression. The input-output relationship of the binary variables, for each gate, can be represented in tabular form by a truth table.

Various types of logic gates are as follows:

1. AND Gate

2. OR Gate

3. Inverter or NOT

4. Buffer

5. NAND Gate

6. NOR Gate

7. Exclusive - OR (XOR) Gate

8. Exclusive - NOR (XNOR) Gate

1. AND Logic Gate: - The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. (A.B)

Algebraic function of AND gate is X= (A . B) or X= (AB)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Input: A B — Output: X

Graphic Symbol

A
B — X

2. OR Logic Gate: - The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus sign (+) is used to show the OR operation.

Algebraic function of OR gate is X= A + B
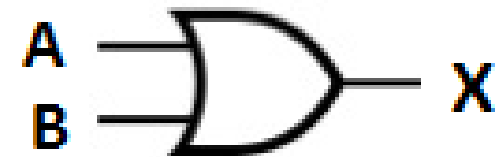
Input		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Graphic Symbol

Truth Table

3. Inverter or NOT Logic gate: - The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top.

Algebraic function of NOT gate is X= A'
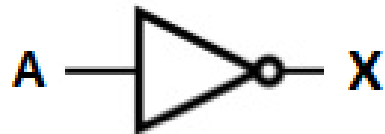
Graphic Symbol

Input A	Output X
0	1
1	0

Truth Table

4. Buffer: - A buffer, is a basic logic gate that passes its input, unchanged, to its output. The main purpose of a buffer is to regenerate the input, usually using a strong high and a strong low. A buffer has one input and one output; its output always equals its input.

Algebraic function of buffer gate is X= A

A ─▷─ X

Graphic Symbol
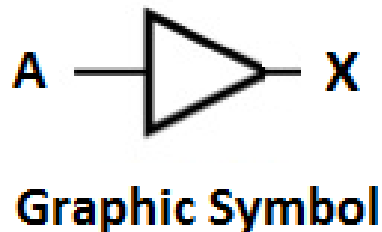
Input A	Output X
0	0
1	1

Truth Table

5. NAND Logic Gate: - NOT-AND gate is equal to an AND gate followed by a NOT gate. The outputs of a NAND gate is high if **any** of the inputs is low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

Algebraic function of NAND gate is X= (A . B)'
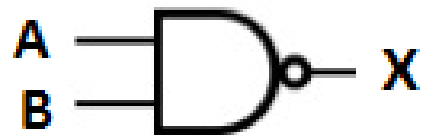
Graphic Symbol

Input		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

6. NOR Logic gate: - This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if **any** of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

Algebraic function of NOR gate is X= (A + B)'

Graphic Symbol
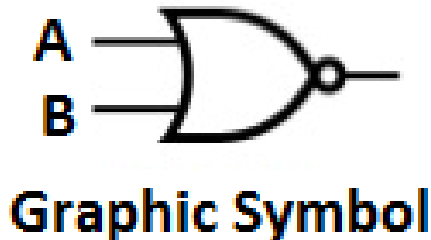
Input		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table

7. Exclusive - OR (XOR): - The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign () is used to show the EOR operation.

Algebraic function of XOR gate is **X= A' B + AB'** or **X = (A ⊕ B)**

Graphic Symbol

Input		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table

8. Exclusive – NOR (XNOR) : - The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.
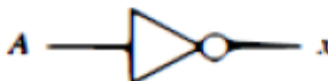
Algebraic function of NOR gate is **X= A'B' + AB or X = (A ⊙ B)**

Graphic Symbol

Input		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table

Name	Graphic symbol	Algebraic function	Truth table
AND	A —⊃— x B	$x = A \cdot B$ or $x = AB$	A B x 0 0 0 0 1 0 1 0 0 1 1 1
OR	A —⊃— x B	$x = A + B$	A B x 0 0 0 0 1 1 1 0 1 1 1 1
Inverter	A —▷○— x	$x = A'$	A x 0 1 1 0
Buffer	A —▷— x	$x = A$	A x 0 0 1 1

			A	B	x
NAND		$x = (AB)'$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$x = (A + B)'$	0	0	1
			0	1	0
			1	0	0
			1	1	0
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	0	0	0
			0	1	1
			1	0	1
			1	1	0
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	0	0	1
			0	1	0
			1	0	0
			1	1	1

Summary of Digital Logic Gates

Unit 2

Topic Name – Demorgan's Theorem

Table of Contents

- Demorgan's Theorems

- Proof of Demorgan's theorems

- References

Demorgan's Theorem: - De Morgan's theorem states that complementing the result of OR'ing variables together is equivalent to AND'ing the complements of the individual variables (Theorem 1). Also, complementing the result of AND'ing variables together is equivalent to OR'ing the complements of the individual variables (Theorem 2).

1. $\overline{A + B} = \overline{A} \cdot \overline{B}$ Theorem 1

2. $\overline{A \cdot B} = \overline{A} + \overline{B}$ Theorem 2

The theorem explains that the complement of the sum of all the terms is equal to the product of the complement of each term. Likewise, the complement of the product of all the terms is equal to the sum of the complement of each term.

Proof of Demorgan's theorem using two variables

A	B	$\overline{A}$	$\overline{B}$	A+B	A.B	$\overline{A+B}$	$\overline{A}.\overline{B}$	$\overline{A.B}$	$\overline{A}+\overline{B}$
0	0	1	1	0	0	1	1	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	1	0	0	0	0
						Theorem 2		Theorem 1	

Truth Table

Demorgan's theorems for three variables

1. $\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C}$

2. $\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$

Proof of Demorgan's theorem using three variables.

A	B	C	$\overline{A}$	$\overline{B}$	$\overline{C}$	A+B+C	A.B.C	$\overline{A+B+C}$	$\overline{A} \cdot \overline{B} \cdot \overline{C}$	$\overline{A.B.C}$	$\overline{A}+\overline{B}+\overline{C}$
0	0	0	1	1	1	0	0	1	1	1	1
0	0	1	1	1	0	1	0	0	0	1	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0	0	0	1	1
1	0	0	0	1	1	1	0	0	0	1	1
1	0	1	0	1	0	1	0	0	0	1	1
1	1	0	0	0	1	1	0	0	0	1	1
1	1	1	0	0	0	1	1	0	0	0	0
								Theorem 2		Theorem 1	

Truth Table

Unit 1

Topic Name – K Map

Table of Contents

- Karnaugh Map

- Use of K - Map

- Steps to solve expression using K-map

- Minterms and Maxterms

- Examples

- References

Karnaugh Map (K - Map): - The K-map is a systematic way of simplifying Boolean expressions. With the help of the K-map method, we can find the simplest POS and SOP expression, which is known as the minimum expression. In many digital circuits and practical problems we need to find expression with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. K-map is a graphical method, which consists of 2^n cells for 'n' variables.

Use of K - Map: - Karnaugh maps reduce logic functions more quickly and easily compared to Boolean algebra. By reduce we mean simplify, reducing the number of gates and inputs. We like to simplify logic to a lowest cost form to save costs by elimination of components. We define lowest cost as being the lowest number of gates with the lowest number of inputs per gate.

SOP and POS forms: - Representation of Boolean expression can be primarily done in two ways. They are as follows:

1. Sum of Products (SOP) form

2. Product of Sums (POS) form

If the value of input variable (let A) is :

Zero (0) – It is represented as A' (or Complement of A)

One (1) – It is represented as A

Sum of Products (SOP): - It is one of the ways of writing a Boolean expression. As the name suggests, it is formed by adding (OR operation) the product terms. These product terms are also called as 'min-terms'. Min-terms are represented as m, they are the product (AND operation) of Boolean variables either in normal form or complemented form.

Example: - F = AB'CD + AB'C'D + A'B'CD + A'B'C'D

Product of Sums (POS): - As the name suggests, it is formed by multiplying (AND operation) the sum terms. These sum terms are also called as 'max-terms'. Max-terms are represented as M, they are the sum (OR operation) of Boolean variables either in normal form or complemented form.

Example: - F = (A + B' + C + D) . (A+ B'+ C' + D) . (A' + B' + C + D)

x	y	z	Minterm		Maxterm	
			Term	Symbolic Notation	Term	Symbolic Notation
0	0	0	$x'y'z'$	m_0	$X + Y + Z$	M_0
0	0	1	$x'y'z$	m_1	$X + Y + Z'$	M_1
0	1	0	$x'yz'$	m_2	$X + Y' + Z$	M_2
0	1	1	$x'yz$	m_3	$X + Y' + Z'$	M_3
1	0	0	$xy'z'$	m_4	$X' + Y + Z$	M_4
1	0	1	$xy'z$	m_5	$X' + Y + Z'$	M_5
1	1	0	xyz'	m_6	$X' + Y' + Z$	M_6
1	1	1	xyz	m_7	$X' + Y' + Z'$	M_7

Minterms and maxterms for three variables

Let us consider the truth table for three variables as listed in table below. A Boolean function can be algebraically from a given truth table by forming a minterm. For each combination of variables that produces 1 in the function and then taking OR between those terms. From the table given below the function Y1 is 1 for minterms x'y'z, xy'z', xyz' and xyz thus Y1 = x'y'z + xy'z' + xyz' + xyz

$$Y1 = m_1 + m_4 + m_6 + m_7$$

x	y	z	Y1	Y2
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

Similarly, the function Y2 can be expressed as Y2 = x'y'z' + x'yz + xyz'
$$= m_0 + m_3 + m_6$$

Example 1: - Express the Boolean function F = B'D + A'D + BD in sum of minterms.

Solution: - The function F contains four variables A, B, C and D. The first term B'D is missing two variables A and C.

Thus   B'D = B'D (A + A')   Because X + X' =1

            = AB'D + A'B'D

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

            AB'CD + AB'C'D + A'B'CD + A'B'C'D   ------------1

The Second term A'D missing B and C

Thus   A'D'= A'D(B + B')

            = A'BD + A'B'D

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

            = A'BCD + A'BC'D + A'B'CD + A'B'C'D   --------------2

The third term BD missing A and C

Thus   BD= BD(A+A')

$= ABD + A'BD$

$= ABD(C + C') + A'BD (C+C')$

$= ABCD + ABC'D + A'BCD + A'BC'D$ ------------3

Combining 1,2 and 3, we have

$F = A'BCD + AB'C'D + A'B'CD + A'B'C'D + A'BCD + A'BC'D + A'B'CD + A'B'C'D + ABCD$

$+ ABC'D + A'BCD + A'BC'D.$

But $A'B'C'D$, $A'B'CD$, $A'BC'D$ and $A'BCD$ appears twice. Rearranging all the terms and using $X + X = X$, We have

$F = A'B'C'D + A'B'CD + A'BC'D + A'BCD + AB'C'D + AB'CD + ABC'D + ABCD$

$= m_1 + m_3 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}$

Also the Boolean function can be expressed as

$F (A, B, C, D) = \sum(1, 3, 5, 7, 9, 11, 13, 15)$

Example 2: - Express the Boolean function F = x + y z as a sum of minterms.

Solution: - This function has three variables: x, y, and z. All terms must have these three variables. In first term x and y are missing and in second term x is missing.

Thus,   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

F(x, y, z) = Σ(3, 4, 5, 6, 7)

Unit 1

Topic Name – K Map

Table of Contents

- 2 Variables K- Map

- 3 Variables K- Map

- Examples

- References

K Map for two variables: -

Y'   Y

	Y'	Y
X'	x'y' 0 00	x'y 1 01
X	xy' 2 10	xy 3 11

K - Map for 2 variables

Example 1: - Simplify the function F= xy' + xy

	Y'	Y
X'		
X	1	1

F = x

Example 1: - Simplify the function F=x'y + xy + xy'

	Y'	Y
X'		1
X	1	1

F = x + y

Example 1: - Simplify the function F=x'y' + xy

	Y'	Y
X'	1	
X		1

F = x'y' + xy

K Map for 3 variables: -

	Y'Z'	Y'Z	YZ	YZ'
X'	x'y'z' 000 0	x'y'z 001 1	x'yx 011 3	x'yz' 010 2
X	xy'z' 100⁻ 4	xy'z 101 5	xyz 111 7	xyz' 110 6

Example 1: - Simplify the function F = A'B'C' + A'BC' + A'B'C

	B'C'	B'C	BC	BC'
A'	1	1		1
A				

$$F = A'B' + A'C'$$

Example 2: - Simplify the function F = x'z + x'y + xy'z + yz

Solution: -   To simplify F, first f must be converted into canonical form . After converting into canonical form, F = x'y'z + x'yz' + x'yz + xy'z + xyz

	Y'Z'	Y'Z	YZ	YZ'
X'		1	1	1
X		1	1	

Simplified function F = z + x' y

Example 3: - Simplify the function F (x, y, z) = ∑ (0, 2, 4, 5, 6)

	Y'Z'	Y'Z	YZ	YZ'
X'	0	1	3	2
X	4	5	7	6

	Y'Z'	Y'Z	YZ	YZ'
X'	1			1
X	1	1		1

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

Example 4: - Simplify the function F = x'yz + xy'z'+ xyz + xyz'

	Y'Z'	Y'Z	YZ	YZ'
X'			1	
X	1		1	1

F = yz + xz'

Example 5: - Simplify the function F = x'yz + x'yz' + xy'z' + xy'z

	Y'Z'	Y'Z	YZ	YZ'
X'			1	1
X	1	1		

F = x'y + xy'

Unit 1

Topic Name –K Map

Table of Contents

- 4 Variables K- Map

- Examples

- References

Important Point regarding K Map: -

1. The square that contains '1' should be taken in simplifying, at least once.

2. The square that contains '1' can be considered as many times as the grouping is possible with it.

3. Group shouldn't include any zeros (0).

4. A group should be the as large as possible.

5. Groups can be horizontal or vertical. Grouping of variables in diagonal manner is not allowed.

6. If the square containing '1' has no possibility to be placed in a group, then it should be added to the final expression.

7. Groups can overlap.

8. The number of squares in a group must be equal to powers of 2, such as 1, 2, 4, 8, 16 etc.

9. Groups can wrap around. As the K-map is considered as spherical or folded, the squares at the corners (which are at the end of the column or row) should be considered as they adjacent squares.

10. The grouping of K-map variables can be done in many ways, so the obtained simplified equation need not to be unique always.

11. The Boolean equation must be in must be in canonical form, in order to draw a K-map.

K Map for 4 variables: -

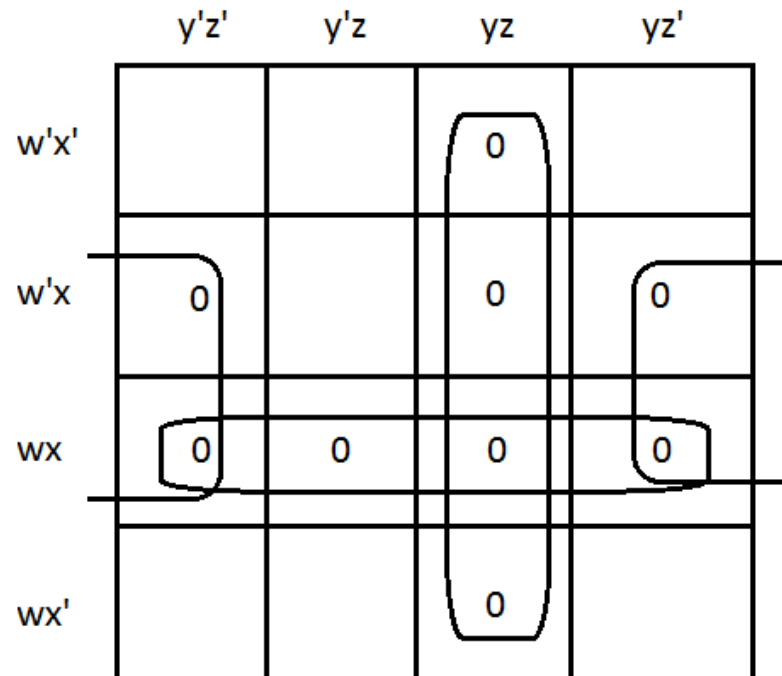|	y'z'	y'z	yz	yz'
w'x'	w'x'y'z' 0000 0	w'x'y'z 0001 1	w'x'yz 0011 3	w'x'yz' 0010 2
w'x	w'xy'z' 0100 4	w'xy'z 0101 5	0111 w'xyz 7	0110 w'xyz' 6
wx	wxy'z' 1100 12	wxy'z 1101 13	wxyz 1111 15	wxyz' 1110 14
wx'	wx'y'z' 1000 8	wx'y'z 1001 9	wx'yz 1011 11	wx'yz' 1010 10

Example 1: - Simplify the following Boolean function

F(w,x,y,z) = ∑(0,2,5,6,7,8,10) in (a) SOP and (b) POS forms

Solution (a): - Place 1 in the given minterms and group the adjacent 1.

	y'z'	y'z	yz	yz'
w'x'	1			1
w'x		1	1	1
wx				
wx'	1			1

Simplified SOP is F= x'z' + w'xz + w'xy

(b): - Place 0 in the minterms which are not included in the function F and group the adjacent 0.

	y'z'	y'z	yz	yz'
w'x'		0	0	
w'x	0			
wx	0	0	0	0
wx'		0	0	
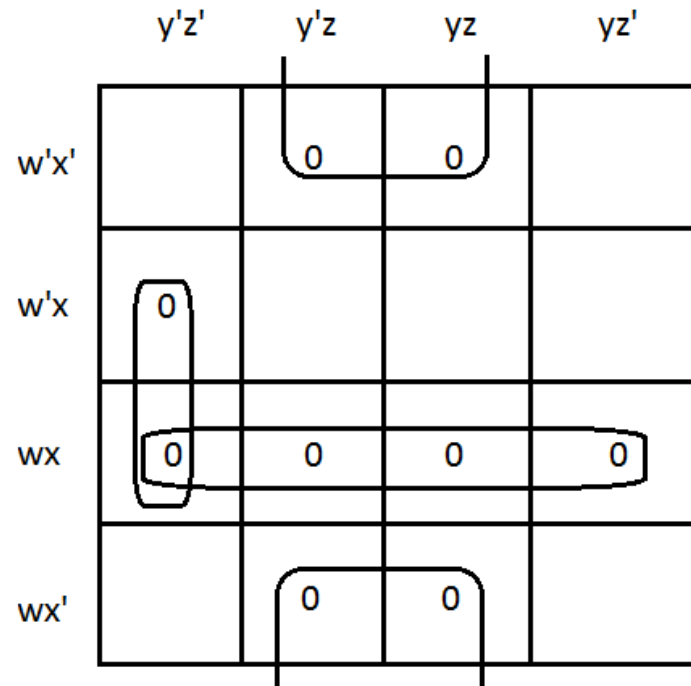
F' = wx + x'z + xy'z' Taking the complement od both of the sides we have,

F = $\overline{wx + x'z + xy'z'}$ Apply Demorgan's law, $\overline{A + B} = \overline{A} . \overline{B}$

F = $\overline{wx} . \overline{x'z} . \overline{xy'z'}$ Apply Demorgan's law, $\overline{A . B} = \overline{A} + \overline{B}$

F = (w' + x') . (x + z') . (x' + y + z)

Example 2: - Simplify the following Boolean function

F(w,x,y,z) = ∑(0,1,2,5,8,9,10) in (a) SOP and (b)POS forms

Solution (a): - Place 1 in the given minterms and group the adjacent 1.

	y'z'	y'z	yz	yz'
w'x'	1	1		1
w'x		1		
wx				
wx'	1	1		1
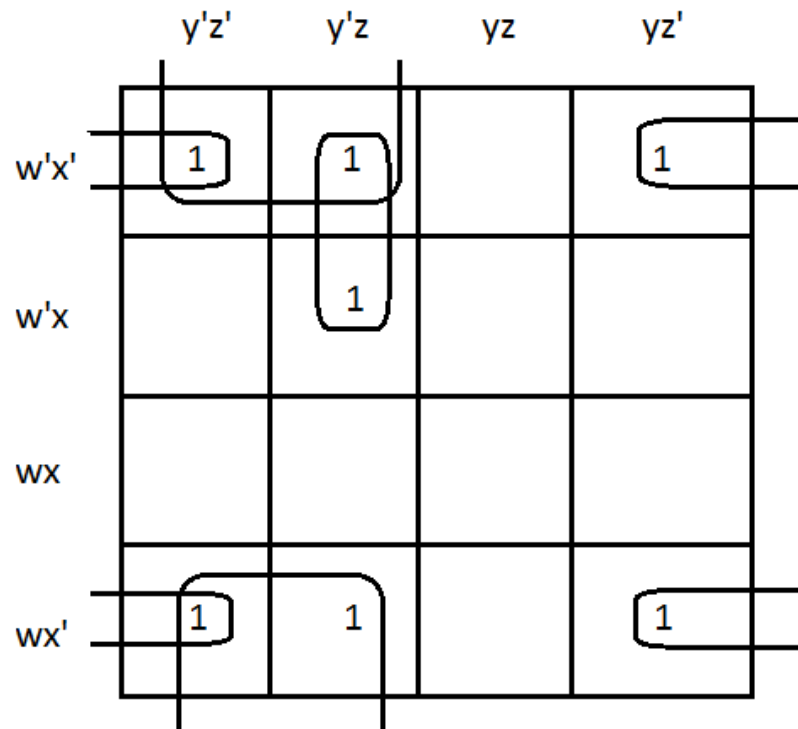
Simplified SOP is F= x'z' + x'y' + w'y'z

(b): - Place 0 in the minterms which are not included in the function F and group the adjacent 0.

	y'z'	y'z	yz	yz'
w'x'			0	
w'x	0		0	0
wx	0	0	0	0
wx'			0	

F = wx + yz + xz' Taking complement of both of the side and applying Demorgan's theore we have

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

Unit 2

Topic Name –K Map

Table of Contents

- 4 Variables K- Map

- Don't care condition

- Examples

- References

K Map for 4 variables: -

	y'z'	y'z	yz	yz'
w'x'	w'x'y'z' 0000 0	w'x'y'z 0001 1	w'x'yz 0011 3	w'x'yz' 0010 2
w'x	w'xy'z' 0100 4	w'xy'z 0101 5	0111 w'xyz 7	0110 w'xyz' 6
wx	wxy'z' 1100 12	wxy'z 1101 13	wxyz 1111 15	wxyz' 1110 14
wx'	wx'y'z' 1000 8	wx'y'z 1001 9	wx'yz 1011 11	wx'yz' 1010 10

Don't care condition: - Some times the function behaves the same even if some minterms have values 0 or 1, i.e., it does not matter if the function produces 0 or 1 for a given minterm. Since the function may be 0 or 1, we can say that we don't care about the output of the function for this minterm. Minterms that may produce either 0 or 1 for the function are said to be don't care conditions and are marked with an X in the map. These don't care conditions can be used to provide for their simplification of the algebraic expression.

When we choose adjacent squares to group for the function in the map, the x's may be assumed to be either 0 or 1, which ever gives the simplest expression. We need not use an X at all if it does not contribute to the simplification of the function.

Example 1:

Simplify the function F with don't care condition d

$$F(A, B, C) = \Sigma(0, 2, 3, 6)$$

$$d = \Sigma(1, 5)$$

Solution: The minterms listed with F produce a 1 for the function. The don't care minterms listed with d may produce either a 0 or 1 for the function. Place 1 in the given minterm positions in the k map and put a cross (X) in minterm positions given with d (i. e. 1, 5) and assume X to be 1 to form a group.

	B'C'	B'C	BC	BC'
A'	1	X	1	1
A		X		1

Example 2:

Simplify the function F with don't care condition d

$$F (w, x, y, z) = \sum (1, 3, 7, 11, 15)$$

$$d = \sum (0, 2,\ 5)$$

Solution: Place 1 in the given minterm positions in the k map and put a cross (X) in minterm positions given with d (i. e. 0, 2, 5) and assume X to be 1 to form a group.

	y'z'	y'z	yz	yz'
w'x'	X	1	1	X
w'x		X	1	
wx			1	
wx'			1	

Simplified function F = x'z + yz

Example 3:

Simplify the function F with don't care condition d

$$F (w, x, y, z) = \sum (0, 3, 7, 8, 9, 11, 12, 13)$$

$$d = \sum (1, 4, \ 14, 15)$$

Solution:

	y'z'	y'z	yz	yz'
w'x'	1	X	1	
w'x	X		1	
wx	1	1	X	X
wx'	1	1	1	

Simplified function F= y'z' + yz + wy'

Example 4:

Simplify the function F with don't care condition d

$$F(w, x, y, z) = \sum (0, 1, 2, 3, 7, 8, 10)$$

$$d = \sum (5, 6, 11, 15) \text{ in SOP and POS}$$

Solution:

	y'z'	y'z	yz	yz'
w'x'	1	1	1	1
w'x		X	1	X
wx			X	
wx'	1		X	1

$$F = x'z' + w'z$$

For POS form, place 0 in the minterm positions which are not given with the function F and put a cross (X) in minterm positions given with d (i. e. 5,6,11,15) and assume X to be 0 to form a group.

	y'z'	y'z	yz	yz'
w'x'				
w'x	0	X		X
wx	0	0	X	0
wx'		0	X	

(POS form) F' = wz + xz' taking complement of both sides and applying Demorgan's theorem

$F = (w' + z') . (x' + z)$

Topic Name – Types of Logic Circuits

Table of Contents

- Logic Circuits

- Combinational Circuit

- Types of Combinational Circuits

- Half Adder

- Full Adder

- References

Digital Circuit and its type: A digital circuit is a type of circuit that operates on different logic gates. A digital circuit comprises various components each of which is used to perform a specific task. In a digital circuit the signal must be one of two discrete levels. Each level is interpreted as one of two different states (for example, on/off, 0/1, true/false).

A digital circuit is designed by using a number of logic gates on a single Integrated Circuit – IC. The input to any digital circuit is in the binary form "0's" and "1's". The output obtained on processing raw digital data is of a precise value. These circuits can be represented in two ways which are as follows.

1. Combinational Circuits
2. Sequential Circuits

1. **Combinational Circuit: -** The combinational circuit is made up of logic gates whose output is determined by the present input only. It is realized by various logic gates like AND, OR, NOT, NAND, NOR etc. Various types of combinational circuits are as follows.

1. Adders

   1.1 Half Adders

   1.2 Full Adders

2. Subtractors

   2.1 Half Subtractors

   2.2 Full Subtractors

3. Decoders

4. Encoders

5. Multiplexers

6. Demultiplexers

n input variables → **Combinational Logic Circuits** → **m output variables**
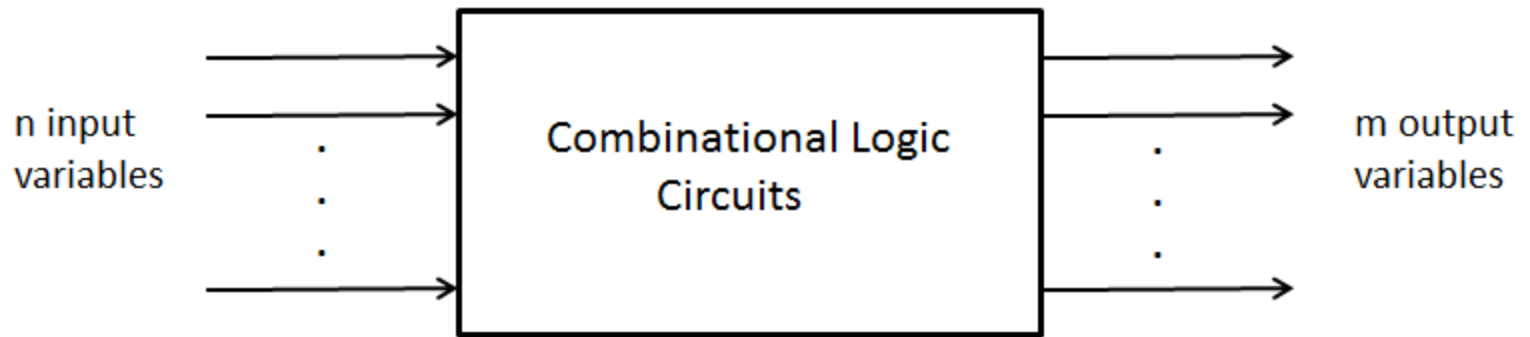
Block Diagram of a Combinational Circuits

1. Adders : An adder is a digital logic circuit which is extensively used for the addition of binary numbers.

1.1 Half Adders : - Half adder is a combinational circuit , which is used to add two binary bits. It requires two binary inputs and two binary outputs sum and carry respectively.

Where, Sum = x'y + xy'   and Carry = xy

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table
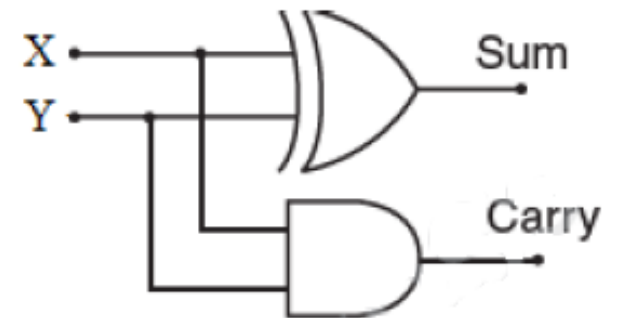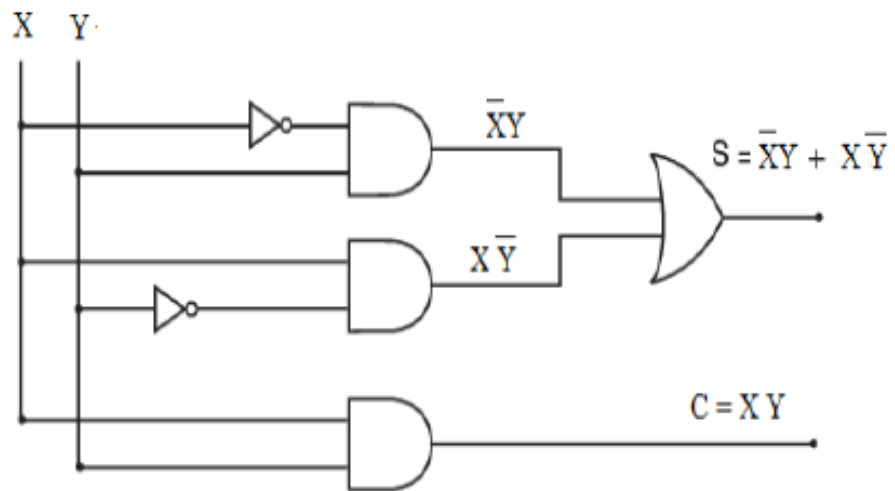
Logic Diagram of Half Adder
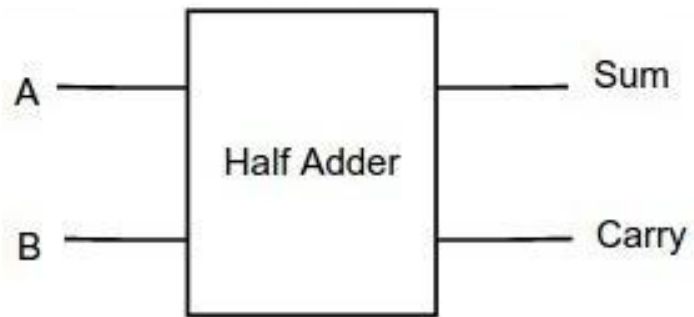
Block Diagram
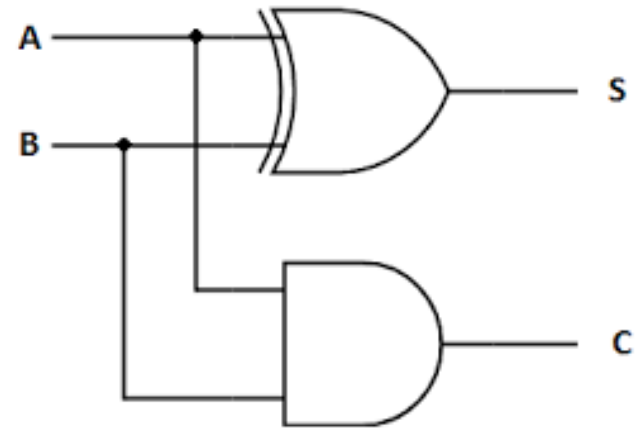
Half adder

1. Adders

1.2 Full Adders: - Full adder is a combinational circuit , which is used to add three binary bits. It requires three binary inputs and two binary outputs sum and carry respectively.

Truth table for full adder is as follow

Inputs			Outputs	
X	Y	Z	Sum S	Carry C
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

Truth Table of Full Adder

(a) S = X'Y'Z + X'YZ' + XY'Z' + XYZ

(b) C = XZ + YZ + XY

K - Map for Sum (s) and Carry (C)
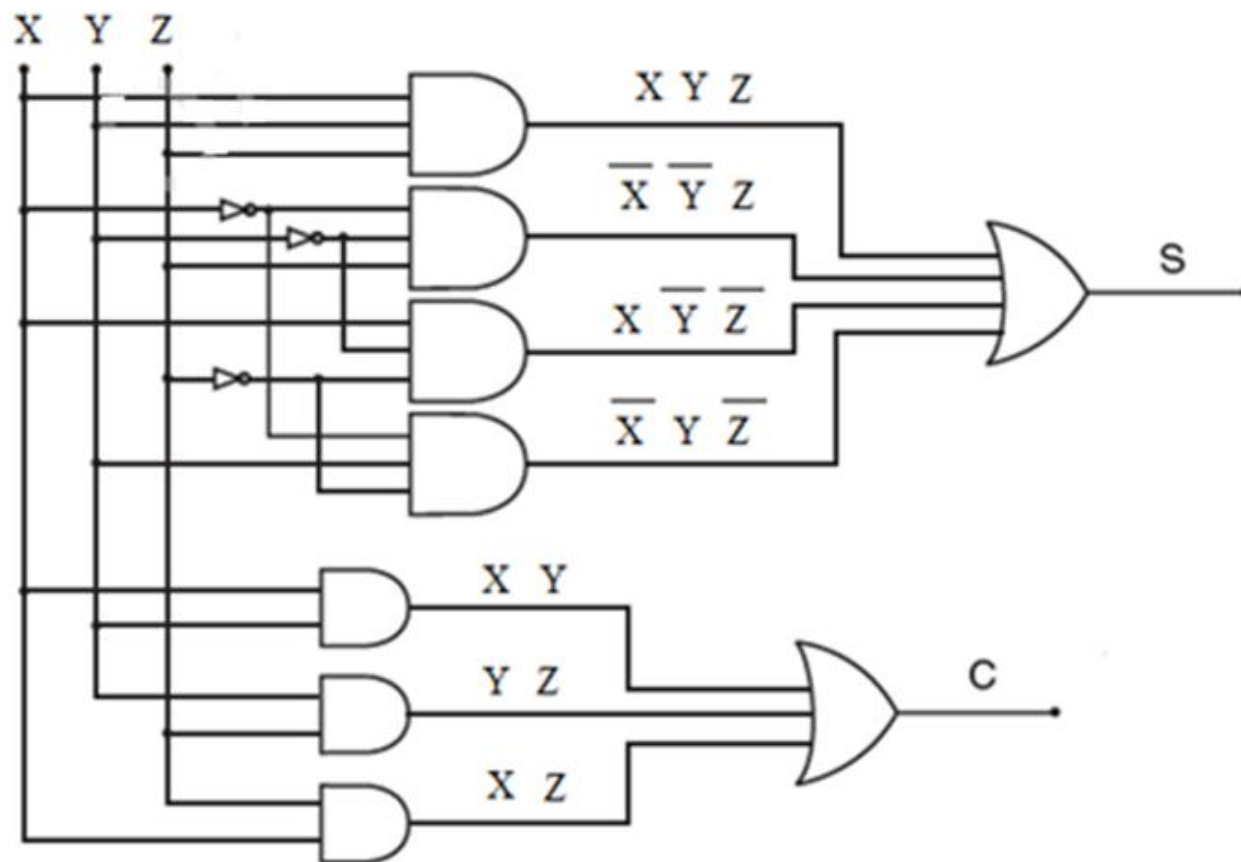
Logic Diagram of Full Adder

2. Subtractors: - Subtractor is used to subtract two or more binary bits.

2.1 Half Subtractors: - The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and Bout (borrow out).

Where, Difference (D) = x'y + xy'   and Borrow (B) = x'y

Truth table of half subtractor is as follows.

Inputs		Outputs	
Minuend X	Subtrahend Y	Difference D	Borrow B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Boolean expressions for difference (D) and Borrow out (B) can be written as follows:

$$D = X'Y + XY' \text{ or } D = X \oplus Y$$

$$B = X'Y$$

The logic diagram of half adders is given below.
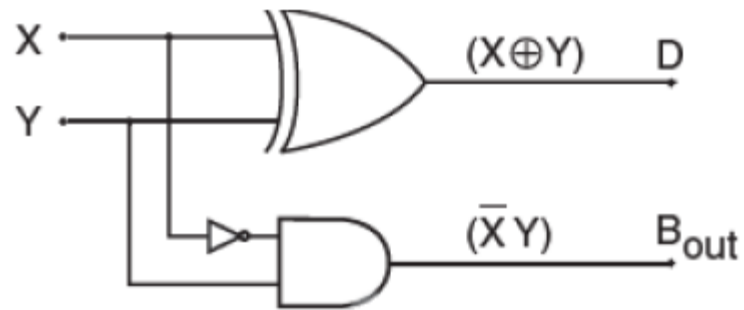
Logic diagram of half subtractor

2.2 Full Subtractor: - Full subtractor is a combinational circuit that performs subtraction of three binary bits. The truth table of the full adders is as follows.

Inputs			Outputs	
X	Y	Z	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

(a) D=X'Y'Z+X'YZ'+XY'Z'+XYZ

(b) B = X'Z+X'Y+YZ

K Map for Diffference (D) and Borrow (B)
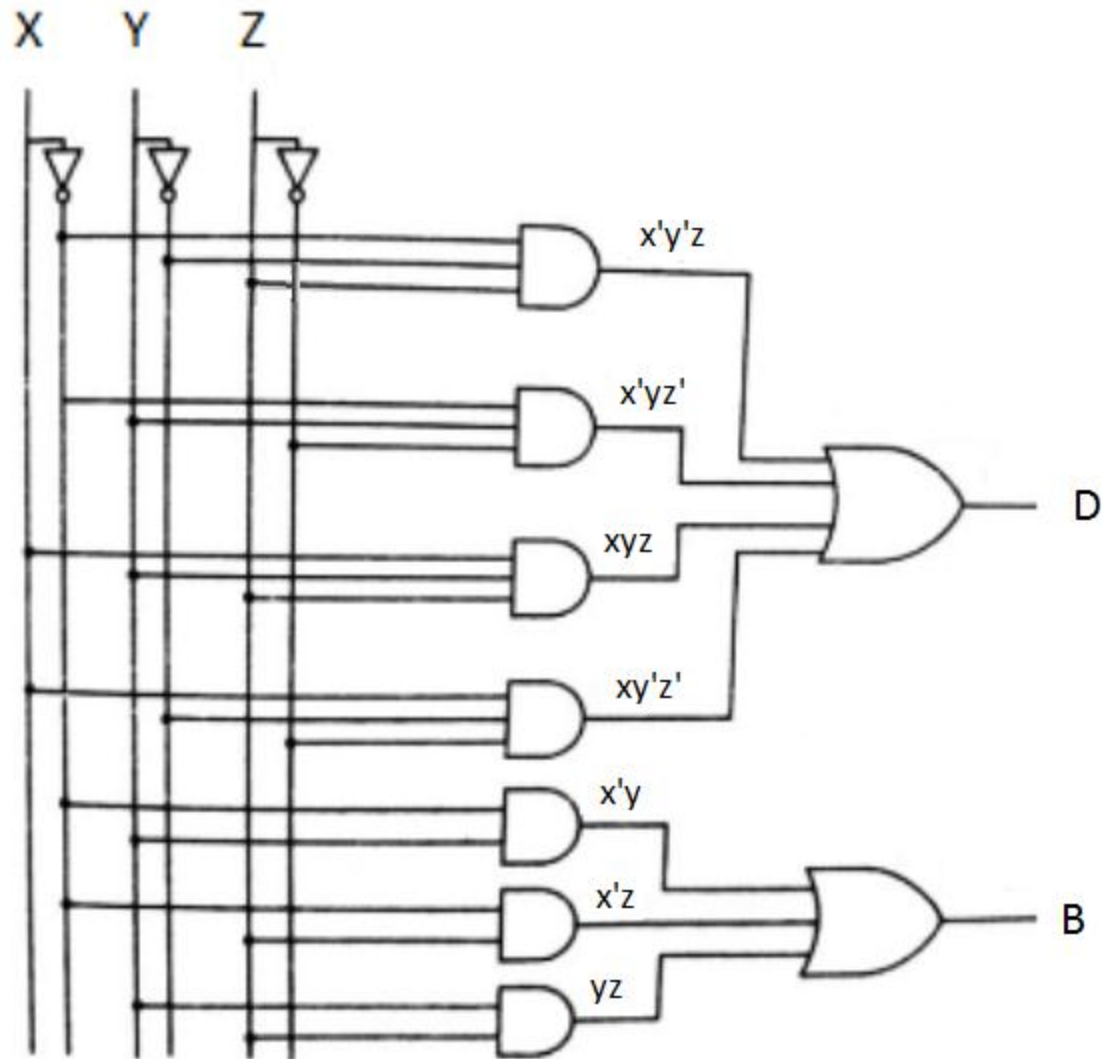
Logic Diagram of Full Subtractor

Topic Name – Decoder and Encoder

Table of Contents

- Decoder

- 3X8 Decoder using two 2X4 decoders

- Encoder

- References

Decoder: - Discrete quantities of information are represented in digital systems by binary codes. A binary code of n bits is capable of representing up to 2^n distinct elements of coded information. A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.
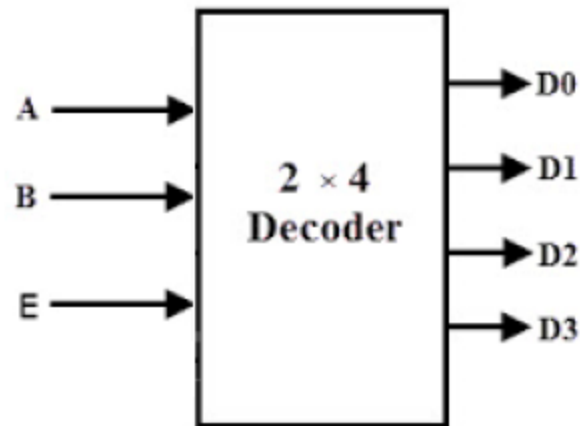
Block Diagram of 2X4 Decoder
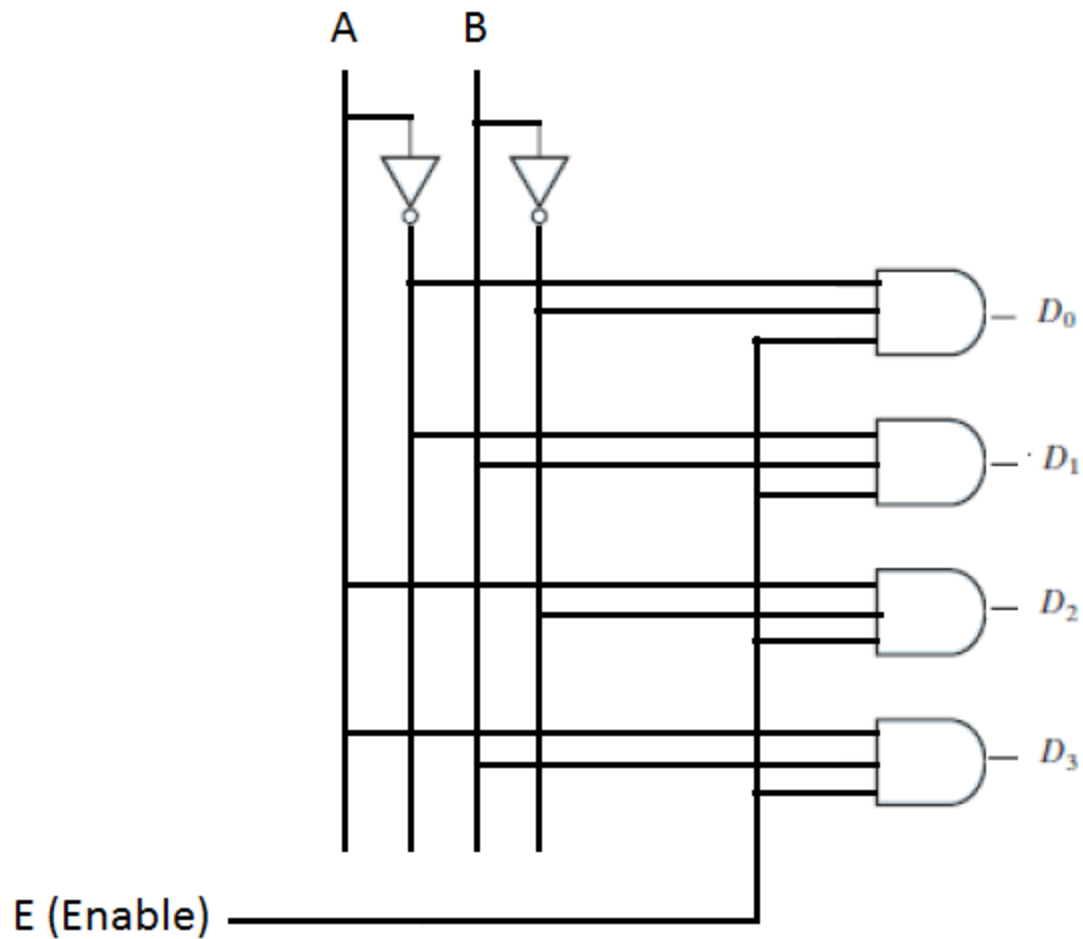
Logic Diagram of 2X4 Decoder

Enable	Inputs		Outputs			
E	A	B	D_3	D_2	D_1	D_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Truth Table of 2X4 Decoder

3X8 Decoder using two 2X4 Decoders: - Decoders with enable inputs can be connected together to form a larger decoder. shows two 3-to-8-line decoders with enable inputs connected to form a 4-to-16-line decoder. When E = 0, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top four outputs generate minterms 000 to 011. When E = 1, the enable conditions are reversed: The bottom decoder outputs generate minterms 100 to 111, while the outputs of the top decoder are all 0's.

3 X 8 Decoder using two 2X4 Decoder
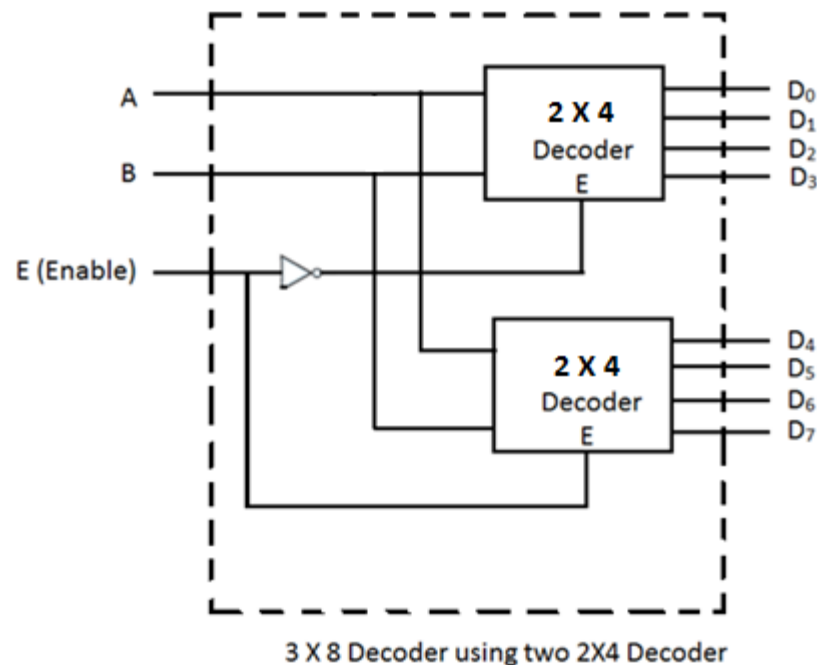
Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Truth Table of 3 X 8 Decoder

Encoder:- An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

An example of an encoder is the octal-to-binary encoder. It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Truth Table of 8X3 Encoder

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7. Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions:

$$X = D_4 + D_5 + D_6 + D_7$$

$$Y = D_2 + D_3 + D_6 + D_7$$

$$Z = D_1 + D_3 + D_5 + D$$

The encoder can be implemented with three OR gates.
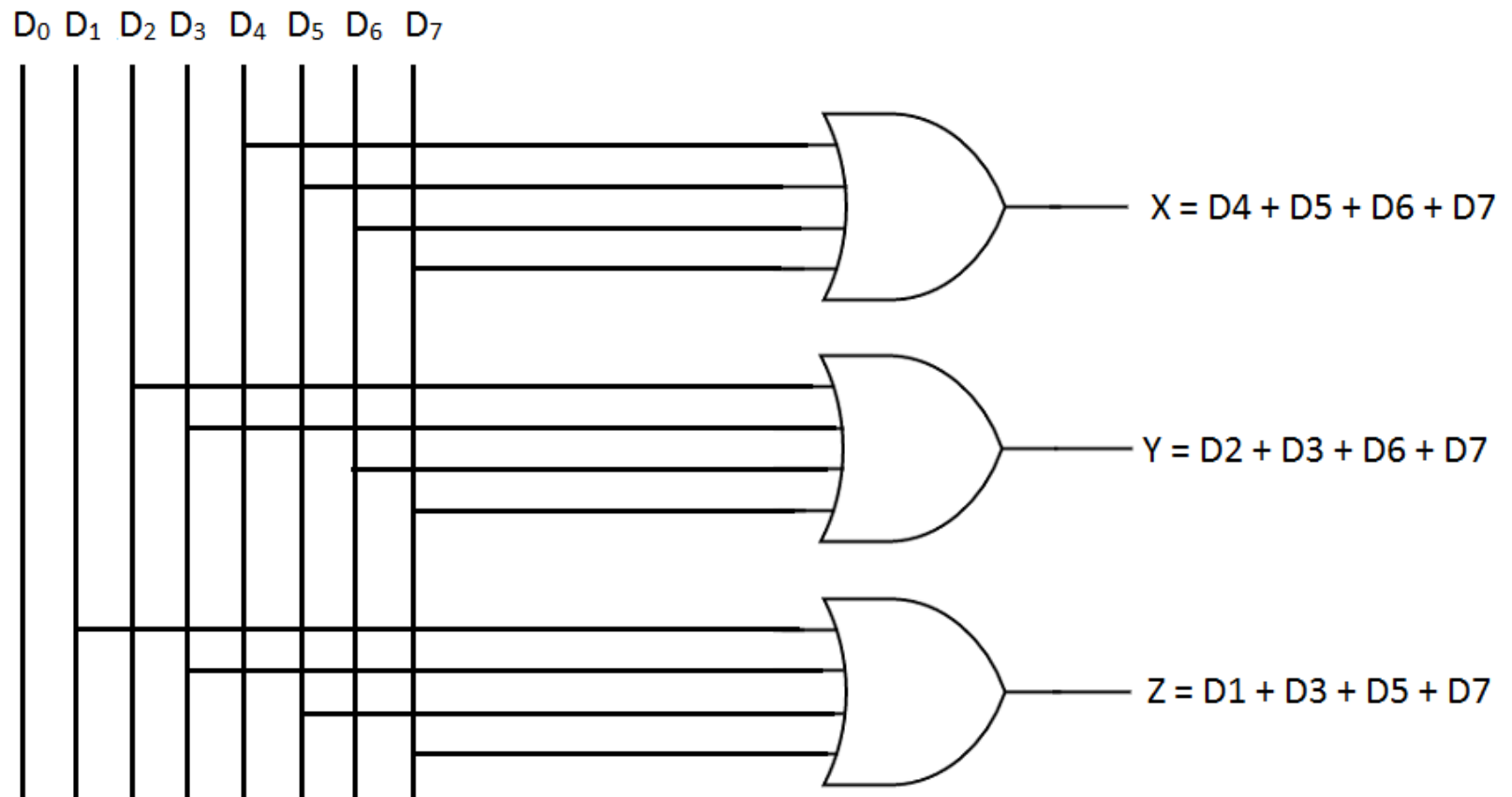
Logic Diagram of 8X3 Encoder

$$X = D4 + D5 + D6 + D7$$

$$Y = D2 + D3 + D6 + D7$$

$$Z = D1 + D3 + D5 + D7$$

Topic Name – Multiplexer and Demultiplexer

Table of Contents

- Multiplexer

- Diagram and Truth table

- Cascading of multiplexers

- Demultiplexer

- Diagram and Truth table

- References

Multiplexer: - A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. In other words it receives 2^n input lines and produces only one output line. The selection of a particular input line is controlled by a set of selection lines. <u>Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.</u>

Multiplexers are also known as **"Data selector, or many to one device.** Multiplexers are mainly used to increase amount of the data that can be sent over the network within certain amount of time and bandwidth.
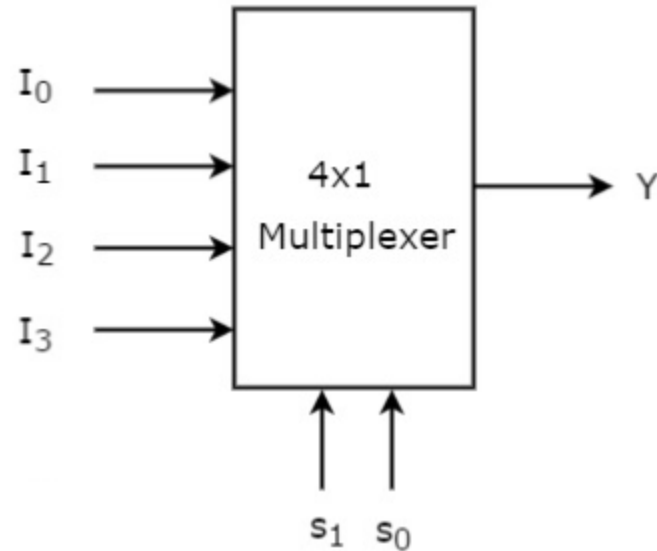
Fig. Block Diagram of a 4X1 multiplexer

I_0, I_1, I_2 and I_3 are inputs and Y is output. S_0 and S_1 are selection lines. Output may be I_0, I_1, I_2 or I_3, it depends on the combination of selection lines S_0 and S_1

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Truth Table of a 4X1 multiplexer

When the selection lines S_0 and S_1 are 0,0 the multiplexer will produce I_0 as output and when selection lines are (0,1) , (1,0) or (1,1) it will produce output I_1 , I_2 or I_3 respectively.
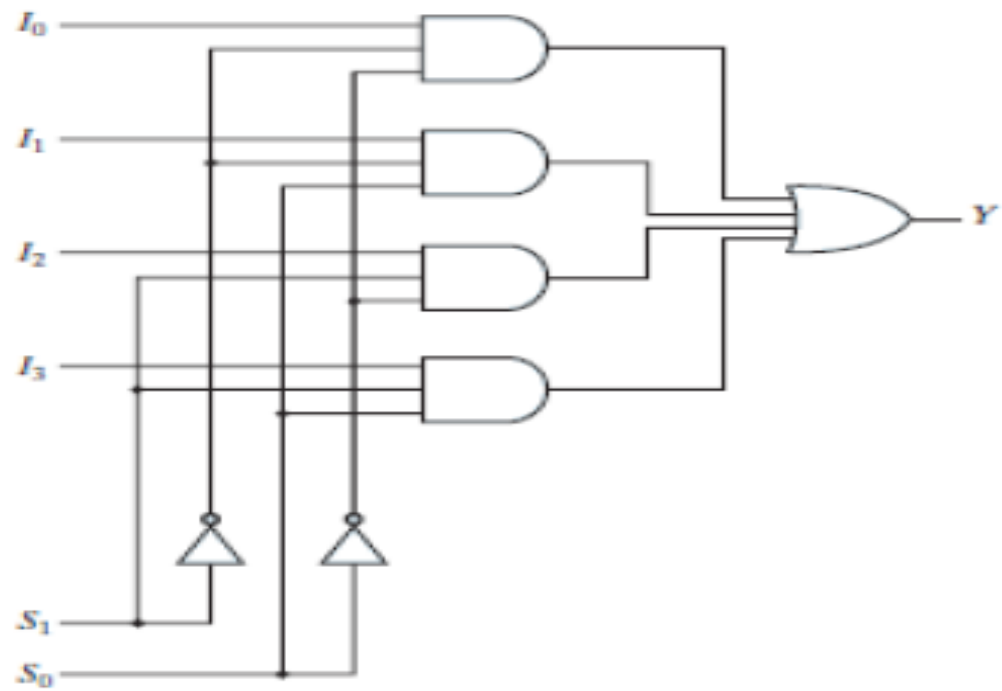
Logic diagram of 4X1 multiplexer

Cascading of multiplexers

Cascading refers a process where a large multiplexer can be designed or implemented using smaller multiplexers.

In this section, let us implement 8x1 Multiplexer using two 4x1 Multiplexers and one 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.
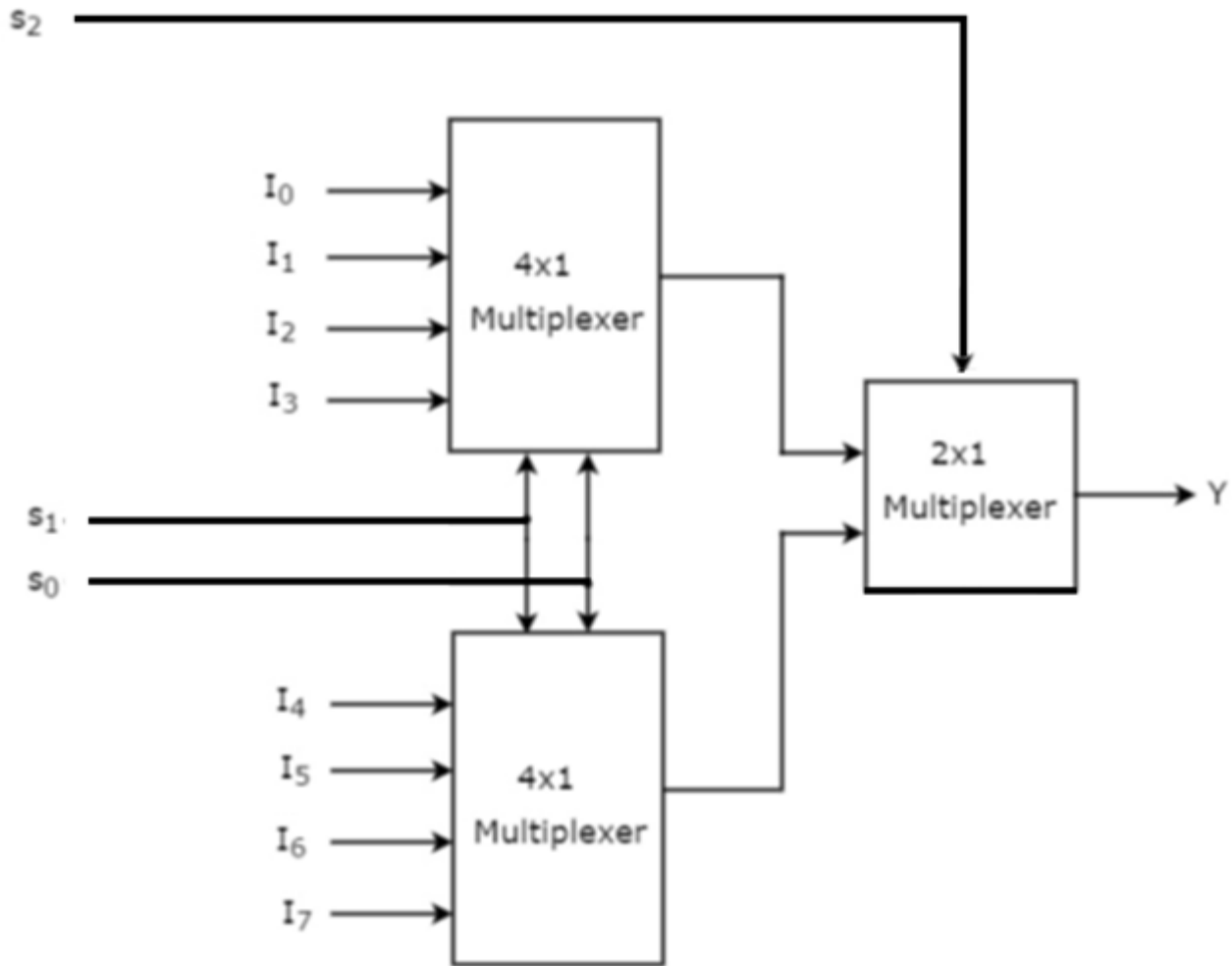
Designing of a 8X1 multiplexer using two 4X1 and one 2X1 multiplexer

Description of the diagram

The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_0 to I_3 and the data inputs of lower 4x1 Multiplexer are I_4 to I_7. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Truth Table of a 4X1 multiplexer

MP3 Player
Docking Station

Laptop
Sound Card

Digital
Satellite

Digital
Cable TV

D0
D1
D2
D3

MUX

Y

Surround Sound System

B	A	Selected Source
0	0	MP3
0	1	Laptop
1	0	Satellite
1	1	Cable TV
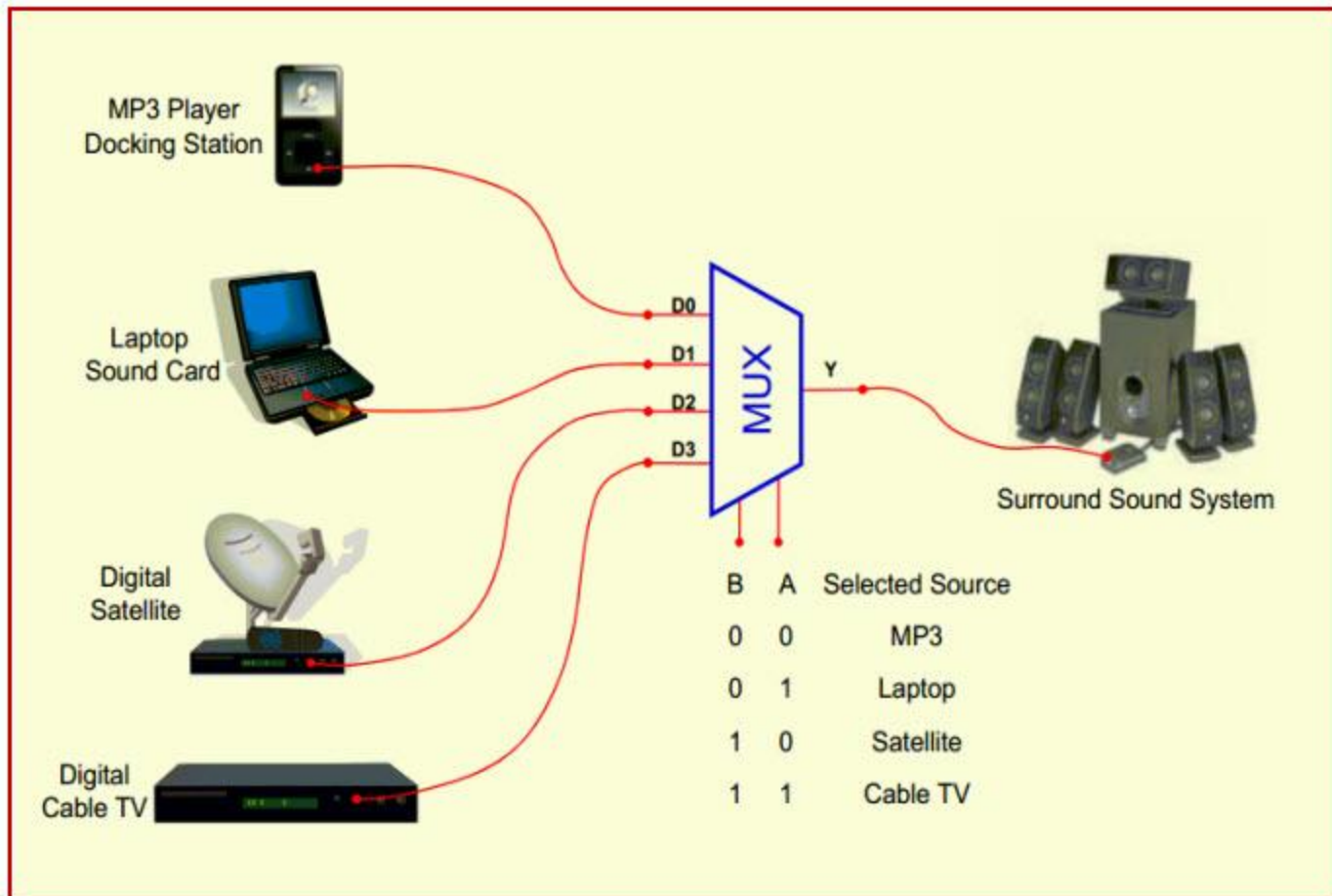
Demultiplexer: - Demultiplexer is a combinational circuit. These are also called data distributors or 'one to many' device. A demultiplexer (abbreviated as DMUX) performs the reverse or opposite operation of a multiplexer. That is, it takes a single input and distributes it over several outputs.

In a multiplexer there are 2^n output lines, n selection lines and only one input line. The role of selection lines is to select one output line from the 2^n output lines. Demultiplexer provides its input data a specific direction to flow through.

Types of Demultiplexer: - Common types of demultiplexers are as follows.

- 1X2 Demultiplexer (1 Selection Line)

- 1X4 Demultiplexer (2 Selection Lines)

- 1X8 Demultiplexer (3 Selection Lines)

- 1X16 Demultiplexer (4 Selection Lines)

- 1X32 Demultiplexer (5 Selection Lines)

Applications of Demultiplexer: -

- Demultiplexer is used to connect a single source to multiple destinations.

- Communication systems – Generally multiplexer and demultiplexer are used together, because communicational systems are bi directional. Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission, on the other hand demultiplexer is used to distribute the single data line into multiple data lines.

- Security monitoring Systems

- Synchronous data transmission
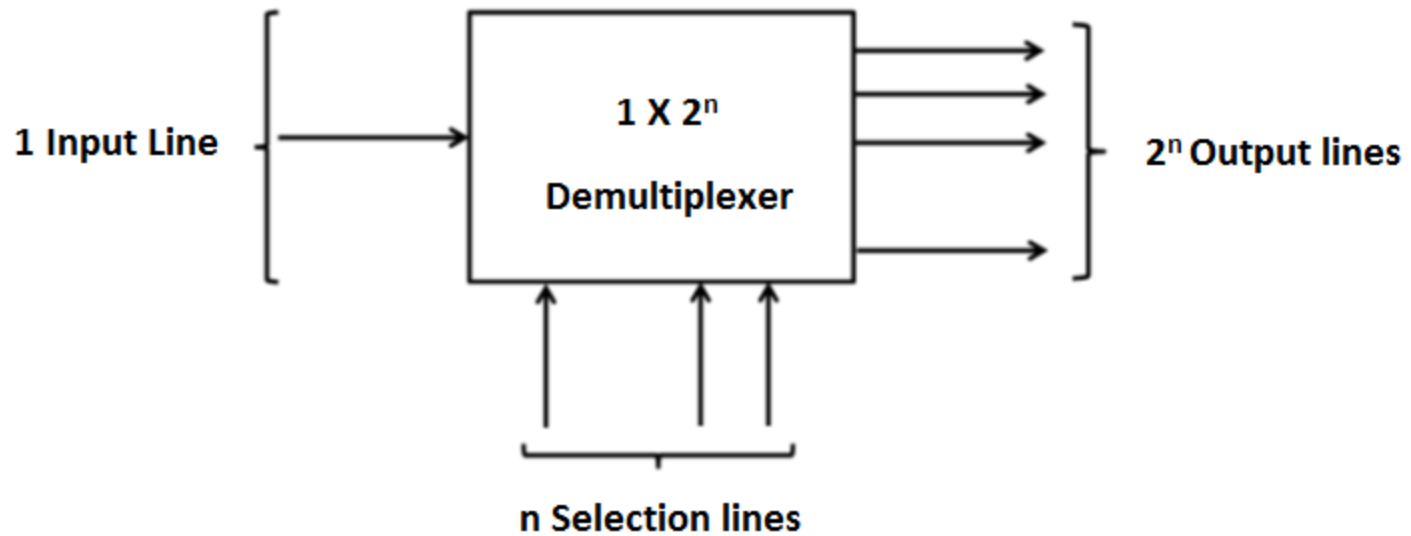
Block Diagram of a Demultiplexer
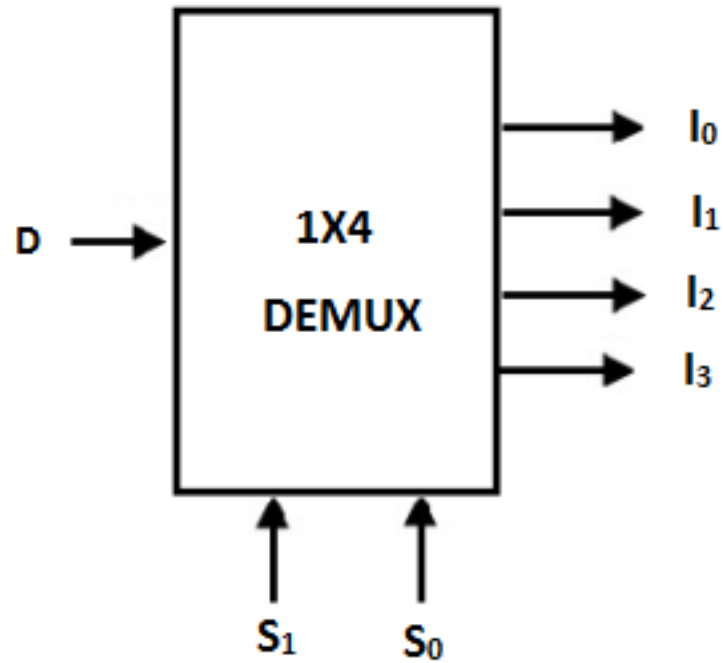
Block Diagram of a 1X4 Demultiplexer

Inputs			Outputs			
D	S_1	S_0	I_3	I_2	I_1	I_0
X	0	0	0	0	0	1
X	0	1	0	0	1	0
X	1	0	0	1	0	0
X	1	1	1	0	0	0

Truth Table of a 1X4 Demultiplexer
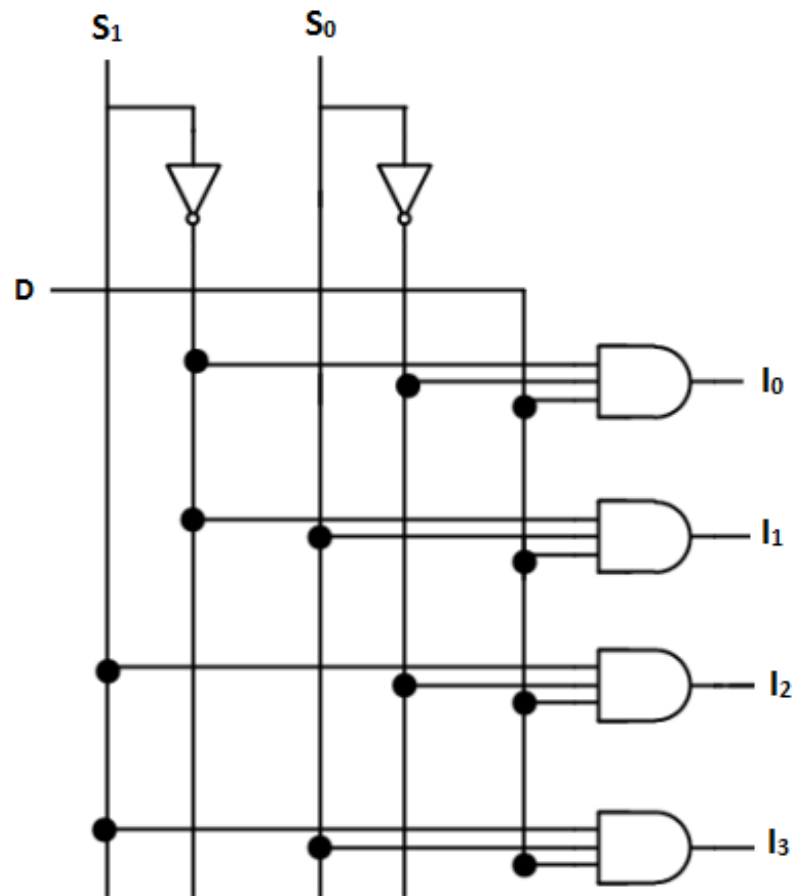
Logic Diagram of a 1X4 Demultiplexer
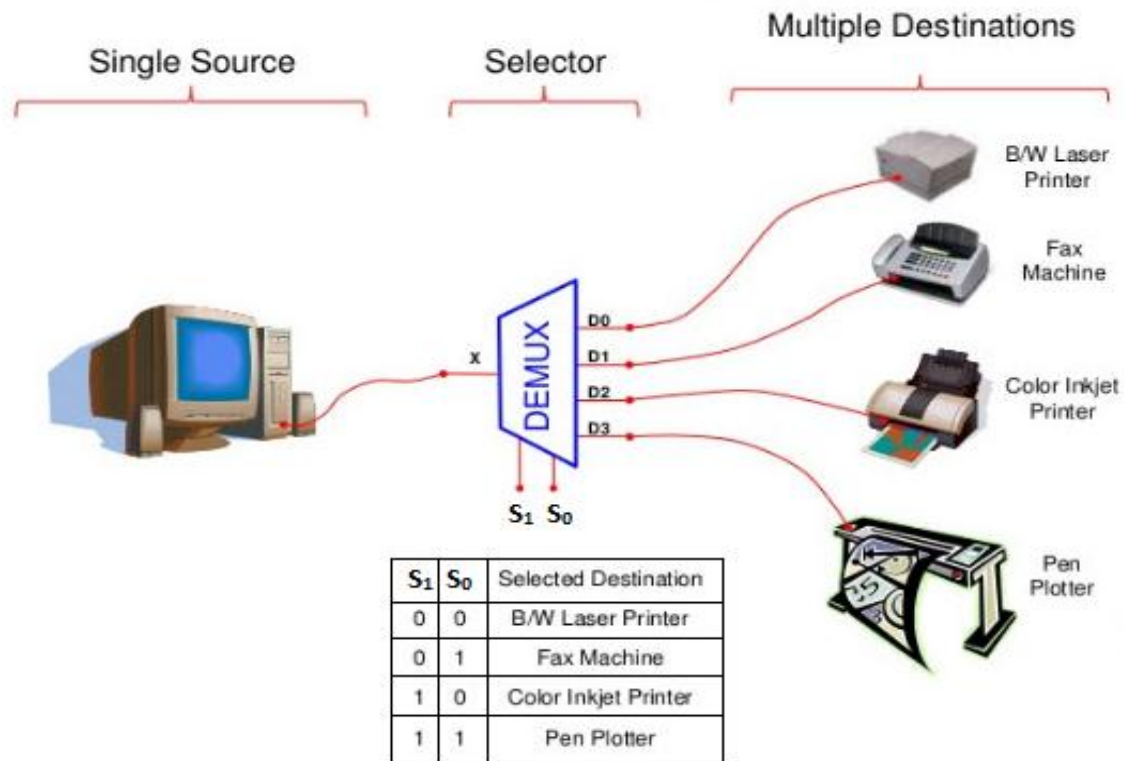
Example of Demultiplexing Process

Topic Name – Sequential circuits, Flip Flops

Table of Contents

- Definition/Concept

- Block Diagram

- Flip flops

- Block/Logic Diagram of SR flip flop

- Truth Table

- Applications

- References

Sequential Circuit: - This sequential circuit contains a set of inputs and outputs. The outputs of sequential circuit depends not only on the combination of present inputs but also on the previous outputs. Therefore, sequential circuits contain combinational circuits along with memory storage elements.

The sequential circuit involves usage of feedback loops, with the help of this the state of the previous output is recorded. Thus, the next output is controlled by the state of the previous output. The memory present in the sequential circuit keeps the track of the output and the thus, the output is produced.

Various types of sequential circuits are

- flip flops

- Registers

- Counters etc.

BASIS FOR COMPARISON	COMBINATIONAL CIRCUIT	SEQUENTIAL CIRCUIT
Basic	The output is discovered by the present state of the inputs.	Both the present input and past state output are used to identify the output.
Storage capability	Does not store data.	Can store a small amount of data.
Application	Used in adders, encoders, multiplexer, etcetera.	Flip-flop and latches.
Clock	Circuits do not rely on the clock.	Clock is utilized for performing triggering functions.
Feedback	No requirement of the feedback.	Feedback is required.

Difference between combinational and sequential circuits
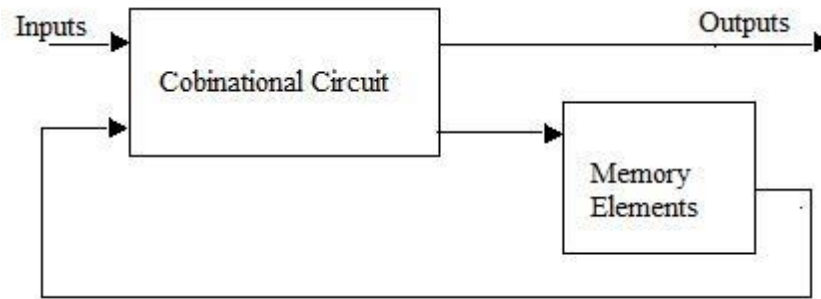
Block Diagram of sequential circuit

Applications are sequential circuits are memory devices, flip flops, registers and counters etc.

Flip Flops: - Flip flops is a sequential circuits which is used to store one bit of binary information. It means if we need to store one bit of information we need one flip flop. To store n bit of binary information we need n number of flip flops.

Various types of flip flops are as follows

1. SR flip flop

2. D flip flop

3. T flip flop

4. JK flip flop

SR flip flop:- The **SR flip-flop**, also known as a *SR Latch*, can be considered as one of the most basic sequential logic circuit. This simple flip-flop is basically a one-bit memory device that has two inputs, one which will "SET" the device (meaning the output = "1"), and is labelled as **S** and another which will "RESET" the device (meaning the output = "0"), labelled as **R**. The the SR description stands for "Set-Reset".

The SR flip-flop can be considered as a 1-bit memory, since it stores the input pulse even after it has passed. Flip-flops (or bi-stables) of different types can be made from logic gates and, as with other combinations of logic gates, the NAND and NOR gates are the most versatile. This is because, as well as being universal, i.e. it can be made to mimic any of the other standard logic functions, it is also cheaper to construct. Other, more widely used types of flip-flop are the JK, the D type and T type, which are developments of the SR flip-flop
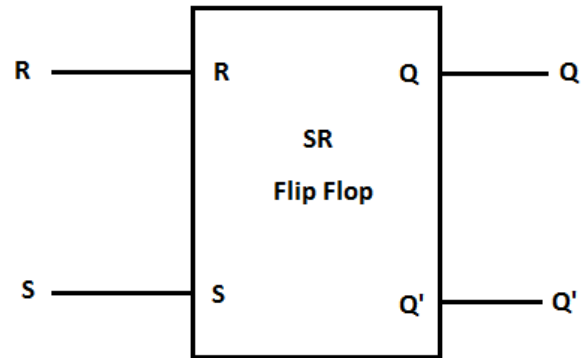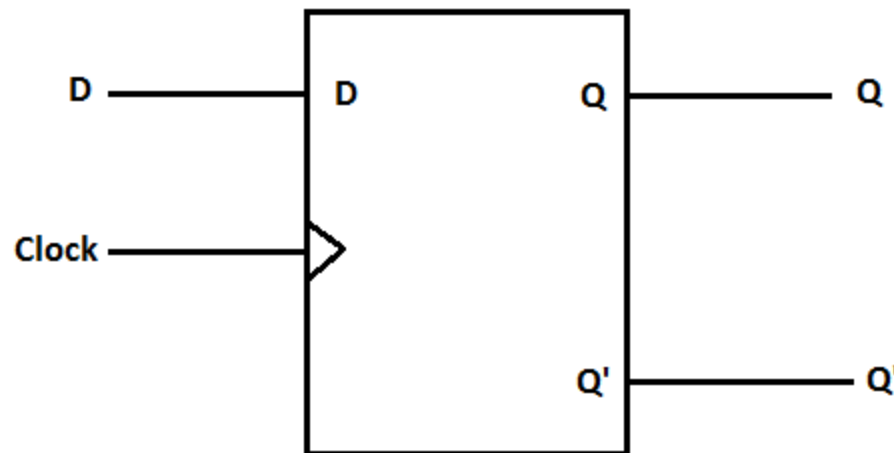
Block diagram of SR flip flop
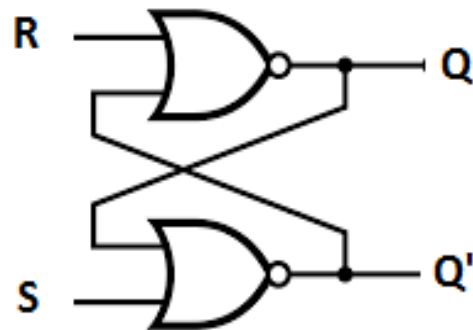
Logic Diagram of SR flip flop

| S	R	Q_{n+1}
0	0	Q_n (No Change)
0	1	0 (Reset)
1	0	1 (Set)
1	1	? (Indeterminate or forbidden)

Truth Table of SR flip flop

D Flip Flop: - There are two disadvantages of SR flip flop, one it needs two inputs (S and R) to store one bit of information, second is, its forbidden condition when S and R both are 1 the output is indeterminate. These disadvantages are removed in D flip flop.

D flip flop is a slight modification of SR flip flop. An SR flip flop is converted in D flip flop by inserting an inverter between S and R inputs and assigning the symbol D to the single input.

Block diagram of D flip flop
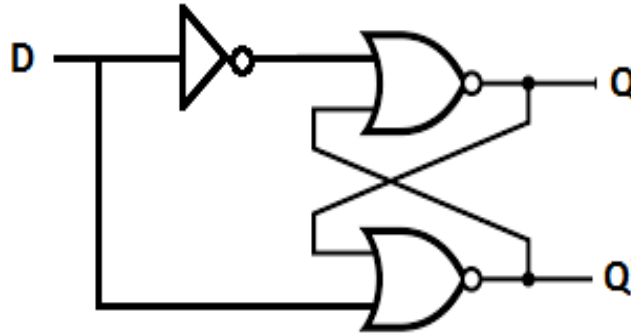
Logic diagram of D flip flop

D	Q_{n+1}
0	0 (Reset)
1	1 (Set)

Truth table of D flip flop

Because there is only one input 'D' in this flip flop, the value of D input may be 0 or 1. When it is 0, the flip flop will store 0, in other words it will reset. When D is 1 (input is 1) the flip flop will store 1 or it will set.

Applications of D flip flop: - Following are the applications of D flip flops.

- Memory Devices

- Registers

- Counters

- Event detectors

T Flip Flop: - T flip flop is one of the sequential circuits. The 'T' in the T Flip flop stands for toggle, so it is also known as Toggle Flip flop or T flip flop. This type of circuits has only one input unit, unlike SR and JK flip flops. when the circuit changes its state from one state output to the complement other state output, then this process is referred to as toggling.
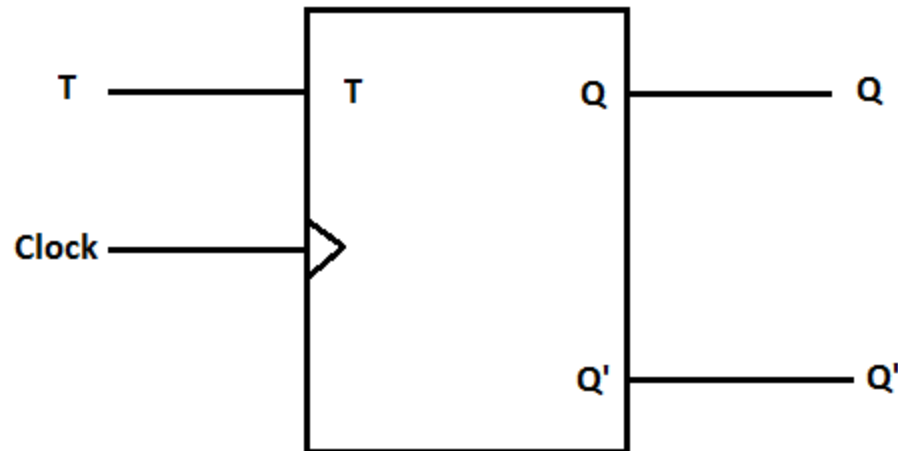
Block diagram of D flip flop
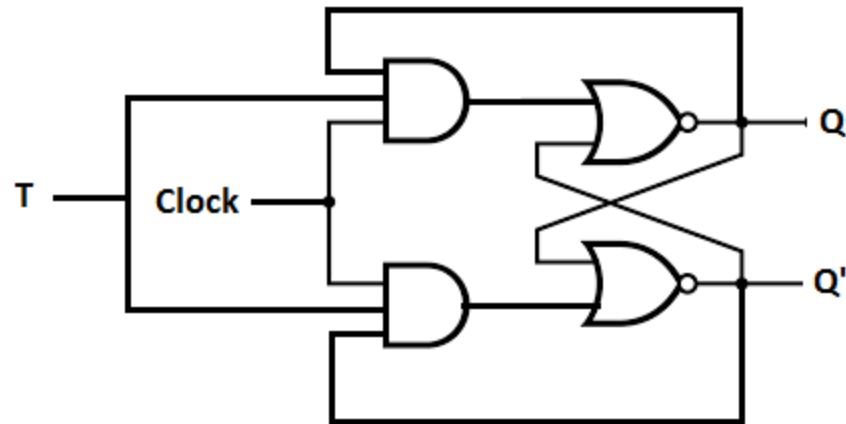
Logic diagram of T flip flop

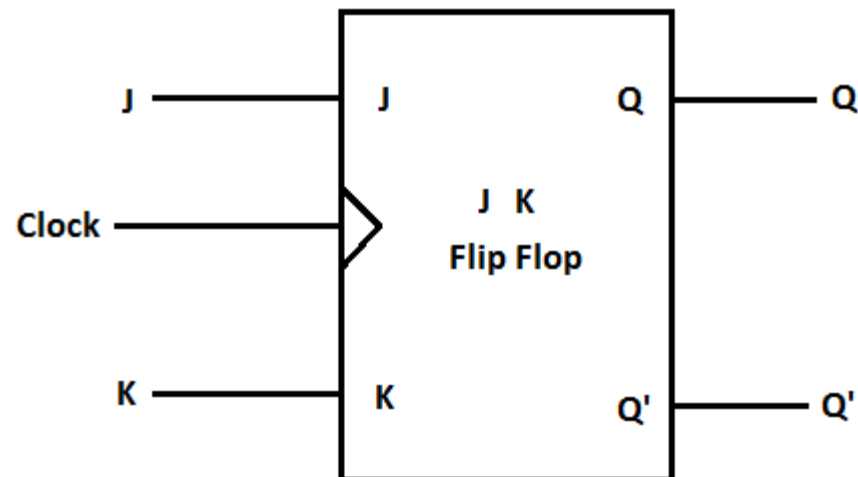| T	Q_{n+1}
0	Q_n (No Change)
1	Q_n' (Toggle)

Truth table of T flip flop

When the input T is o, there is no change in flip flop, means whatever value is inside the flip flop, it will remain same. But when the input T is 1, the value which is inside the flip flop is complemented, means it will be the complement of previous value.
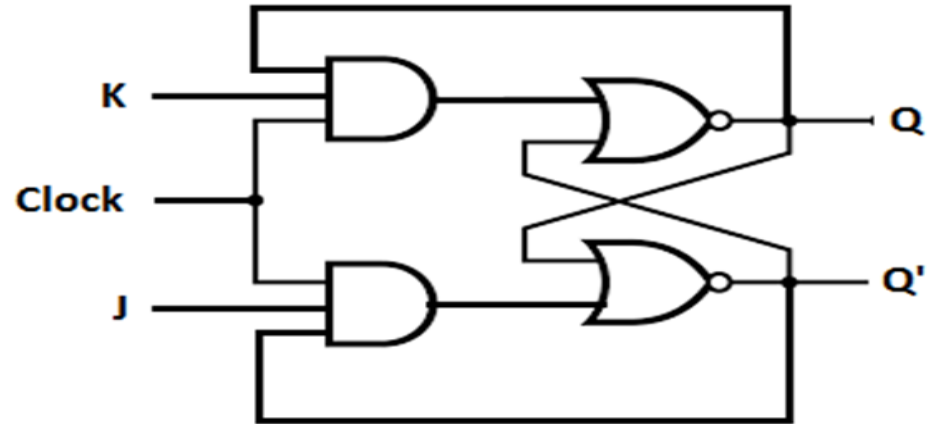
Applications of T flip flop: - Following are the applications of T flip flops.

- It is used in counter designs.

- These flip flops are used for constructing binary counters.

- They are used in frequency dividers.

- This type of sequential circuits is also present in binary addition devices.

- It is also used in 2-bit parallel load registers.

- It also used in shift registers.

JK Flip Flop: -The JK Flip Flop name has been kept on the inventor name of the circuit known as **Jack Kilby**. The **JK Flip Flop** is the most widely used flip flop. It is considered to be a universal flip-flop circuit. The sequential operation of the JK Flip Flop is same as for the RS flip-flop with the same **SET** and **RESET** input. The difference is that the JK Flip Flop does not have the invalid input states (forbidden condition) of the SR flip flop (when S and R are both 1).

```
J ─────────┤ J          Q ├──────── Q
           │   J  K        │
Clock ─────▷│  Flip Flop   │
           │               │
K ─────────┤ K         Q' ├──────── Q'
```

Block diagram of JK flip flop

Logic diagram of JK flip flop

J	K	Q_{n+1}
0	0	Q_n (No Change)
0	1	0 (Reset)
1	0	1 (Set)
1	1	Q_n' (Toggle)

Truth table of T flip flop

The forbidden condition of SR flip flop (S=1 ,R=1) is removed in JK flip flop.

Four cases of truth table of JK flip flop are as follows

1. When J=0 and K=0 in this case whatever value is stored in the flip flop, it will remain the same as it is and it is known as 'no change' (Q_n).

2. When J=0 and K=1, in this case the next new value in the flip flop will be 0 (Reset).

3. When J=1 and K=0, in this case the next new value in the flip flop will be 1 (Set).

4. When J=1 and K=1, the out or the next new value in the flip flop will be the complement of the previous value and it is known as toggle or (Q_n').

   $\mathbf{Q_n}$ represents the previous value and $\mathbf{Q_{n+1}}$ represents the current value.

Applications of JK flip flop: - Following are the applications of JK flip flops.

- Registers

- Counters

- Event Detectors

- Frequency Divider Circuits

Master Slave Flip Flop: - The Master-Slave Flip-Flop is basically a combination of two flip-flops connected together in a series configuration. Out of these, one acts as the **"master"** and the other as a **"slave"**.
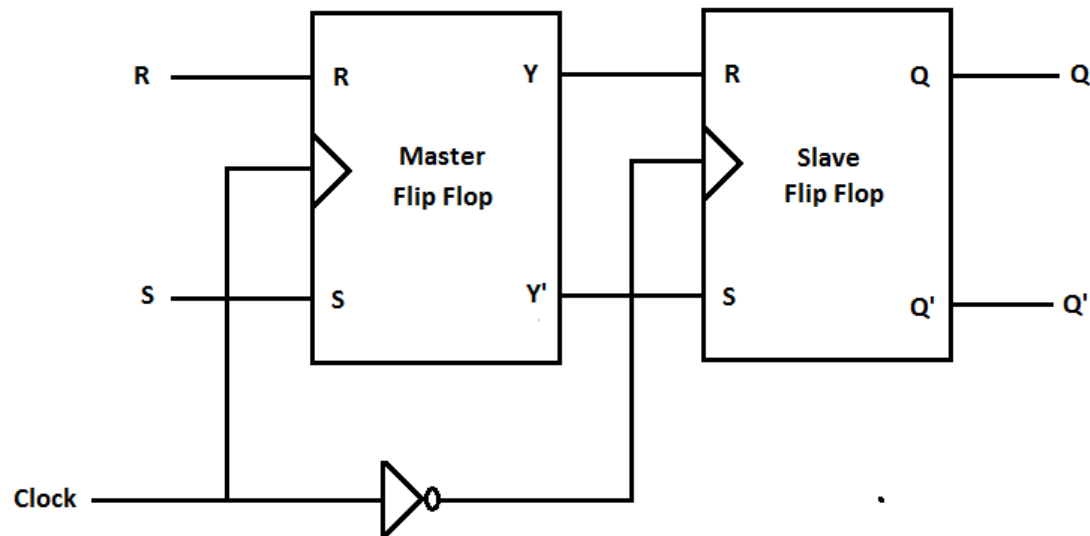
Block Diagram of SR Master Slave Flip Flop
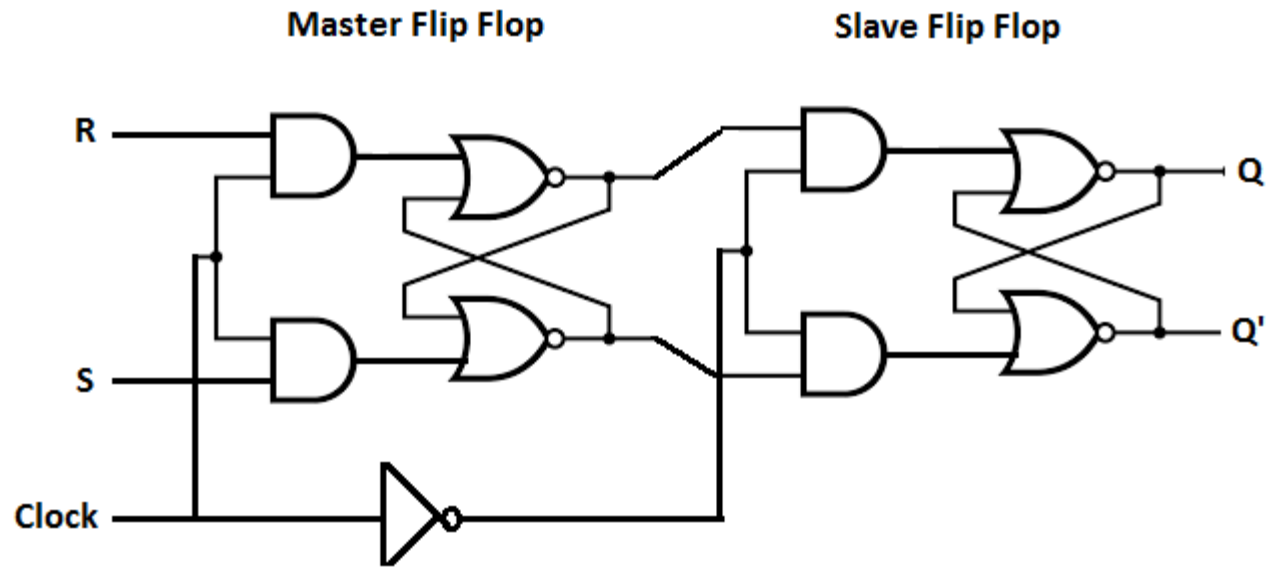
Logic Diagram of SR master Slave Flip Flop

Description: - In a master slave flip flop, two flip flop are connected in a cascaded way. The first one is known as master and the second is known slave. Both of the flip flop share a common clock signal. An inverter is placed between the clock signal. At a time only one flip flop is enabled and the second one is disabled. When the clock of master is 1, it generates output according to the R and S inputs. At this time, because the clock of slave is 0, it stays in no change state.

Next time when the clock of master is 0, it stays in no change position. Because the clock of slave is 1, it is now enabled and it receives the inputs from master and generates output according to the R and S inputs. A slave flip flop always receives inputs from the master flip flop. The truth table of this flip flop is same as RS flip flop. A master slave flip can be constructed with the help of SR and JK flip flops etc.

Unit 1

Topic Name – Registers and Counters

Table of Contents

- Registers

- Counters

- Synchronous Counter

- Asynchronous Counter

- References

Registers: - Flip flops can store a single bit of binary data i.e. 1 or 0. But if we need to store multiple bits of data, we need multiple flip flops. As we know that a single flip flop is used for one bit storage, n flip flops are connected in an order to store n bits of data. In digital electronics, a Register is a device which is used to store the information. Flip flops are used in construction of registers. Register is a group of flip flops used to store multiple bits of data. For example, if a memory is to store 16 bit data, then it needs a set of 16 flip flops. The input and outputs of a register are may be serial or parallel based on the requirements.

Register Load: - The transfer of new information into a register is referred to as loading the register. If all the bits of a register are loaded simultaneously with a common clock pulse transition, we say that the loading is done in **parallel**.
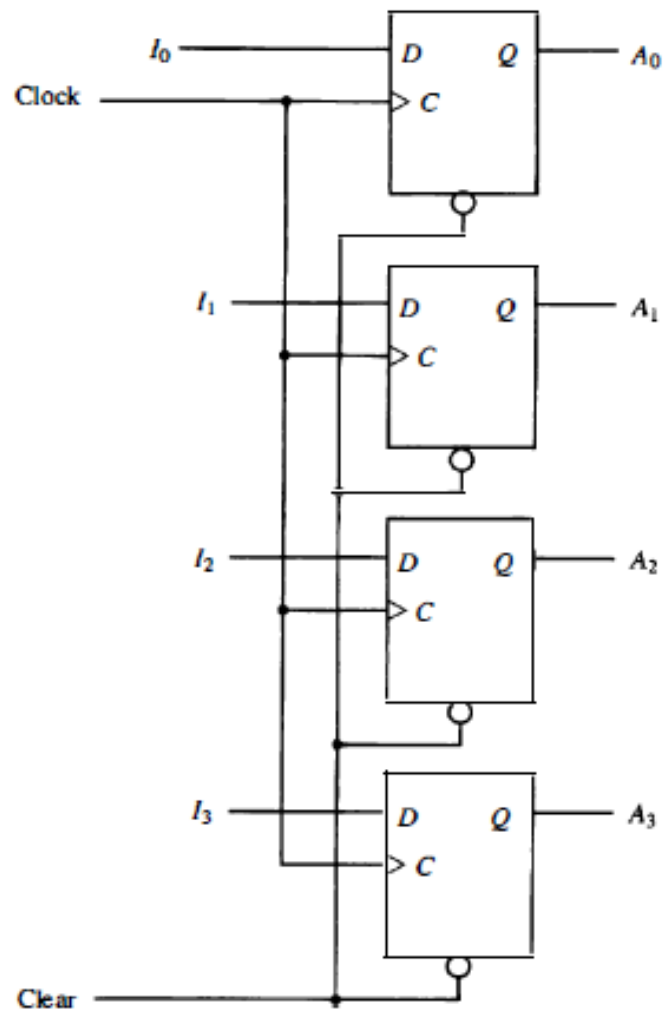
A 4 Bit register

Shift Registers: - A register capable of shifting its binary information in one or both directions is called a shift register. The logical configuration of a shift register consists of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive common clock pulses that initiate the shift from one stage to the next.

The simplest possible shift register is one that uses only flip-flops. The output of a given flip-flop is connected to the D input of the flip-flop at its right. The clock is common to all flip-flops. The serial input determines what goes into the leftmost position during the shift. The serial output is taken from the output of the rightmost flip-flop.
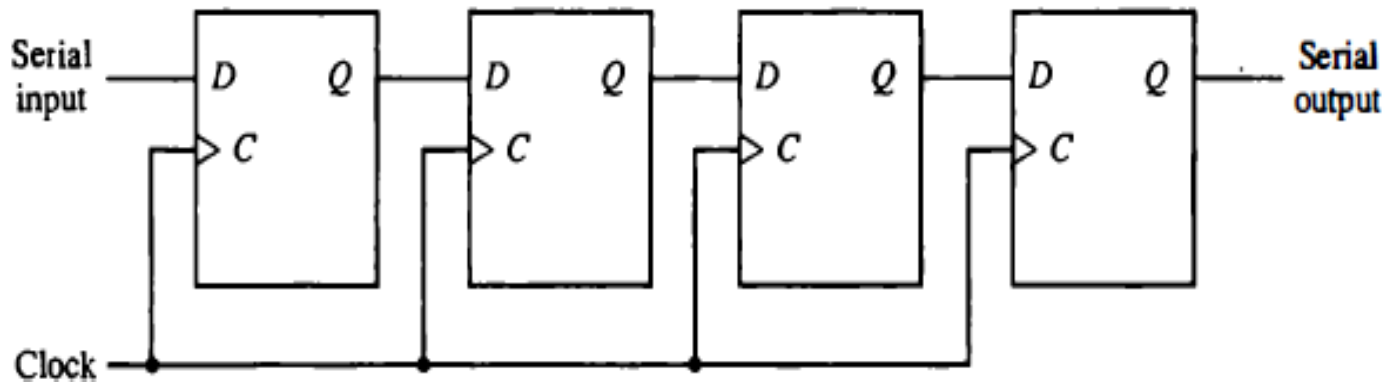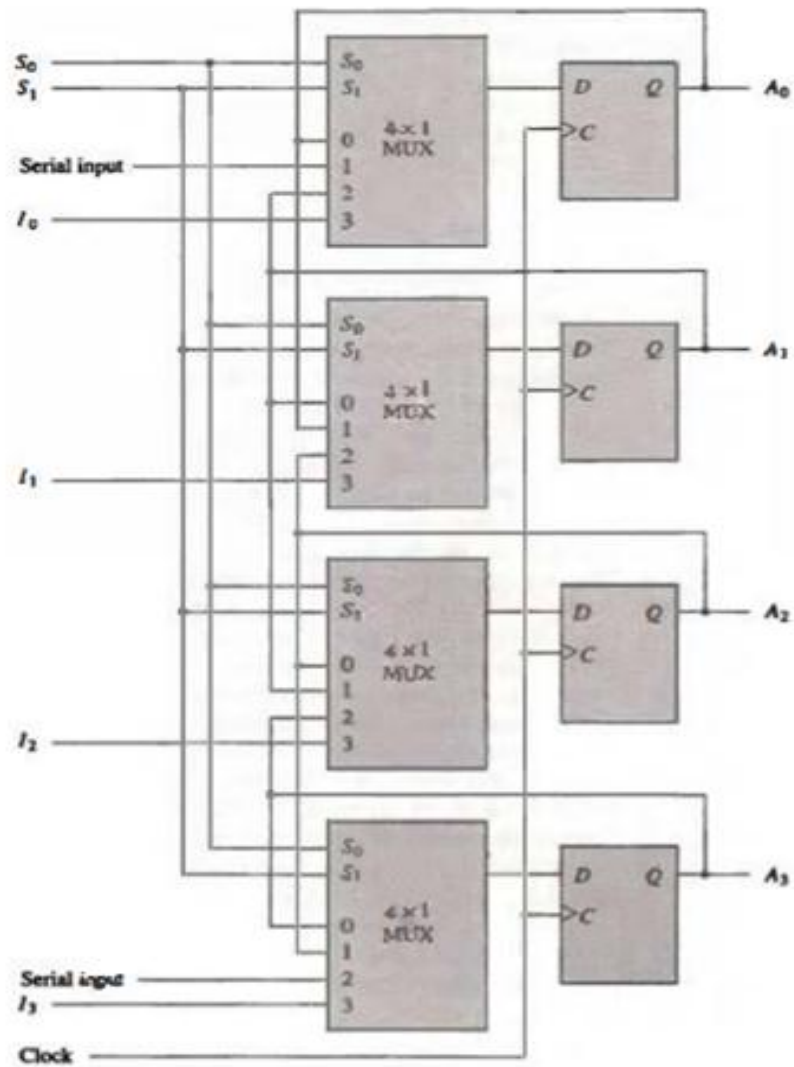
4 Bit Shift Register

Bi Directional shift register: - A register capable of shifting in one direction only is called a unidirectional shift register. A register that can shift in both directions is called a bidirectional shift register.

Bidirectional Shift Register with Parallel Load: - A register capable of shifting in one direction only is called a unidirectional shift register. A register that can shift in both directions is called a bidirectional shift register. A bidirectional shift register with parallel load capability is known as bidirectional shift register with parallel Load or universal register.

Working: - When the mode control $S_1 S_0$ = 0 0, data input 0 of each multiplexer is selected. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock transition transfers into each flip-flop the binary value it held previously, and no change of state occurs. When $S_1 S_0$ = 0 1, the terminal marked 1 in each multiplexer has a path to the D input of the corresponding flip-flop. This causes a shift-right operation. When $S_1 S_0$ = 10 a shift-left operation results.

When $S_1 S_0$ = 1 1 , the binary information from each input 10 through I, is transferred into the corresponding flip-flop, resulting in a parallel load operation.

Mode control		Register operation
S_1	S_0	
0	0	No change
0	1	Shift right (down)
1	0	Shift left (up)
1	1	Parallel load

Function Table

Applications of Shift Registers

Registers are used in digital electronic devices like computers as

- Temporary data storage

- Data transfer

- Data manipulation

- As counters

Counters: - Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.

- Synchronous counters.

In digital logic and computing, a counter is a device which stores the number of times a particular event or process has occurred. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops.

Asynchronous or Ripple Counters: -In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following counters is driven by output of previous flip flops. It means all the flip flops in the counter share the different clock signals.

A binary ripple counter can be constructed using clocked *JK* flip-flops. In asynchronous counter each flip-flop is triggered by the previous flip-flop. The system clock, a square wave, drives flip-flop *A.* The output of *A* drives *B,* and the output of *B* drives flip-flop C. All the J and K inputs are tied to *+Vcc·* This means that each flip-flop will change state (toggle) with a negative transition at its clock input.
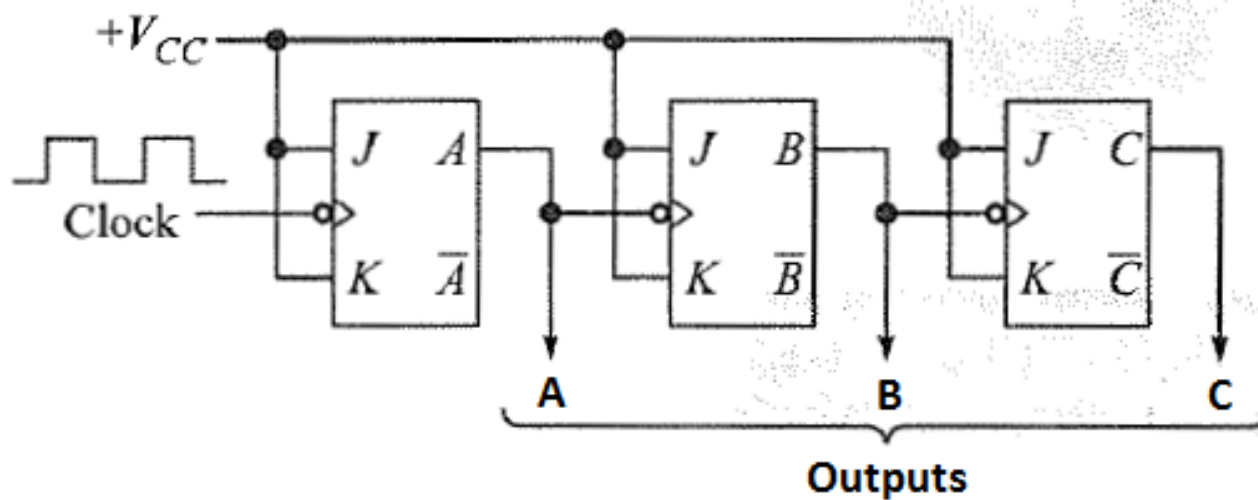
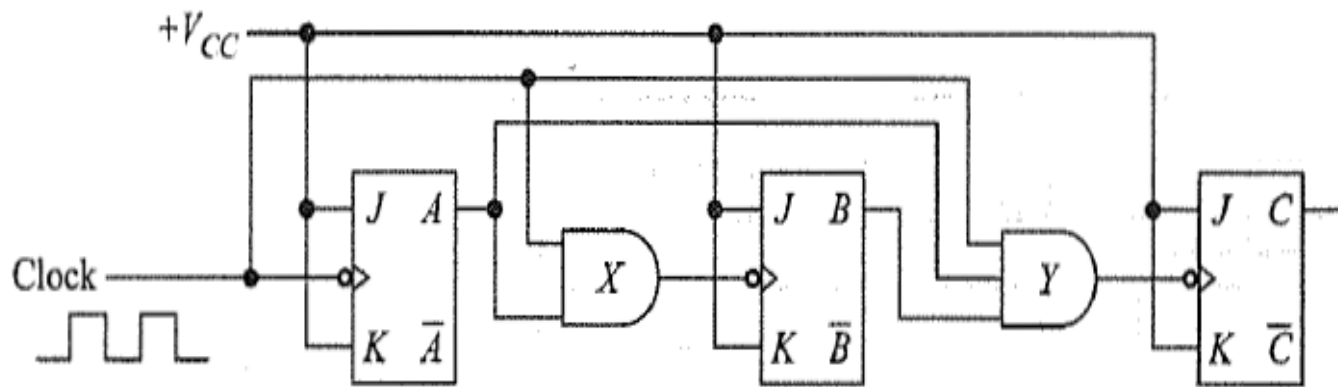**Three bit asynchronous or ripple counter**

C	B	A	State or Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0

Truth table

When the output of a flip-flop is used as the clock input for the next flip-flop, we call the counter a ripple counter, or asynchronous counter. The A flip-flop must change state before it can trigger the B flip-flop, and the B flip-flop has to change state before it can trigger the C flip-flop. The triggers move through the flip-flops like a ripple in water.

Synchronous Counters: - If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter. In other words in a synchronous counter all the flip flops in a counter share a common clock signal.

3 Bit Synchronous Counter

C	B	A	State or Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0

Truth table

Synchronous Vs. Asynchronous counter

SYNCHRONOUS COUNTERS	ASYNCHRONOUS COUNTERS
All flip flops share a common clock signal.	All flip flops share a different clock signal.
The propagation delay is very low	Propagation delay is higher than that of synchronous counters.
These are faster than that of ripple counters.	These are slow in operation.
Large number of logic gates are required to design	Less number of logic gates required.
High cost.	Low cost.
Synchronous circuits are easy to design.	Complex to design.

Applications of counters: - Counter found their applications in many digital electronic devices. Some of their applications are listed below.

- Frequency counters

- Digital clocks

- Analog to digital convertors.

- With some changes in their design, counters can be used as frequency divider circuits. The frequency divider circuit is that which divides the input frequency exactly by '2'.

- In time measurement. That means calculating time in timers such as electronic devices like ovens and washing machines.

- We can design digital triangular wave generator by using counters.

References

- Mano Morris, "Computer System Architecture", PHI

- William Stalling, "Computer Organization & Architecture", Pearson education Asia

- Hamacher vranesic zaky, "Computer Organization", McGraw Hill

- B. Ram, "Computer Fundamental Architecture & Organization", New Age.

- Tannenbaum, "Structured Computer Organization", PHI.