

BASIC ELECTRONICS ENGINEERING

UNIT III DIGITAL ELECTRONICS

Overview of Number Systems, Logic gates including Universal Gates, BCD codes, Excess-3 code, Gray code, Hamming code. Boolean Algebra, Basic Theorems and properties of Boolean Algebra, Truth Tables and Functionality of Logic Gates – NOT, OR, AND, NOR, NAND, XOR and XNOR. Simple combinational circuits–Half and Full Adders. Introduction to sequential circuits, Flip flops, Registers and counters (Elementary Treatment only).

INTRODUCTION:

Electronic circuits and systems are of two kinds. They are **analog** and **digital**. The distinction between them is not so much in the types of semiconductor devices used in these circuits as it is in voltage and current variations that occur when each type of circuit performs the function for which it is designed.

Analog circuits are those in which voltages and currents vary continuously through the given range. They can take infinite values within the specified range.

Digital circuit is one in which the voltage levels assume a finite number of distinct values. In virtually all modern digital circuits, there are just two discrete voltage levels.

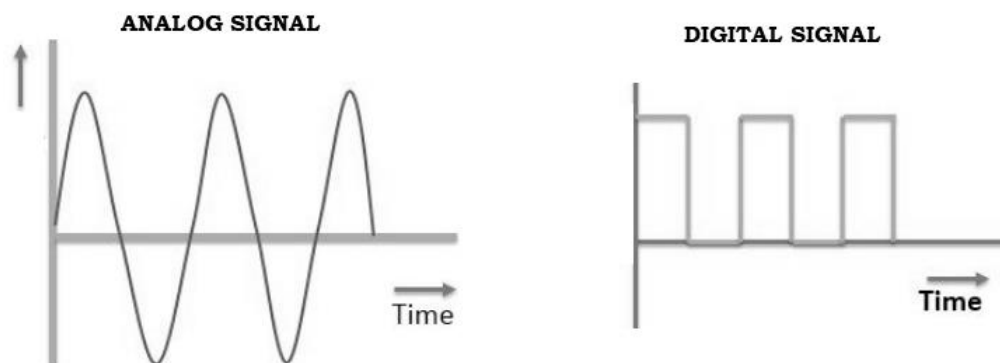
Digital circuits are also called logic circuits, because each type of digital circuit obeys a certain set of logic rules. The manner in which a logic circuit responds to an input is referred to as the circuit's logic.

Digital systems are used extensively in computation and data processing, control systems, communications and measurement. Digital systems have a number of advantages over analog systems.

ADVANTAGES OF DIGITAL SYSTEMS OVER ANALOG SYSTEMS

- Digital systems are easier to design
- These are small in size and simple operation (LOW or HIGH)
- Information storage is easy
- Accuracy and precision are greater
- Digital systems are more versatile
- Digital circuits are less affected by noise, temperature etc.,
- Reliability is more
- Requires less space & cost is low (Chips).
- Speed is high.

Example:



BASIC ELECTRONICS ENGINEERING

Applications of Digital systems:

- Cameras
- Recorders(Audio & Video)
- Telephone systems
- Traffic Lights
- Computers, TVs, DVDs etc.,
- Elevator Display
- Digital thermo meter
- Digital watch

TYPES OF NUMBER SYSTEMS

- Decimal Number System
- Binary Number System
- Octal Number System
- Hexadecimal Number System

i) DECIMAL NUMBER SYSTEM

The decimal number system contains ten unique symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Since counting in decimal involves ten symbols, we say that its **base or radix** is ten. There is no symbol for its base, i.e. for ten. In this system, any number (integer, fraction, or mixed) of any magnitude can be represented by the use of these ten symbols only. Each symbol in the number is called a **digit**.

Example:

When we write a decimal number say 5678.9, we know it can be represented as

$$5000+600+70+8+0.9=5678.9$$

The decimal number 5678.9 can also be written as $(5678.9)_{10}$, where the 10 subscript indicates the radix or base.

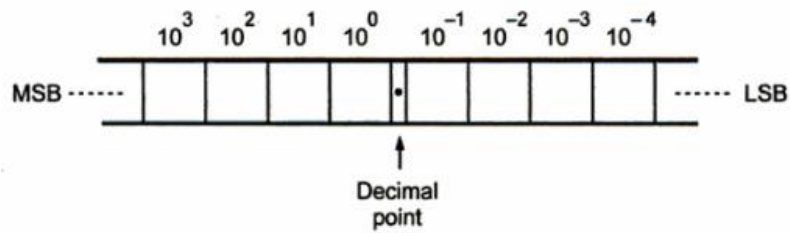
In power of 10, we can write as

$$\boxed{5 \times 10^3} + \boxed{6 \times 10^2} + \boxed{7 \times 10^1} + \boxed{8 \times 10^0} + \boxed{9 \times 10^{-1}}$$

5 6 7 8 . 9

This says that, the position of a digit with reference to the decimal point determines its value/weight. The sum of all the digits multiplied by their weights gives the total number being represented. The leftmost digit, which has the greatest weight is called the most significant digit/bit(MSD or MSB) and the rightmost digit, which has the least weight, is called the least significant digit/bit(LSD or LSB).

BASIC ELECTRONICS ENGINEERING



In general, the value of any mixed decimal number

$$d_n d_{n-1} d_{n-2} \dots d_1 d_0 . d_{-1} d_{-2} d_{-3} \dots d_{-k}$$

is given by

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + \dots + (d_1 \times 10^1) + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \dots$$

EX:

$$\begin{aligned}
 9256.26 &= 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100) \\
 &= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}
 \end{aligned}$$

ii) BINARY NUMBER SYSTEM

The binary number system is a positional weighted system. The base or radix of this number system is 2. Hence, it has two independent symbols. The base itself cannot be a symbol. The symbols used are **0** and **1**. A binary digit is called a **bit**. A binary number consists of a sequence of bits, each of which is either 0 or 1.

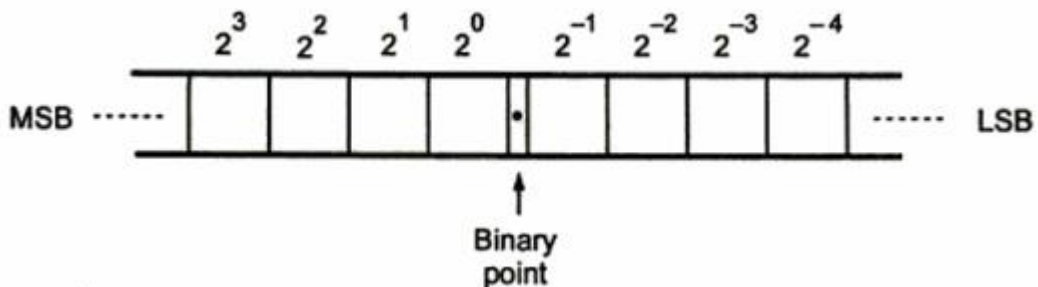
In general, a binary number with an integer part of $(n + 1)$ bits and a fraction part of 'k' bits can be written as

$$d_n d_{n-1} d_{n-2} \dots d_1 d_0 . d_{-1} d_{-2} d_{-3} \dots d_{-k}$$

Its decimal equivalent is

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

However in binary system, weight is expressed as a power of 2 as shown in Fig.



EX: Represent the binary number 1101.101 in power of 2 and find its decimal equivalent.

SOL:

BASIC ELECTRONICS ENGINEERING

Representing given binary number in power of 2 we have,

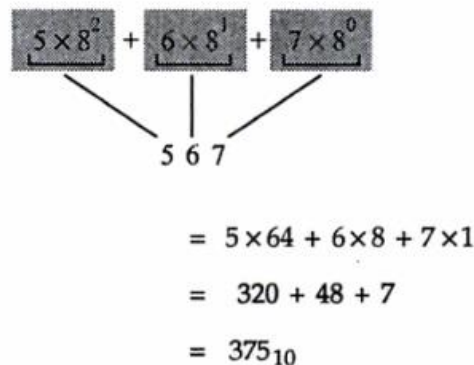
$$\begin{aligned} N &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125 \\ &= 13.625_{10} \end{aligned}$$

iii) OCTAL NUMBER SYSTEM

We know that the base of the decimal number system is 10 because it uses the digits 0 to 9, and the base of binary number system is 2 because it uses digits 0 and 1. The octal number system uses first eight digits of decimal number system : 0, 1, 2, 3, 4, 5, 6, and 7. As it uses 8 digits, its base is 8.

EX: Represent octal number 567 in power of 8 and find its decimal equivalent.

The given octal number 567 can be represented in power of 8 as


$$\begin{aligned} &= 5 \times 64 + 6 \times 8 + 7 \times 1 \\ &= 320 + 48 + 7 \\ &= 375_{10} \end{aligned}$$

iv) HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system has a base of 16 having 16 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. It is another number system that is particularly useful for human communications with a computer. Although it is somewhat more difficult to interpret than the octal number system, it has become the most popular. Since its base is a power of 2 (2^4), it is easy to convert hexadecimal numbers to binary and vice versa.

The following table shows the relationship between decimal, binary and hexadecimal. Note that each hexadecimal digit represents a group of four binary digits, called **nibbles**, which are fundamental parts of larger binary words.

BASIC ELECTRONICS ENGINEERING

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

EX: Represent the hexadecimal number 3FD in power of 16 and find its decimal equivalent.
SOL:

The given hexadecimal number $3FD_{16}$ can be represented in power of 16.

$$\begin{array}{c} \boxed{3 \times 16^2} + \boxed{F \times 16^1} + \boxed{D \times 16^0} \\ \swarrow \quad \downarrow \quad \searrow \\ 3 \quad F \quad D \end{array}$$

$$\begin{aligned} &= 3 \times 256 + F(15) \times 16 + D(13) \times 1 \\ &= 768 + 240 + 13 \\ &= 1021_{10} \end{aligned}$$

Counting in Radix or Base (r)

We know that the number systems with radix (base) 'r' equal to 10, 2, 8 and 16. Each number system has 'r' set of characters. For example, in decimal number system 'r' equals to 10 has 10 characters from 0 to 9, in binary number system 'r' equals to 2 has 2 characters 0 and 1 and so on. In general we can say that, a number represented in radix r, has r characters in its set and r can be any value. This is illustrated in the following Table.

BASIC ELECTRONICS ENGINEERING

Radix (Base) r	Characters in set
2 (Binary)	0, 1
3	0, 1, 2
4	0, 1, 2, 3
:	
:	
7	0, 1, 2, 3, 4, 5, 6
8 (Octal)	0, 1, 2, 3, 4, 5, 6, 7
:	
:	
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
:	
:	
16 (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

NOTE :

The classification of individual bits into larger groups is generally referred to by the following more common names of:

Number of Binary Digits (bits)	Common Name
1	Bit
4	Nibble
8	Byte
16	Word
32	Double Word
64	Quad Word

BASIC ELECTRONICS ENGINEERING

NUMBER BASE CONVERSIONS

CASE-(i): Binary to Decimal Conversion

1) Convert 10101_2 to decimal

Positional weights $2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$

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

2) Convert 11011.101_2 to decimal

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

$$\begin{aligned}1 \ 1 \ 0 \ 1 \ 1 \cdot 1 \ 0 \ 1 &= (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &\quad + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= 16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\ &= 27.625_{10}\end{aligned}$$

CASE-(ii): Octal to Decimal Conversion

3) Convert 4057.06_8 to decimal

$$\begin{aligned}4057.06_8 &= 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2} \\ &= 2048 + 0 + 40 + 7 + 0 + 0.0937 \\ &= 2095.0937_{10}\end{aligned}$$

4) Convert 172_8 to decimal

$$\begin{array}{ccc}1 & 7 & 2 \\ 8^2 & 8^1 & 8^0 \\ \text{Weight} &= 1 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 \\ &= 1 \times 64 + 7 \times 8 + 2 \times 1 \\ &= 122_{10}\end{array}$$

CASE-(iii): Hexadecimal to Decimal Conversion

5) Convert $5C7_{16}$ to decimal

Multiply each digit of 5C7 by its position weight and add the product terms.

$$\begin{aligned}5C7_{16} &= (5 \times 16^2) + (12 \times 16^1) + (7 \times 16^0) \\ &= 1280 + 192 + 7 \\ &= 1479_{10}\end{aligned}$$

6) Convert $A0F9.0EB_{16}$ to decimal

BASIC ELECTRONICS ENGINEERING

$$\begin{aligned}A0F9.0EB_{16} &= (10 \times 16^3) + (0 \times 16^2) + (15 \times 16^1) \\&\quad + (9 \times 16^0) + (0 \times 16^{-1}) + (14 \times 16^{-2}) + (11 \times 16^{-3}) \\&= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026 \\&= 41209.0572_{10}\end{aligned}$$

CASE-(iv): Decimal to Binary Conversion

In this method, the decimal integer number is converted to the binary integer number by successive division by 2, and the decimal fraction is converted to binary fraction by successive multiplication by 2. This is also known as the **double-dabble method**. In the successive division-by-2 method, the given decimal integer number is successively divided by 2 till the quotient is zero. The last remainder is the MSB. The remainders read from bottom to top give the equivalent binary integer number. In the successive multiplication-by-2 method, the given decimal fraction and the subsequent decimal fractions are successively multiplied by 2, till the fraction part of the product is 0 or till the desired accuracy is obtained. The first integer obtained is the MSB. Thus, the integers read from top to bottom give the equivalent binary fraction.

To convert a mixed number to binary, convert the integer and fraction parts separately to binary and then combine them.

7) Convert 52_{10} to binary

<i>Successive division</i>		<i>Remainder</i>	
2	52		
2	26		0
2	13		0
2	6		1
2	3	↑	0
2	1		1
	0		1

Reading the remainders from bottom to top, the result is $52_{10} = 110100_2$.

BASIC ELECTRONICS ENGINEERING

8) Convert 105.15_{10} to binary

Conversion of integer 105

Successive division	Remainder
2 105	
2 52	1
2 26	0
2 13	0
2 6	1
2 3	↑ 0
2 1	1
0	1

Reading the remainders from bottom to top, $105_{10} = 1101001_2$.

Conversion of fraction 0.15_{10}

Given fraction	0.15
Multiply 0.15 by 2	0.30
Multiply 0.30 by 2	0.60
Multiply 0.60 by 2	1.20
Multiply 0.20 by 2	↓ 0.40
Multiply 0.40 by 2	0.80
Multiply 0.80 by 2	1.60

This particular fraction can never be expressed exactly in binary. This process may be terminated after a few steps.

Reading the integers from top to bottom, $0.15_{10} = 0.001001_2$.

Therefore, the final result is, $105.15_{10} = 1101001.001001_2$.

BASIC ELECTRONICS ENGINEERING

CASE-(v): Decimal to Octal Conversion

9) Convert 378.93_{10} to Octal

Conversion of 378_{10} to octal

Successive division

8	378
8	47
8	5
	0

Remainders

↑	2
	7
	5

Read the remainders from bottom to top. Therefore, $378_{10} = 572_8$.

Conversion of 0.93_{10} to octal

0.93×8		7.44
0.44×8		3.52
0.52×8	↓	4.16
0.16×8		1.28

Read the integers to the left of the octal point downwards.

Therefore, $0.93_{10} = 0.7341_8$. Hence $378.93_{10} = 572.7341_8$.

CASE-(vi): Decimal to Hexadecimal Conversion

10) Convert 2598.675_{10} to Hexadecimal

Conversion of 2598_{10}

Successive division

16	2598
16	162
16	10
	0

Remainder

<i>Decimal</i>	<i>Hex</i>
6	↑ 6
2	2
10	A

Reading the remainders upwards, $2598_{10} = A26_{16}$.

BASIC ELECTRONICS ENGINEERING

Conversion of 0.675_{10}
Given fraction is 0.675

$$\begin{array}{r|l} 0.675 \times 16 & 10.8 \\ 0.800 \times 16 & 12.8 \\ 0.800 \times 16 & 12.8 \\ 0.800 \times 16 & \downarrow 12.8 \end{array}$$

Reading the integers to the left of hexadecimal point downwards, $0.675_{10} = 0.ACCC_{16}$.
Therefore, $2598.675_{10} = A26.ACCC_{16}$.

CASE-(vii): Binary to Octal Conversion

To convert a binary number to an octal number, starting from the binary point make groups of 3 bits ($8=2^3$) each, on either side of the binary point and replace each 3-bit binary group by the equivalent octal digit.

11)

Convert 110101.101010_2 to octal.

SOL:

$$\begin{array}{l} \text{Groups of three bits are} \quad 110 \quad 101 \quad . \quad 101 \quad 010 \\ \text{Convert each group to octal} \quad 6 \quad 5 \quad . \quad 5 \quad 2 \\ \text{The result is} \quad \quad \quad 65.52_8 \end{array}$$

12)

Convert 10101111001.0111_2 to octal.

SOL:

$$\begin{array}{l} \text{Groups of three bits are} \quad 10 \quad 101 \quad 111 \quad 001 \quad . \quad 011 \quad 1 \\ = \quad 010 \quad 101 \quad 111 \quad 001 \quad . \quad 011 \quad 100 \\ \text{Convert each group into octal} \quad 2 \quad 5 \quad 7 \quad 1 \quad . \quad 3 \quad 4 \\ \text{The result is} \quad \quad \quad 2571.34_8 \end{array}$$

CASE-(viii): Binary to Hexadecimal Conversion

To convert a binary number to a hexadecimal number, starting from the binary point make groups of 4 bits ($16=2^4$) each, on either side of the binary point and replace each 4-bit binary group by the equivalent octal digit.

13)

Convert 1011011011_2 to hexadecimal.

SOL:

Make groups of 4 bits, and replace each 4-bit group by a hex digit.

BASIC ELECTRONICS ENGINEERING

Given binary number is 1011011011
Groups of four bits are 0010 1101 1011
Convert each group to hex 2 D B
The result is 2DB₁₆

14)

Convert 0101111011.011111₂ to hexadecimal.

SOL:

Given binary number is 0101111011.011111
Groups of four bits are 0010 1111 1011 . 0111 1100
Convert each group to hex 2 F B . 7 C
The result is 2FB.7C₁₆

CASE-(ix): Octal to Binary Conversion

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

15) Convert 367.52₈ to binary

SOL:

Given octal number is 3 6 7 . 5 2
Convert each octal digit to binary 011 110 111 . 101 010
The result is 011110111 . 101010₂

16) Convert 777₈ to binary

SOL:

$$(777)_8 = (111111111)_2$$

CASE-(x): Hexadecimal to Binary Conversion

To convert a given hexadecimal number to a binary number, just replace each hex digit by its 4-bit binary group.

17) Convert 4BAC₁₆ to binary

Given hex number is 4 B A C
Convert each hex digit to 4-bit binary 0100 1011 1010 1100
The result is 0100101110101100₂

18) Convert 3A9E.B0D₁₆ to binary

Given hex number is 3 A 9 E . B 0 D
Convert each hex digit to 4-bit binary 0011 1010 1001 1110 . 1011 0000 1101
The result is 0011101010011110.101100001101₂

BASIC ELECTRONICS ENGINEERING

CASE-(xi): Octal to Hexadecimal Conversion

To convert an octal number to hexadecimal the simplest way is to first convert the given octal number to binary and then binary number to hexadecimal.

19) Convert 756.603₈ to hexadecimal

SOL:

Given octal number is	7	5	6	.	6	0	3
Convert each octal digit to binary	111	101	110	.	110	000	011
Groups of four bits are	0001	1110	1110	.	1100	0001	1000
Convert each four-bit group to hex	1	E	E	.	C	1	8
The result is	1EE.C18 ₁₆						

CASE-(xii): Hexadecimal to Octal Conversion

To convert a hexadecimal to an octal number the simplest way is to first convert the given hexadecimal number to binary and then binary number to octal.

20) Convert B9F.AE₁₆ to Octal

Given hex number is	B	9	F	.	A	E		
Convert each hex digit to binary	1011	1001	1111	.	1010	1110		
Groups of three bits are	101	110	011	111	.	101	011	100
Convert each three-bit group to octal	5	6	3	7	.	5	3	4
The result is	5637.534 ₈							

BINARY CODED DECIMAL (BCD)

(or) (NATURAL BCD CODE) (or) 8421 BCD CODE

BCD is a numeric code in which each digit of a decimal number is represented by a separate group of bits. The most common BCD code is 8-4-2-1 BCD, in which each decimal digit is represented by a 4-bit binary number. It is called 8-4-2-1 BCD because the weights associated with 4 bits are 8-4-2-1 from left to right. This means that, bit 3 has weight 8, bit 2 has weight 4-bit 1 has weight 2 and bit 0 has weight 1.

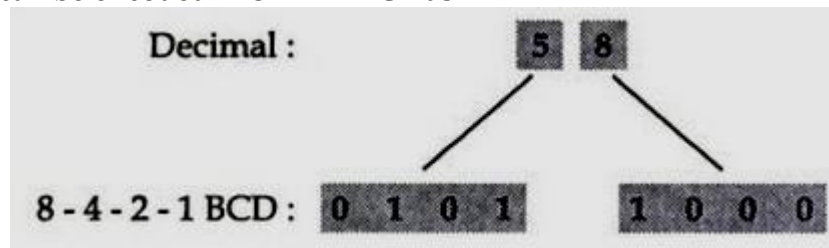
The following Table shows the 4-bit 8-4-2-1 BCD code used to represent a single decimal digit. The 8-4-2-1 BCD code is so widely used that it is common practice to refer to it simply as BCD.

Decimal	BCD Code			
Digit	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

BASIC ELECTRONICS ENGINEERING

In multi digit coding, each decimal digit is individually coded with 8-4-2-1 BCD code.

EX: 58 in decimal can be encoded in 8-4-2-1 BCD as :



As seen from the above example, in multi-digit coding of 8-4-2-1 BCD numbers we require 4-bits per decimal digit. Therefore, total 8-bits are required to encode 58_{10} in 8-4-2-1 BCD. When we represent the same number (58) in binary : 111010_2 , we require only 6 digits. This means that, for representing numbers, 8-4-2-1 BCD is less efficient than pure binary number system.

The **advantage** of a BCD code is that it is easy to convert between it and decimal. The principle **disadvantage** of a BCD, besides its low efficiency, is that arithmetic operations are more complex than they are in pure binary.

EXCESS THREE (XS-3) CODE

Excess-3 code is a modified form of a BCD number. The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.

For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). It is a non-weighted code.

The following table shows excess-3 codes to represent single decimal digit. It is sequential code because we get any code word by adding binary 1 to its previous code word as shown in the table.

<i>Decimal Digit</i>	<i>Binary</i>	<i>Excess-3 code (XS3)</i>
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

We have seen that in BCD subtraction we have to compute 9's (or 10's) complement of the number before subtraction. In excess-3 code we get 9's complement of a number by just complementing each bit. Due to this excess-3 code is called **self-complementing code or reflective code**.

BASIC ELECTRONICS ENGINEERING

GRAY CODE (REFLECTIVE CODE)

Gray code is a non-weighted code & is not suitable for arithmetic operations. It is not a BCD code. It is a cyclic code because successive code words in this code differ in one bit position only i.e, it is a unit distance code. It is also a reflective code i.e, it is both reflective & unit distance.

The 'n' least significant bits for 2^n through $2^{n+1}-1$ are the mirror images of those for 0 through 2^n-1 . An N bit gray code can be obtained by reflecting an N-1 bit code about an axis at the end of the code, & putting the MSB of 0 above the axis & the MSB of 1 below the axis. The reflection of gray codes is shown in the following table.

Gray Code				Decimal	4-bit binary
1-bit	2-bit	3-bit	4-bit		
0	00	000	0000	0	0000
1	01	001	0001	1	0001
	11	011	0011	2	0010
	10	010	0010	3	0011
		110	0110	4	0100
		111	0111	5	0101
		101	0101	6	0110
		100	0100	7	0111
			1100	8	1000
			1101	9	1001
			1111	10	1010
			1110	11	1011
			1010	12	1100
			1011	13	1101
			1001	14	1110
			1000	15	1111

Applications:

Gray codes are used in instrumentation and data acquisition systems where linear or angular displacement is measured.

They are also used in shaft encoders, I/O devices, A/D converters and other peripheral equipment.

(i) Binary to Gray code conversion:

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

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

Where the symbol \oplus stands for the Exclusive OR (X-OR) operation explained below.
The conversion procedure is as follows:

BASIC ELECTRONICS ENGINEERING

1. Record the MSB of the binary as the MSB of the Gray code.
2. Add the MSB of the binary to the next bit in binary, recording the sum and ignoring the carry, if any. i.e. X-OR the bits. This sum is the next bit of the Gray code.
3. Add the 2nd bit of the binary to the 3rd bit of the binary, the 3rd bit to the 4th bit, and so on.
4. Record the successive sums as the successive bits of the Gray code until all the bits of the binary number are exhausted.

Example:

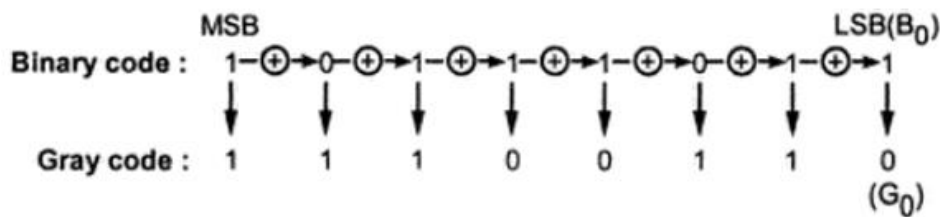
1) Convert the binary 1001 to the gray code.

SOL:

$$\begin{array}{rcll}
 \text{(a)} & \text{Binary} & 1 & \xrightarrow{\oplus} 0 \xrightarrow{\oplus} 0 \xrightarrow{\oplus} 1 \\
 & & || & || \quad || \quad || \\
 & \text{Gray} & 1 & 1 \quad 0 \quad 1 \\
 \\
 \text{(b)} & \text{Binary} & & 1\ 0\ 0\ 1 \\
 & \text{Shifted binary} & & \underline{1\ 0\ 0\ 1} \\
 & \text{Gray} & & 1\ 1\ 0\ 1
 \end{array}$$

2) Convert 10111011 in binary in to gray code.

SOL:



(ii) Gray to Binary code conversion:

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

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

The conversion procedure is:

1. The MSB of the binary number is the same as the MSB of the Gray code number: record it.
2. Add the MSB of the binary to the next significant bit of the Gray code. i.e. X-OR them: record the sum and ignore the carry.
3. Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code, and so on, each time recording the sum and ignoring the carry.
4. Continue this till all the Gray bits are exhausted. The sequence of bits that has been written down is the binary equivalent of the Gray code number.

Example:

3) Convert the gray code 1101 into binary.

SOL:

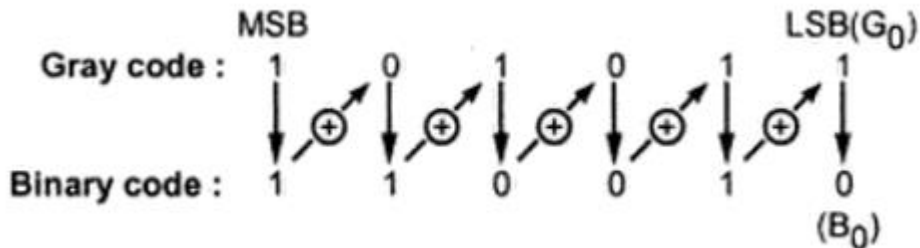
BASIC ELECTRONICS ENGINEERING

The conversion is done as shown below:

Gray	1	1	0	1
	⊗	⊗	⊗	
Binary	1	0	0	1

4) Convert the gray code 101011 into binary.

SOL:



5)

(a) Convert the following into the Gray number.

(i) $3A7_{16}$ (ii) 527_8 (iii) 652_{10}

(b) Convert the Gray number 10110010 into

(i) hex (ii) octal (iii) decimal

Solution

(a) To convert the given number in any system into Gray number first convert it into binary and then convert that binary number into the Gray code using the normal procedure.

(i) $3A7_{16} = 0011, 1010, 0111_2 = 1001110100$ (Gray)

(ii) $527_8 = 101, 011, 011_2 = 111110110$ (Gray)

(iii) $652_{10} = 1010001100_2 = 1111001010$ (Gray)

(b) To convert the given Gray number into any other number system, first convert the given Gray number into a binary number and then convert that binary number into the required number system.

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

ERROR DETECTING CODES

When the digital information in binary form is transmitted & processed from one circuit to another, an error may occur due to noise. The error may be single bit change i.e. 0 changed to 1 or 1 changed to 0. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

PARITY BIT:

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1's is either odd or even. The message, including the parity bit is transmitted and then checked at the receiving end for errors.

BASIC ELECTRONICS ENGINEERING

An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a **parity generator** and the circuit that checks the parity in the receiver is called a **parity checker**.

In even parity, the added parity bit will make the total number of 1's is an even amount. In odd parity the added parity bit will make the total number of 1's is an odd amount.

The following table shows the 3-bit message with even parity and odd parity.

3-bit message			Message with odd parity		Message with even parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	0 0 0	1	0 0 0	0
0	0	1	0 0 1	0	0 0 1	1
0	1	0	0 1 0	0	0 1 0	1
0	1	1	0 1 1	1	0 1 1	0
1	0	0	1 0 0	0	1 0 0	1
1	0	1	1 0 1	1	1 0 1	0
1	1	0	1 1 0	1	1 1 0	0
1	1	1	1 1 1	0	1 1 1	1

When the digit data is received, a parity checking circuit generates an error signal if the total no of 1's is even in an odd parity system or odd in an even parity system. This parity check can always detect a single bit error but cannot detect 2 or more errors with the same word. Odd parity is used more often than even parity because even parity does not detect the situation where all 0's are created by a short circuit or some other fault condition.

EXAMPLE:

1) In an even parity scheme, which of the following words contain error?

(a) 10101010 (b) 11110110 (c) 10111001

SOL:

(a) The number of 1s in the word is even (4). Therefore, there is no error.

(b) The number of 1s in the word is even (6). Therefore, there is no error.

(c) The number of 1s in the word is odd (5). So, this word has an error.

2) In an odd parity scheme, which of the following words contain error?

(a) 10110111 (b) 10011010 (c) 11101010

SOL:

(a) The number of 1s in the word is even (6). So, this word has an error.

(b) The number of 1s in the word is even (4). So, this word has an error.

(c) The number of 1s in the word is odd (5). Therefore, there is no error.

BASIC ELECTRONICS ENGINEERING

ERROR CORRECTING CODES

A code is said to be an error-correcting code, if the code word can always be deduced from an erroneous word. For a code to be a single bit error correcting code, the minimum distance of that code must be three. The minimum distance of that code is the smallest no. of bits by which any two code words must differ. A code with minimum distance of three can't only correct single bit errors but also detect (but can't correct) two bit errors. The key to error correction is that it must be possible to detect & locate erroneous digits. If the location of an error has been determined, then by complementing the erroneous digit, the message can be corrected,

One type of error correcting code is the Hamming code.

These codes not only detect error but also correct them. These error correcting codes are used normally in satellite communication, memories, networking, hard disk, CDROM, DVD etc.

HAMMING CODE:

The Hamming code which is named after 'Richard Hamming' who invented it in 1950. It is widely used for error correction in digital data communications and computer memory.

Hamming code not only provides the detection of a bit error, but also identifies which bit is in error so that it can be corrected. Thus Hamming code is called error detecting and correcting code. The code uses a number of parity bits (dependent on the number of information bits) located at certain positions in the code group.

PROCEDURE FOR WRITING HAMMING CODE:

i) Number of Parity Bits :

The number of parity bits depends on the number of information bits. If the number of information bits is designed 'x', then the number of parity bits, P is determined by the following relationship:

$$2^P \geq x + P + 1$$

For example, if we have four information bit, i.e. $x = 4$, then P is found by trial and error using the above equation. Let $P = 2$. Then

$$2^P = 2^2 = 4$$

and

$$x + P + 1 = 4 + 2 + 1 = 7$$

Since 2^P must be equal to or greater than $x + P + 1$, the relationship in equation is not satisfied.

Hence we have to try with next value of P. Let $P = 3$.

$$2^P = 2^3 = 8$$

and

$$x + P + 1 = 4 + 3 + 1 = 8$$

This value of P satisfies the relationship given in the equation, and therefore we can say that three parity bits are required to provide single error correction for four information bits.

BASIC ELECTRONICS ENGINEERING

ii) Locations of the Parity Bits in the Code:

Now we know that how to calculate the number of parity bits required to provide single error correction for given number of information bits. In our example we have four information bits and three parity bits. Therefore, the code is of seven bits. The rightmost bit is designated bit 1, the next bit is bit 2, and so on, as shown below:

Bit 7, Bit 6, Bit 5, Bit 4, Bit 3, Bit 2, Bit 1

The parity bits are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8 ...). Therefore, for 7 - bit code, locations for parity bits and information bits are as shown below :

D₇, D₆, D₅, P₄, D₃, P₂, P₁

Where symbol P_n designates a particular parity bit, D_n designates a particular information bit and n is the location number.

iii) Assigning Values to Parity Bits:

Now we know the format of the code. Let us see how to determine 1 or 0 value to each parity bit. In Hamming code, each parity bit provides a check on certain other bits in the total code; therefore, we must know the value of these others in order to assign the parity bit value. To do this, we must write the binary number for each decimal location number as shown in the third row of the following table.

Bit designation	D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Information bits (D _n)							
Parity bits (P _n)							

Assignment of P₁ :

Looking at the Table, we can see that the binary location number of parity bit P₁ has a 1 for its rightmost digit. This parity bit checks all bit locations, including itself, that have 1's in the same location in the binary location numbers. Therefore, parity bit P₁ checks bit locations 1, 3, 5 and 7, and assigns P₁ according to even or odd parity. For even parity Hamming code, it assigns P₁ such that bit locations 1, 3, 5, and 7 will have even parity.

Assignment of P₂ :

Looking at the Table, we can see that the binary location number of parity bit P₂ has a 1 for its middle bit. This parity bit checks all bit locations, including itself, that have 1 in the middle bit. Therefore, parity bit P₂ checks bit locations 2, 3, 6 and 7 and assigns P₂ according to even or odd parity.

Assignment of P₄ :

Looking at the Table, we can see that the binary location number of parity bit P₄ has a 1 for its leftmost digit. This parity bit checks all bit locations, including itself, that have 1's in the leftmost bit. Therefore, parity bit P₄ checks bit locations 4, 5, 6 and 7 and assigns P₄ according to even and odd parity.

BASIC ELECTRONICS ENGINEERING

7-bit HAMMING CODE

Example:

1) Encode the binary word 1011 into seven bit even parity hamming code.

SOL:

Step 1 : Find the number of parity bits required. Let $P = 3$, then

$$2^P = 2^3 = 8$$

$$x + P + 1 = 4 + 3 + 1 = 8$$

Three parity bits are sufficient.

$$\therefore \text{Total code bits} = 4 + 3 = 7$$

Step 2 : Construct a bit location table.

Bit designation	D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Information bits	1	0	1		1		
Parity bits				0		0	1

Step 3 : Determine the parity bits.

For P_1 : Bit locations 3, 5, and 7 have three 1s and therefore to have an even parity P_1 must be 1.

For P_2 : Bit locations 3, 6, and 7 have two 1s and therefore to have an even parity P_2 must be 0.

For P_3 : Bit locations 5, 6, and 7 two 1s and therefore to have an even parity P_4 must be 0.

Step 4 : Enter the parity bits into the table to form a seven bit Hamming code = 1010101.

Detecting and Correcting an Error

In the last section we have seen how to construct Hamming code for given number of information bits. Now we will see how to use it to locate and correct an error. To do this, each parity bit, along with its corresponding group of bits must be checked for proper parity. The correct result of individual parity check is marked by 0 whereas wrong result is marked by 1. After all parity checks, binary word is formed taking resulting bit for P_1 as LSB. This word gives bit location where error has occurred. If word has all bits 0 then there is no error in the Hamming code.

Example:

2) Assume that the even parity Hamming code (0110011) is transmitted and that 0100011 is received. The receiver does not know what was transmitted. Determine bit location where error has occurred using received code.

BASIC ELECTRONICS ENGINEERING

SOL:

Step 1 : Construct the bit location table.

Bit designation	D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Received Code	0	1	0	0	0	1	1

Step 2 : Check for parity bits

For P₁ : P₁ checks locations 1, 3, 5, and 7.

There is one 1 in the group.

∴ Parity checks for even parity is wrong → 1 (LSB)

For P₂ : P₂ checks locations 2, 3, 6 and 7.

There are two 1s in the group

∴ Parity check for even parity is correct→ 0

For P₄ : P₄ checks locations 4, 5, 6 and 7.

There are one 1 in the group

∴ Parity check for even parity is wrong → 1

The resultant word is 1 0 1. This says that the bit in the number 5 location is in error. It is 0 and should be a 1. Therefore, the correct code is 0 1 1 0 0 1 1, which agrees with the transmitted code.

12-bit Hamming Code

To transmit eight data bits, four parity bits located at positions 2⁰, 2¹, 2² and 2³ from left are added to make a 12-bit code word which is then transmitted. The Word format would be as shown below:

$$D_{12} D_{11} D_{10} D_9 P_8 D_7 D_6 D_5 P_4 D_3 P_2 P_1$$

Where the D bits are the data bits and the P bits are the parity bits.

P₁ is to be set to 0 or 1 so that it establishes even parity over bits 1, 3, 5, 7, 9, and 11 (i.e. over bits P₁ D₃ D₅ D₇ D₉ D₁₁).

P₂ is to be set to 0 or 1 so that it establishes even parity over bits 2, 3, 6, 7, 10, and 11 (i.e. over bits P₂ D₃ D₆ D₇ D₁₀ D₁₁).

P₃ is to be set to 0 or 1 so that it establishes even parity over bits 4, 5, 6, 7, and 12 (i.e. over bits P₄ D₅ D₆ D₇ D₁₂).

P₄ is to be set to 0 or 1 so that it establishes even parity over bits 8, 9, 10, 11 and 12 (i.e. over bits P₈ D₉ D₁₀ D₁₁ D₁₂).

Example:

3) (a) Given the 8-bit data word 01011011, generate the 12-bit composite word for the Hamming code that corrects and detects single errors.

BASIC ELECTRONICS ENGINEERING

(b) A 12-bit hamming code word containing 8 bits of data and 4 parity bits is read from memory. What is the original 8-bit word if the 12-bit read out is as follows.

- (i) 1000 1110 1010
- (ii) 1011 1000 0110
- (iii) 1011 1111 0100

SOL:

(a) For a 8-bit data word, 4 parity bits located at positions 2^0 , 2^1 , 2^2 , and 2^3 from right are added to generate the composite word for the Hamming code that corrects and detects single errors. The format and the data bits are shown below.

D12	D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1
0	1	0	1	0	1	0	1	0	1	1	1

P_1 is to be selected so that bits 1, 3, 5, 7, 9, 11 (P_1 11111) have even parity. So $P_1 = 1$

P_2 is to be selected so that bits 2, 3, 6, 7, 10, 11 (P_2 10101) have even parity. So $P_2 = 1$

P_4 is to be selected so that bits 4, 5, 6, 7, 12 (P_4 1010) have even parity. So $P_4 = 0$.

P_8 is to be selected so that bits 8, 9, 10, 11, 12 (P_8 1010) have even parity. So $P_8 = 0$

Based on these, the 12-bit hamming code word is 010101010111

(b) Out of the 12 bits in the Hamming code, 4 bits located in positions 1, 2, 4, 8 from left are parity bits. The remaining 8 bits are data bits. So the original 8 bit word is made of bits 3, 5, 6, 7, 9, 10, 11, 12 from left. The data word is as shown below.

	P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}	Data word
(i)	0	0	0	0	1	1	1	0	1	0	1	0	0 1 1 1 1 0 1 0
(ii)	1	0	1	1	1	0	0	0	0	1	1	0	1 1 0 0 0 1 1 0
(iii)	1	0	1	1	1	1	1	0	0	1	0	0	1 1 1 1 0 1 0 0

15-bit Hamming Code

To transmit eleven data bits, four parity bits located at positions 2^0 , 2^1 , 2^2 and 2^3 from left are added to make a 15-bit code word which is then transmitted. The word format would be as shown below:

$$D_{15} D_{14} D_{13} D_{12} D_{11} D_{10} D_9 P_8 D_7 D_6 D_5 P_4 D_3 P_2 P_1$$

where the D bits are the data bits and the P bits are the parity bits.

P_1 is to be set to a 0 or a 1 so that it establishes even parity over bits 1, 3, 5, 7, 9, 11, 13 and 15 (i.e. over bits $P_1 D_3 D_5 D_7 D_9 D_{11} D_{13} D_{15}$).

P_2 is to be set to a 0 or a 1 so that it establishes even parity over bits 2, 3, 6, 7, 10, 11, 14 and 15 (i.e. over bits $P_2 D_3 D_6 D_7 D_{10} D_{11} D_{14} D_{15}$).

P_4 is to be set to a 0 or a 1 so that it establishes even parity over bits 4, 5, 6, 7, 12, 13, 14 and 15 (i.e. over bits $P_4 D_5 D_6 D_7 D_{12} D_{13} D_{14} D_{15}$).

P_8 is to be set to a 0 or a 1 so that it establishes even parity over bits 8, 9, 10, 11, 12, 13, 14 and 15 (i.e. over bits $P_8 D_9 D_{10} D_{11} D_{12} D_{13} D_{14} D_{15}$).

Example:

BASIC ELECTRONICS ENGINEERING

4)

- (a) Derive a single error correcting code for a 11-bit group 01101110101.
 (b) Test the following Hamming code sequence for 11-bit message and correct it if necessary (101001011101011).

SOL:

(a) For the given 11-bit group, the bit pattern is

D15	D14	D13	D12	D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1
0	1	1	0	1	1	1	1	0	1	0	1	1	1	0

We know that the parity bits $P_1, P_2, P_4,$ and P_8 are to be selected such that:

Bits 1,3,5,7,9,11,13,15 (i.e. P_1 1001110) must have even parity. So P_1 must be a 0.

Bits 2,3,6,7,10,11,14,15 (i.e. P_2 1101110) must have even parity. So P_2 must be a 1.

Bits 4,5,6,7,12,13,14,15 (i.e. P_4 0100110) must have even parity. So P_4 must be a 1.

Bits 8,9,10,11,12,13,14,15 (i.e. P_8 1110110) must have even parity. So P_8 must be a 1.

Putting the above parity bits the final code word is 011011110101110

(b) The given Hamming code sequence for 11-bit message is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
1	0	1	0	0	1	0	1	1	1	0	1	0	1	1

Bits 1, 3, 5, 7, 9, 11, 13, 15 (11001001) → no error → put a 0 in the 1's position. $c_1 = 0$

Bits 2, 3, 6, 7, 10, 11, 14, 15 (01101011) → error → put a 1 in the 2's position. $c_2 = 1$

Bits 4, 5, 6, 7, 12, 13, 14, 15 (00101011) → no error → put a 0 in the 4's position. $c_3 = 0$

Bits 8, 9, 10, 11, 12, 13, 14, 15 (11101011) → no error → put a 0 in the 8's position. $c_4 = 0$

The error word is $c_4 c_3 c_2 c_1 = 2_{10}$. So the 2nd bit (from left) is in error. Complement it.
 Therefore, the correct code is 111001011101011.

PRACTICE PROBLEMS

1) Determine the decimal numbers represented by the following binary numbers

- (a) 110101 (b) 11111111 (c) 1100.1011

SOL:

$$\begin{aligned}
 \text{(a) } (110101)_2 &= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &= 32 + 16 + 0 + 4 + 0 + 1 \\
 &= (53)_{10}
 \end{aligned}$$

$$\begin{aligned}
 \text{(b) } (11111111)_2 &= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 \\
 &= (255)_{10}
 \end{aligned}$$

$$\begin{aligned}
 \text{(c) } (1100.1011)_2 &= 8 + 4 + 0 + 0 + 0.5 + 0 + 0.125 + 0.0625 \\
 &= (12.6875)_{10}
 \end{aligned}$$

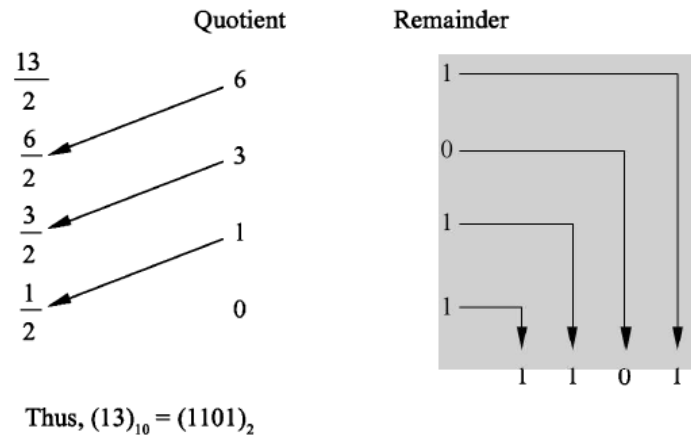
2) Convert the following decimal numbers into equivalent binary numbers

- (a) $(13)_{10}$ (b) $(0.65625)_{10}$ (c) $(25.5)_{10}$

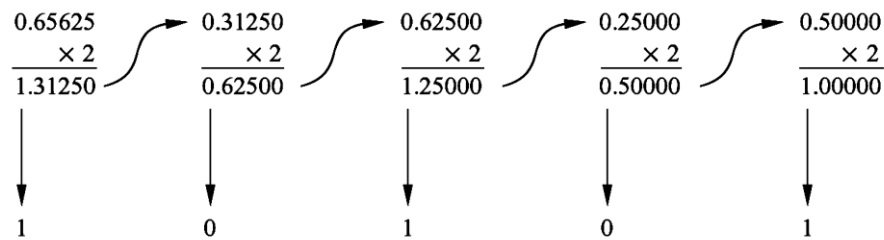
SOL:

BASIC ELECTRONICS ENGINEERING

(a)



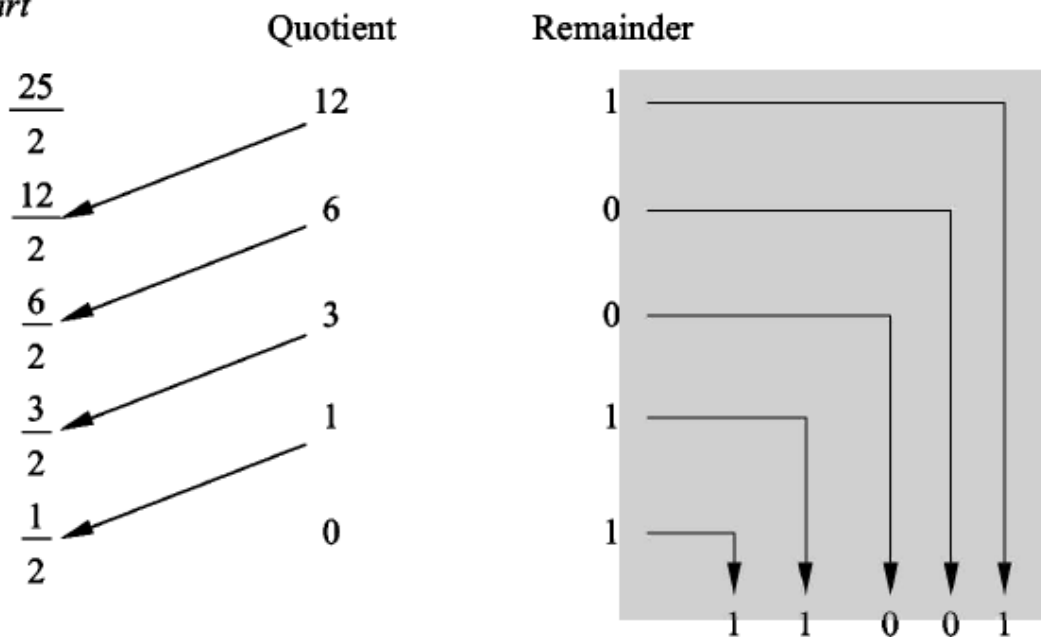
(b)



Thus, $(0.65625)_{10} = (0.10101)_2$

(c)

Integer part



Therefore, $(25)_{10} = (11001)_2$

BASIC ELECTRONICS ENGINEERING

Fractional part

$$\begin{array}{r} 0.5 \\ \times 2 \\ \hline 1.0 \\ \downarrow \\ 1 \end{array}$$

i.e., $(0.5)_{10} = (0.1)_2$

Therefore, $(25.5)_{10} = (11001.1)_2$

3) Convert octal $(623.77)_8$ to decimal, binary and hexadecimal

SOL:

$$(623.77)_8 = (?)_2$$

$$623 = 110010011$$

$$.77 = 111111$$

$$(623.77)_8 = (110010011.111111)_2$$

$$(623.77)_8 = (?)_{16}$$

$$\begin{array}{ccc} \overline{11} & \overline{1001} & \overline{0011} \\ \hline \overline{0001} & \overline{1001} & \overline{0011} \\ \hline 1 & 9 & 3 \end{array} = 193$$

$$\begin{array}{ccc} \overline{1111} & \overline{11} & \overline{-} \\ \hline \overline{1111} & \overline{1100} & \\ \hline 15 = F & 12 = C & \end{array} = FC$$

$$(623.77)_8 = (193.FC)_{16}$$

$$(623.77)_8 = (?)_{10}$$

$$623 = 6 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$$

$$= 384 + 16 + 3$$

$$= 403$$

$$.77 = 7 \times 8^{-1} + 7 \times 8^{-2}$$

$$= 7 \times .125 + 7 \times .015625$$

$$= .875 + .109375$$

$$= 0.9843$$

$$(623.77)_8 = (403.9843)_{10}$$

BASIC ELECTRONICS ENGINEERING

4) Find the radix/base of $\sqrt{41} = 5$

SOL: Let

$$(\sqrt{41})_b = 5_b$$

$$\sqrt{4 \times b^1 + 1 \times b^0} = 5 \times b^0$$

$$\sqrt{4b + 1} = 5$$

$$4b + 1 = 25$$

$$4b = 24$$

$$b = 6$$

5) Determine the Hamming code for the information code 10111 with odd parity

SOL:

Step 1 : Find the number of parity bits required. Let $P = 3$.

$$\text{Then } 2^P = 2^3 = 8$$

$$\therefore x + P + 1 = 5 + 3 + 1 = 9$$

This will not work. Try $P = 4$. Then

$$2^P = 2^4 = 16$$

$$x + P + 1 = 5 + 4 + 1 = 10$$

Equation 1 is satisfied and hence four bits are sufficient.

$$\therefore \text{Total code bit} = 5 + 4 = 9$$

Step 2 : Construct a bit location table

Bit designation	D ₉	P ₈	D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Information bits	1		0	1	1		1		
Parity bits		0				1		1	0

Step 3 : Determine the Parity Bits

For P_1 : Bit locations 3, 5, 7 and 9 have three 1s and therefore to have odd parity P_1 must be 0.

For P_2 : Bit locations 3, 6, 7 have two 1s and therefore to have an odd parity P_2 must be 1.

For P_4 : Bit locations 5, 6 and 7 have two 1s and therefore to have an odd parity P_4 must be 1.

For P_8 : Bit P_8 checks bit locations 8 and 9 and must be 0 to have an odd parity.

BASIC ELECTRONICS ENGINEERING

Step 4 : Enter the parity bits into the table to form a nine bit Hamming code=100111110.

6) The Hamming code 101101101 is received. Correct it if any errors. There are four parity bits and odd parity is used.

SOL:

Step 1 : Construct a bit location table

Bit designation	D ₉	P ₈	D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Received code	1	0	1	1	0	1	1	0	1

Step 2 : Check for parity bits

For P₁ : P₁ checks locations 1, 3, 5, 7 and 9.

There are four 1s in the group

∴ Parity check for odd parity is wrong → 1 (LSB)

For P₂ : P₂ checks locations 2, 3, 6 and 7.

There are three 1s in the group

∴ Parity check for odd parity is correct → 0

For P₄ : P₄ checks locations 4, 5, 6 and 7.

There are three 1s in the group

∴ Parity check for odd parity is correct → 0

For P₈ : P₈ checks locations 8 and 9.

There is one 1 in the group

∴ Parity check for odd parity is correct → 0

The resultant word is 0 0 0 1. This says that the bit in the number 1 location is in error. It is 1 and should be a 0. Therefore, the correct code is 1 0 1 1 0 1 1 0 0.

7. Convert (1543)₉ into base 10 & base 6

SOL: (1543)₉ = $1 \times 9^3 + 5 \times 9^2 + 4 \times 9 + 3 \times 9^0 = (1173)_{10}$

$$\begin{array}{r}
 6 \overline{) 1173} \\
 \underline{6 195} - 3 \\
 6 \overline{) 32} - 3 \\
 \underline{ 5} - 2
 \end{array}
 = (5233)_6$$

BASIC ELECTRONICS ENGINEERING

8. Determine the possible base value for $(302)_{(b)} = 12.1$

SOL:

$$\text{Consider, } 302 = 3b^2 + 0b + 2$$

$$20 = 2b + 0$$

$$12.1 = 1b + 2 + 1/b$$

Now,

$$(3b^2 + 2) / 2b = b + 2 + 1/b$$

$$3b^2 + 2 = 2b^2 + 4b + 2 \text{ [multiplying by } 2b \text{]}$$

$$b^2 = 4b$$

$$\text{So, } b = 4$$

Therefore, base is 4.

9. Convert i) $(2311)_{16} = (?)_{10} = (?)_2$

ii) $(A44D)_{16} = (?)_{10} = (?)_2$

SOL:

$$\text{i) } (2311)_{16} = 2 \times 16^3 + 3 \times 16^2 + 1 \times 16^1 + 1 \times 16^0$$

$$= 2 \times 4096 + 3 \times 256 + 16 + 1$$

$$= (8977)_{10}$$

2	3	1	1
0010	0011	0001	0001

Hex number

Binary number

$$(2311)_{16} = (8977)_{10} = (10001100010001)_2$$

$$\text{ii) } (A44D)_{16} = 10 \times 16^3 + 4 \times 16^2 + 4 \times 16^1 + 13 \times 16^0$$

$$= 10 \times 4096 + 4 \times 256 + 4 \times 16 + 13$$

$$= (42061)_{10}$$

A	4	4	D
1010	0100	0100	1101

Hex number

Binary number

$$(A44D)_{16} = (42061)_{10} = (1010010001001101)_2$$

10) Determine the base of the numbers for the following options to be correct.

i) $\frac{14}{2} = 2$ ii) $24 + 17 = 40$

SOL:

i) $14/2 = 2;$

Find decimal equivalent

$$14 = 1 \times r^1 + 4 \times r^0 = r + 4$$

$$2 = 2 \times r^0 = 2$$

$$2 = 2 \times r^0 = 2$$

BASIC ELECTRONICS ENGINEERING

$$(4+r)/2=2$$

Solving this equation, we get $r=0$, Hence No base value.

ii) $24+17=40$;

Find decimal equivalent

$$24=2 \times r^1 + 4 \times r^0 = 2r + 4$$

$$17=1 \times r^1 + 7 \times r^0 = r + 7$$

$$40=4 \times r^1 + 0 \times r^0 = 4r + 0$$

$$(2r + 4) + (r + 7) = 4r$$

Solving this equation, we get $r=11$, base 11

11) Given that $(16)_{10} = (100)_b$, Find the value of 'b'.

SOL:

$$\text{Given } (16)_{10} = (100)_b$$

$$1 \times b^2 + 0 \times b^1 + 0 \times b^0 = 16$$

$$b^2 = 16$$

$$\text{Base, } b = 4$$

BOOLEAN POSTULATES

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof & also we can build a set of useful theorems, rules and properties of system. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

I) Closure

(a) : Closure with respect to the operator '+'

When two binary elements are operated by operator '+' the result is a unique binary element (1 or 0).

(b) : Closure with respect to the operator • (dot).

When two binary elements are operated by operator • (dot), the result is a unique binary element.

II) Identity element:

An identity element with respect to +, designated by '0'.

$$A+0=0+A=A$$

An identity element with respect to • (dot), designated by '1'

$$A \cdot 1 = 1 \cdot A = A$$

III) Commutative Law:

Commutative with respect to '+'

$$A + B = B + A$$

Commutative with respect to '•'

$$A \cdot B = B \cdot A$$

IV) Distributive Law:

a) Distributive property of '•' over '+'

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

b) Distributive property of '+' over '•'

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

V) Complement:

For every binary element, there exists complement element.

BASIC ELECTRONICS ENGINEERING

For example, if A is an element, we have \bar{A} is a complement of A. i.e. if $A = 0$, $\bar{A} = 1$ and if $A = 1$, $\bar{A} = 0$.

$$A + \bar{A} = 1$$

$$A \cdot \bar{A} = 0$$

VI) There exists at least two elements, say A and B in the set of binary elements such that $A \neq B$.

BASIC THEOREMS/BASIC LAWS IN BOOLEAN ALGEBRA

1) PRINCIPLE OF DUALITY:

This principle states that one expression can be obtained from the other in each pair by

- interchanging every element i.e., every 0 with 1, every 1 with 0
- interchanging the operators i.e., every (+) with (•) and every (•) with (+).

This important property of Boolean algebra is called principle of duality.

Example:

<i>Given Expression</i>	<i>Dual</i>
1. $\bar{0} = 1$	$\bar{1} = 0$
2. $0 \cdot 1 = 0$	$1 + 0 = 1$
3. $0 \cdot 0 = 0$	$1 + 1 = 1$
4. $A \cdot 1 = A$	$A + 0 = A$
5. $A \cdot B = B \cdot A$	$A + B = B + A$

THEOREMS:

2) Null Law:

a) $A + 1 = 1$

b) $A \cdot 0 = 0$

Proof: a)

$$\begin{aligned} A + 1 &= 1 \\ A + 1 &= 1 \cdot (A + 1) \\ &= (A + A') (A + 1) \quad (\text{V postulate}) \\ &= A + A' \cdot 1 \quad (\text{IV Postulate}) \\ &= A + A' \\ &= 1 \end{aligned}$$

b) $A \cdot 0 = 0$

$$\begin{aligned} A \cdot 0 &= A \cdot 0 + 0 \\ &= A \cdot 0 + A \cdot A' \end{aligned}$$

BASIC ELECTRONICS ENGINEERING

$$\begin{aligned}
 &= A \cdot (0 + A') \\
 &= AA' \\
 &= 0 \quad (\because \text{Duality})
 \end{aligned}$$

$$\begin{aligned}
 A + AB &= A \cdot 1 + AB \\
 &= A (1 + B) \\
 &= A \cdot 1 \\
 &= A
 \end{aligned}$$

3) Involution

Theorem:

$$\overline{\overline{A}} = A$$

Proof:

Let

$$A = 0$$

$$\overline{\overline{A}} = 1$$

$$= 0$$

$$= A$$

$$\begin{aligned}
 A (A + B) &= A \cdot A + AB \\
 &= A + AB \\
 &= A
 \end{aligned}$$

4) Idempotence Laws:

a) $A + A = A$

Proof:

(b) $A \cdot A = A$

Proof:

$$\begin{aligned}
 A.A &= A.A + 0 \\
 &= A.A + A.A' \\
 &= A(A + A') \\
 &= A.1 \\
 &= A
 \end{aligned}$$

5) Absorption Laws:

a) $A + AB = A$

Proof:

$$\begin{aligned}
 A + A &= (A + A).1 \\
 &= (A + A) (A + A') \\
 &= A + AA' \\
 &= A + 0 \\
 &= A
 \end{aligned}$$

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

Proof:

6) Redundant Literal Rule (RLR): {Simplification}

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

Proof:

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

Proof:

BASIC ELECTRONICS ENGINEERING

$$\begin{aligned}
 A + \bar{A}B &= A + AB + \bar{A}B \\
 &= A + B \cdot (A + \bar{A}) \\
 &= A + B \cdot 1 \\
 &= A + B
 \end{aligned}$$

7) Association Law:

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

Proof:

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

$$\begin{aligned}
 A \cdot (\bar{A} + B) &= (A + AB) \cdot (\bar{A} + B) \\
 &= A\bar{A} + AB + ABB \\
 &= AB + ABB \\
 &= AB + AB \\
 &= AB
 \end{aligned}$$

b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

Proof:

A	B	C	BC	A(BC)	AB	(AB)C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

8) Consensus Theorem: (Included Factor Theorem)

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

Proof:

$$\begin{aligned}
 \text{LHS} &= AB + \bar{A}C + BC \\
 &= AB + \bar{A}C + BC(A + \bar{A}) \\
 &= AB + \bar{A}C + BCA + BC\bar{A} \\
 &= AB(1 + C) + \bar{A}C(1 + B) \\
 &= AB(1) + \bar{A}C(1) \\
 &= AB + \bar{A}C \\
 &= \text{RHS}
 \end{aligned}$$

BASIC ELECTRONICS ENGINEERING

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

Proof:

$$\begin{aligned} \text{LHS} &= (A + B)(\bar{A} + C)(B + C) = (A\bar{A} + AC + B\bar{A} + BC)(B + C) \\ &= (AC + BC + \bar{A}B)(B + C) \\ &= ABC + BC + \bar{A}B + AC + BC + \bar{A}BC = AC + BC + \bar{A}B \\ \text{RHS} &= (A + B)(\bar{A} + C) \\ &= A\bar{A} + AC + BC + \bar{A}B \\ &= AC + BC + \bar{A}B = \text{LHS} \end{aligned}$$

Note:

If a sum of products comprises a term containing A and a term containing \bar{A} , and a third term containing the left-out literals of the first two terms, then the third term is redundant, that is, the function remains the same with and without the third term removed or retained.

9) De Morgan's Theorem:

Law-1:

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

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

Proof:

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

Law-2:

This law states that the complement of the product of the variables is equal to the sum of their individual complements. i.e.

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

Proof:

A	B	\overline{AB}	\bar{A}	\bar{B}	$\bar{A} + \bar{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Note: These laws can be extended to any number of variables, i.e.

BASIC ELECTRONICS ENGINEERING

$$\overline{A + B + C + D + \dots} = \bar{A} \bar{B} \bar{C} \bar{D} \dots$$
$$\overline{ABCD \dots} = \bar{A} + \bar{B} + \bar{C} + \bar{D} + \dots$$

10) Transposition Theorem:

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

Proof:

$$\begin{aligned}\text{RHS} &= (A + C)(\bar{A} + B) \\ &= A\bar{A} + C\bar{A} + AB + CB \\ &= 0 + \bar{A}C + AB + BC \\ &= \bar{A}C + AB + BC(A + \bar{A}) \\ &= AB + ABC + \bar{A}C + \bar{A}BC \\ &= AB + \bar{A}C \\ &= \text{LHS}\end{aligned}$$

EXAMPLES:

1) Simplify the expression

$$XY + XYZ + XY\bar{Z} + \bar{X}YZ$$

SOL:

$$\begin{aligned}&= XY(1 + Z) + XY\bar{Z} + \bar{X}YZ \\ &= XY + XY\bar{Z} + \bar{X}YZ \\ &= XY(1 + \bar{Z}) + \bar{X}YZ \\ &= XY + \bar{X}YZ \\ &= Y(X + \bar{X}Z) \\ &= Y(X + Z)\end{aligned}$$

2) Simplify the expression

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

SOL:

Step 1 : Apply the DeMorgan's theorem and multiply out all terms to get expression in sum of products form

BASIC ELECTRONICS ENGINEERING

$$\begin{aligned}Z &= A B + A \bar{B} \cdot (\overline{\bar{A} \bar{C}}) \\&= A B + A \bar{B} \cdot (\overline{\bar{A}} + \bar{C}) \\&= A B + A \bar{B} \cdot (A + C) \\&= A B + A \bar{B} A + A \bar{B} C\end{aligned}$$

Step 2 : Search for common terms for factorization and apply Boolean rules

$$\begin{aligned}Z &= A B + A \bar{B} A + A \bar{B} C \\&= A B + A \bar{B} + A \bar{B} C \\&= A B + A \bar{B} (1 + C) \\&= A B + A \bar{B} \\&= A (B + \bar{B}) \\&= A\end{aligned}$$

3) Reduce the expression

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

SOL:

The given expression is

Demorganize $\overline{AB + AC}$

Demorganize \overline{AB} and \overline{AC}

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

Simplify

Simplify

Simplify

Simplify

Simplify

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

$$= A[B + \bar{C}(\overline{AB} \overline{AC})]$$

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

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

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

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

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

$$= AB + 0 + 0$$

$$= AB$$

4) Reduce the expression

$$f = (B + BC) (B + \bar{B}C) (B + D).$$

SOL:

BASIC ELECTRONICS ENGINEERING

The given expression is

Multiply the first two terms

Reduce

Factor

Reduce

Expand

Simplify

$$f = (B + BC)(B + \bar{B}C)(B + D)$$

$$= (BB + BCB + B\bar{B}C + BC\bar{B}C)(B + D)$$

$$= (B + BC + 0 + 0)(B + D)$$

$$= B(1 + C)(B + D)$$

$$= B(B + D)$$

$$= BB + BD$$

$$= B(1 + D) = B$$

5) Simplify the expression

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

SOL:

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

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

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

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

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

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

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

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

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

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

$$= 1$$

BASIC ELECTRONICS ENGINEERING

LOGIC GATES

Logic gates are the fundamental building blocks of digital systems. A gate is defined as a logic circuit with one or more inputs and only one output which implement a desired logic function.

The interconnection of gates to perform a variety of logical operations is called logic design. The logic gates are usually embedded in LSI and VLSI circuits.

The Inputs and outputs of logic gates can occur only in two levels. These two levels are termed HIGH and LOW or TRUE and FALSE or ON and OFF or simply 1 and 0. The terminology used for logic levels is shown in the fig.

<i>Logic 0</i>	<i>Logic 1</i>
False	True
Off	On
Low	High
No	Yes
Open switch	Close switch

A table which lists all the possible combinations of input variables and the corresponding outputs is called a **truth table**. It shows how the logic circuit's output responds to various combinations of logic levels at the inputs.

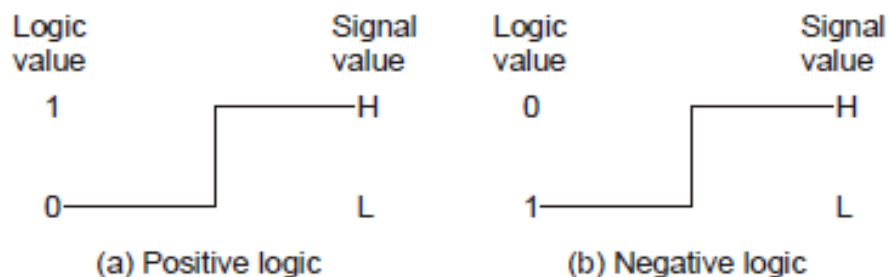
LEVEL LOGIC

The level logic may be positive logic or negative logic.

Positive and Negative Logic Designation:

The binary signals at the inputs or outputs of any gate can have one of the two values except during transition. One signal levels represents logic 1 and the other logic 0. Since two signal values are assigned two logic values, there exist two different assignments of signals to logic.

Logics 1 and 0 are generally represented by different voltage levels. Consider the two values of a binary signal as shown in the following Fig. One value must be higher than the other since the two values must be different in order to distinguish between them. We designate the higher voltage level by H and lower voltage level by L. There are two choices for logic values assignment. Choosing the high-level (H) to represent logic 1 as shown in (a) defines a **positive logic** system. Choosing the low level L to represent logic-1 as shown in (b), defines a **negative logic** system.



BASIC ELECTRONICS ENGINEERING

TYPES OF LOGIC GATES

The logic gates are of three types.

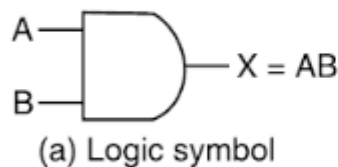
- 1) Basic gates : AND, OR, NOT
- 2) Universal gates : NAND, NOR
- 3) Desired gates : EX-OR, EX-NOR

i) AND GATE:

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

The logic symbol and truth table of AND gate are shown in the following fig.

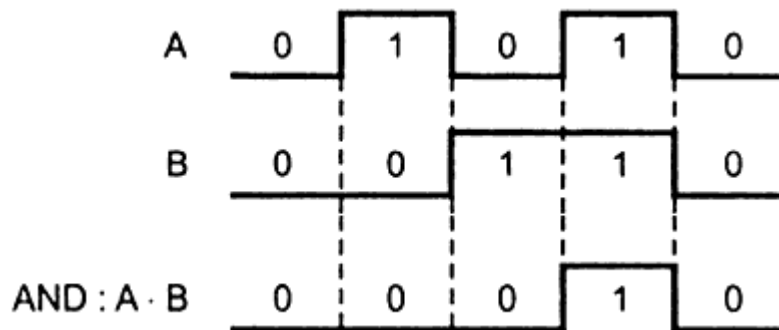
- IC 7408 contains 4 two input AND gates
- IC 7411 contains 3 three input AND gates
- IC 7421 contains 2 four input AND gates



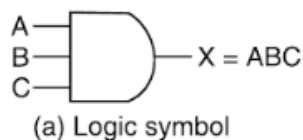
Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

The timing diagram of AND gate is shown in the following fig.



The logic symbol and the truth table of a three-input AND gate are shown in Figure



Inputs			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b) Truth table

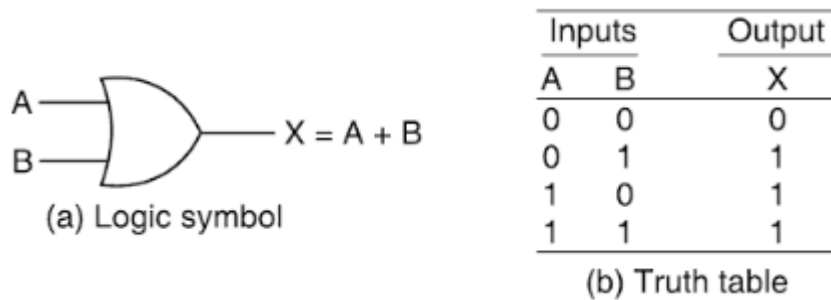
BASIC ELECTRONICS ENGINEERING

ii) OR GATE:

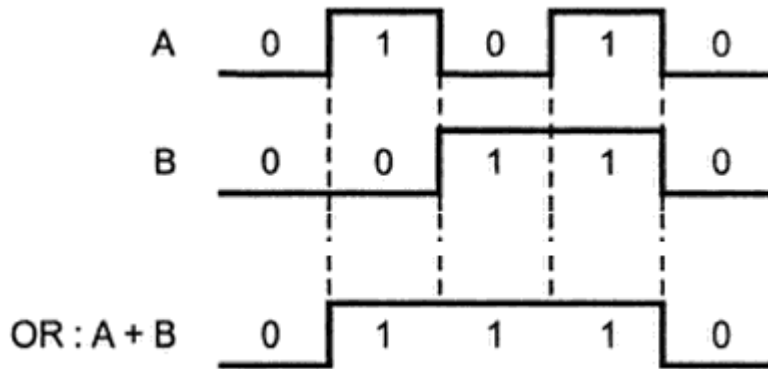
It is represented by '+' (plus). It has two or more inputs but only one output. The output assumes the logic 1 state only when one of its inputs is at logic 1 state. The output assumes the logic 0 state even if each input is at logic 0 state. The OR gate is also called an any or All gate. Also called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

OR gate operation is nothing but the addition of all inputs. The logic symbol and truth table of OR gate are shown in the following fig.

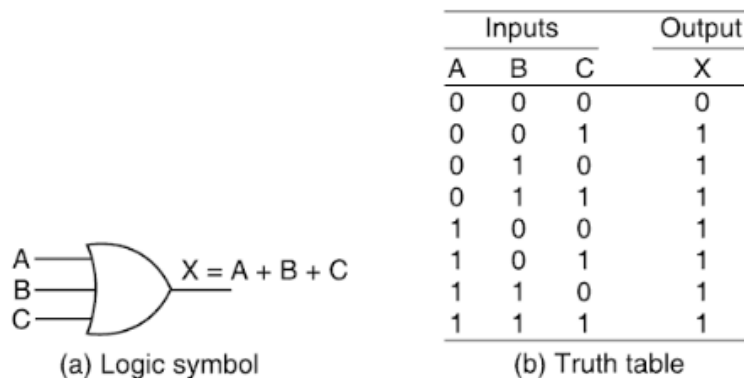
- IC 7432 Contains 4 two input OR gates.



The timing diagram of OR gate is shown in the following fig



The logic symbol and the truth table of a three-input OR gate are shown in Figure



NOTE:

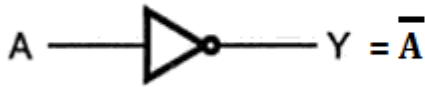
From the truth tables of AND & OR gates, the positive logic can be represented by AND gate and negative logic can be represented by OR gate.

BASIC ELECTRONICS ENGINEERING

iii) NOT GATE:

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

The logic symbol and truth table of NOT gate are shown in the fig.



(a) Logic symbol

A	Y
0	1
1	0

(b) Truth table

- IC7404 contains 6 inverter gates.
- Logic circuits which use AND/OR/INVERT gate are called AOI logic.
- Logic circuits which use AND/OR gate are called AO logic.

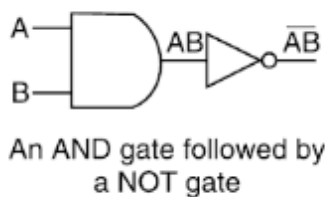
UNIVERSAL GATES

The NAND and NOR gates are widely used and are readily available in most integrated circuits. A major reason for the widespread use of these gates is that they are both UNIVERSAL gates, universal in the sense that both can be used for AND operators, OR operators, as well as Inverter. Thus, we see that a complex digital system can be completely synthesized using only NAND gates or NOR gates.

(iv) NAND GATE:

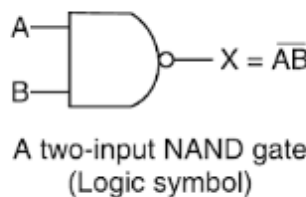
NAND gate means NOT AND i.e, AND output is NOTed. So NAND gate is a combination of AND & NOT gates. The little invert bubble (small circle) on the right end of the symbol means to invert the output of AND.

The logic symbol and truth table of NAND gate are shown in the fig.



An AND gate followed by a NOT gate

=



A two-input NAND gate (Logic symbol)

(a)

Truth table		
Inputs		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

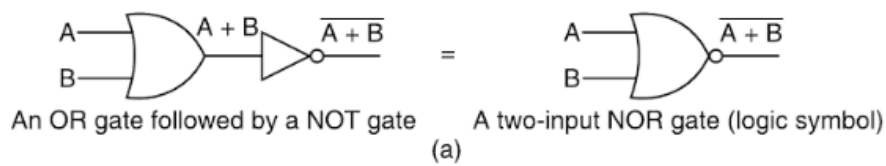
NAND assumes Logic 0 when each of inputs assumes logic 1. Otherwise the output is logic

1.

v) NOR GATE

NOR gate is NOT gate with OR gate. i.e, OR gate is NOTed or an inverted OR function. The standard logic symbol and truth table for the NOR gate are shown in Fig.

BASIC ELECTRONICS ENGINEERING



Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

The unique property of NOR gate is that it produces an output of logic '1' if and only if all its inputs are logic '0'.

NAND & NOR GATES AS UNIVERSAL GATES

(I) THE NAND GATE AS A UNIVERSAL GATE:

Logic Function	Symbol	Circuit using NAND gates only
Inverter		
AND		
OR		

(II) THE NOR GATE AS A UNIVERSAL GATE:

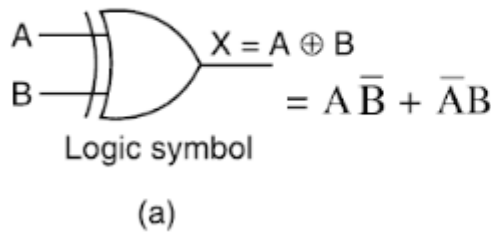
Logic Function	Symbol	Circuit using NOR gates only
Inverter		
AND		
OR		

BASIC ELECTRONICS ENGINEERING

vi) EX-OR GATE (EXCLUSIVE OR)

The exclusive OR gate is sometimes referred to as the “Odd but not the even gate”. It is often shortened as “XOR gate”. The logic diagram is shown in Fig. with its Boolean expression. The symbol \oplus means the terms are XORed together.

It has 2 inputs & only 1 output. It assumes output as 1 when inputs are not equal and it is called anti-coincidence gate or inequality detector. The logic symbol and truth tables are shown in the fig.



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

(b) Truth table

- TTL IC 746 has 4 EX-OR gate
- CMOS IC 74C8C has 4 EX-OR gates.

PROPERTIES OF EX-OR GATE:

1. $A \oplus 1 = \bar{A}$

Proof: $A \oplus 1 = A \cdot \bar{1} + \bar{A} \cdot 1 = A \cdot 0 + \bar{A} = \bar{A}$

2. $A \oplus 0 = A$

Proof: $A \oplus 0 = A \cdot \bar{0} + \bar{A} \cdot 0 = A \cdot 1 + 0 = A$

3. $A \oplus A = 0$

Proof: $A \oplus A = A \cdot \bar{A} + \bar{A} \cdot A = 0 + 0 = 0$

4. $A \oplus \bar{A} = 1$

Proof: $A \oplus \bar{A} = A \cdot \bar{\bar{A}} + \bar{A} \cdot \bar{A} = A + \bar{A} = 1$

5. $AB \oplus AC = A(B \oplus C)$

Proof: $AB \oplus AC = AB \cdot \bar{AC} + \bar{AB} \cdot AC = AB(\bar{A} + \bar{C}) + (\bar{A} + \bar{B})AC$
 $= AB\bar{C} + A\bar{B}C = A(B\bar{C} + \bar{B}C) = A(B \oplus C)$

6. If $A \oplus B = C$, then $A \oplus C = B$, $B \oplus C = A$, $A \oplus B \oplus C = 0$

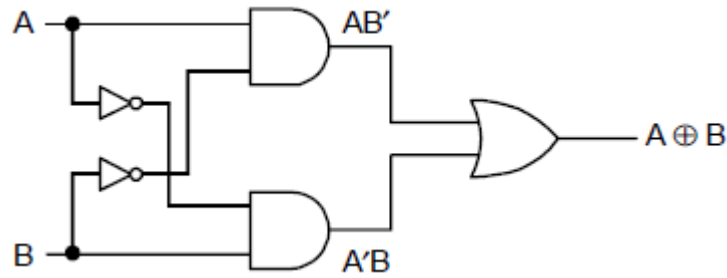
Proof: $A \oplus B \oplus C = (A \oplus B) \oplus (A \oplus B) = (A \oplus B) \cdot \overline{(A \oplus B)} + \overline{(A \oplus B)} \cdot (A \oplus B) = 0 + 0 = 0$

Proof: $A \oplus C = A \oplus (A \oplus B) = A \cdot \overline{(A \oplus B)} + \bar{A} \cdot (A \oplus B) = A(\bar{A}B + \bar{A}\bar{B}) + \bar{A}(A\bar{B} + \bar{A}B)$
 $= AB + \bar{A}\bar{B} = B(A + \bar{A}) = B \cdot 1 = B$

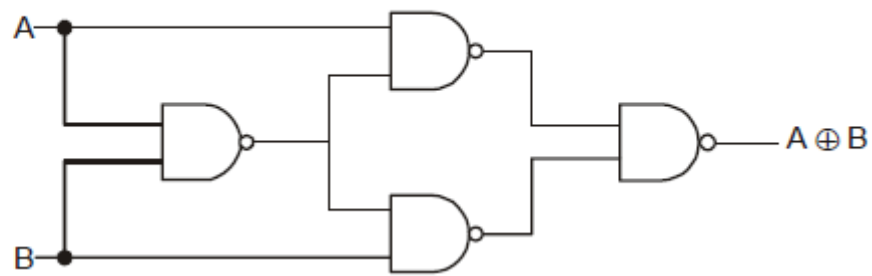
Proof: $B \oplus C = B \oplus (A \oplus B) = B \cdot \overline{(A \oplus B)} + \bar{B} \cdot (A \oplus B) = B \cdot (\bar{A}B + \bar{A}\bar{B}) + \bar{B}(A\bar{B} + \bar{A}B)$
 $= AB + A\bar{B} = A(B + \bar{B}) = A$

BASIC ELECTRONICS ENGINEERING

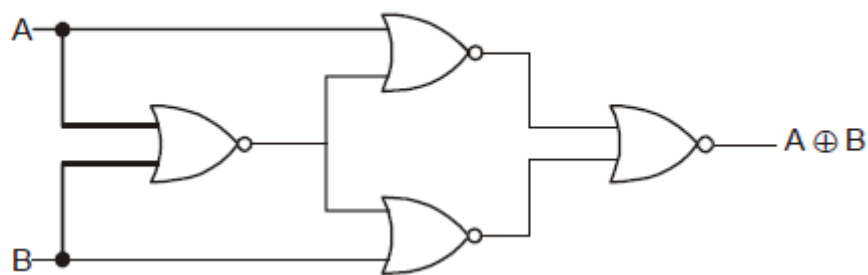
XOR GATE USING AND-OR-NOT GATES:



XOR gate using NAND gates only.

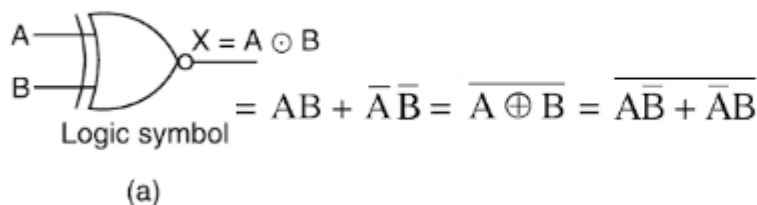


XOR using NOR gates only.



vii) EX-NOR GATE

The Exclusive NOR gate is sometimes referred to as the 'COINCIDENCE' or 'EQUIVALENCE' gate. This is often shortened as 'XNOR' gate. The logic diagram and truth table are shown in Fig. It is a combination of EX-OR and NOT gates. The invert of XOR function denoted by symbol \odot .



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

(b) Truth table

BASIC ELECTRONICS ENGINEERING

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

COMBINATIONAL LOGIC CIRCUITS

ADDERS

Digital computers perform various arithmetic operations and the most basic arithmetic operation is the addition of two binary digits .i.e, 4 basic possible operations are:

$$\begin{aligned}0 + 0 &= 0 \\0 + 1 &= 1 \\1 + 0 &= 1 \\1 + 1 &= 10_2\end{aligned}$$

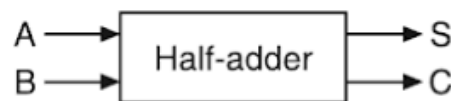
The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a **carry**. A combinational circuit that performs the addition of two bits is called a **half-adder**. One that performs the addition of 3 bits (two significant bits & previous carry) is called a **full adder** & two half adders can be employed to implement a full-adder.

HALF-ADDER

A Half Adder is a combinational circuit with two binary inputs (augends and addend bits) and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic circuit used to perform the arithmetic operation of addition of two single bit words. The truth table and block diagram of half adder are shown in the following figs.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

K-map simplification for sum & carry:

BASIC ELECTRONICS ENGINEERING

For carry		For sum	
A \ B		A \ B	
0	0	0	0
1	0	1	0

Carry = AB

Sum = $A\bar{B} + \bar{A}B$
 $= A \oplus B$

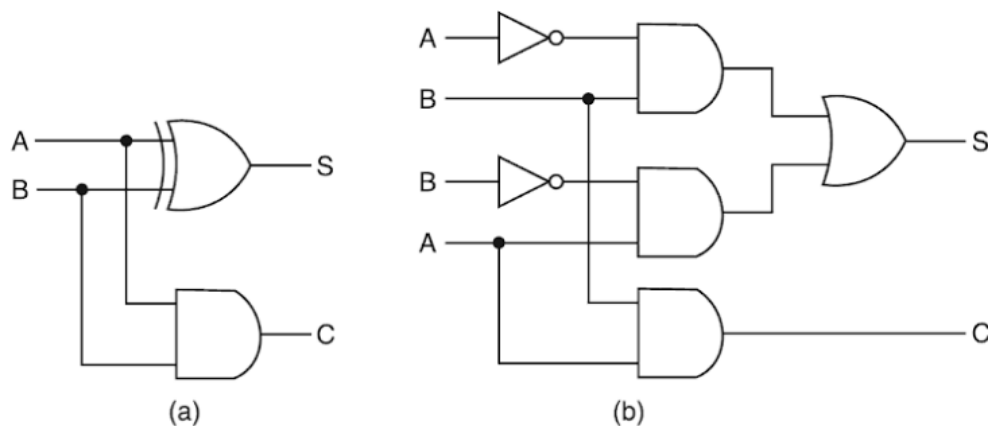
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B. Therefore,

$$S = A\bar{B} + \bar{A}B = A \oplus B$$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1). Therefore,

$$C = AB$$

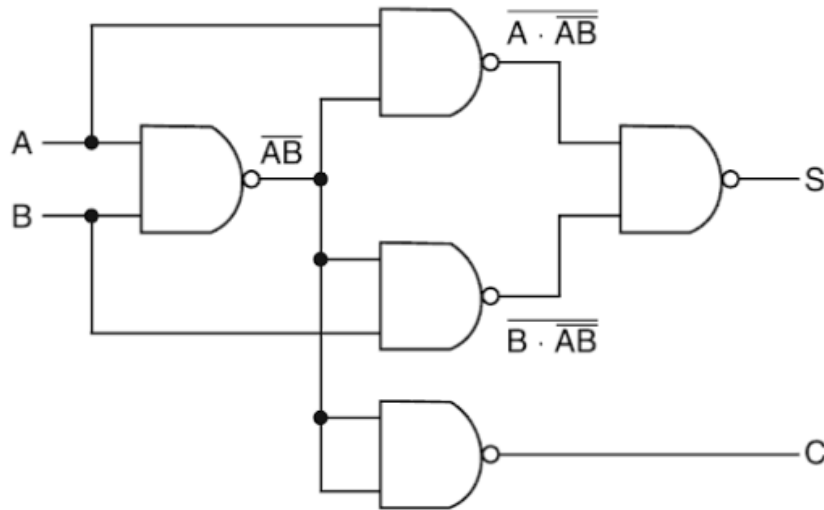
A half-adder can be realized by using one X-OR gate and one AND gate as shown in fig.



Realization of Half-adder using NAND logic:

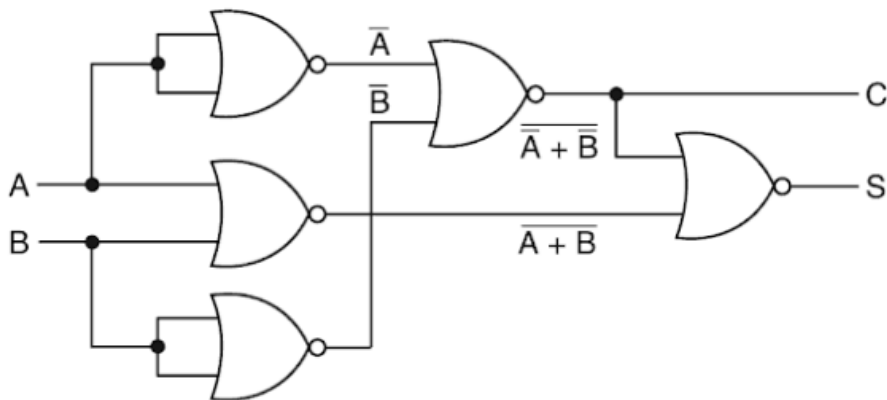
$$\begin{aligned}
 S &= A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B} \\
 &= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B}) \\
 &= A \cdot \overline{AB} + B \cdot \overline{AB} \\
 &= \overline{\overline{A \cdot AB} \cdot \overline{B \cdot AB}} \\
 C &= AB = \overline{\overline{AB}}
 \end{aligned}$$

BASIC ELECTRONICS ENGINEERING



Realization of Half-adder using NOR logic:

$$\begin{aligned}
 S &= A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B} \\
 &= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B}) \\
 &= (A + B)(\bar{A} + \bar{B}) \\
 &= \overline{\overline{A + B + \bar{A} + \bar{B}}} \\
 C &= AB = \overline{\overline{AB}} = \overline{\bar{A} + \bar{B}}
 \end{aligned}$$



FULL ADDER

A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry.

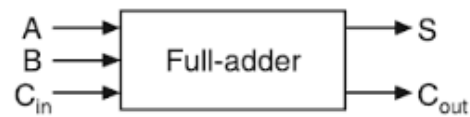
The block diagram and truth table of full adder are shown in the following fig. The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s, the output is 0. The S output is equal to 1 when only

BASIC ELECTRONICS ENGINEERING

1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	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

(a) Truth table



(b) Block diagram

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A, B and C_{in} is described by

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$= (\bar{A}\bar{B} + \bar{A}B)C_{in} + (A\bar{B} + AB)\bar{C}_{in} = (\bar{A} \oplus B)C_{in} + (A \oplus B)\bar{C}_{in} = A \oplus B \oplus C_{in}$$

$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in} = AB + (A \oplus B)C_{in} = AB + AC_{in} + BC_{in}$$

K-map simplification for carry and sum

For carry (C_{out})

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_{out} = AB + AC_{in} + BC_{in}$$

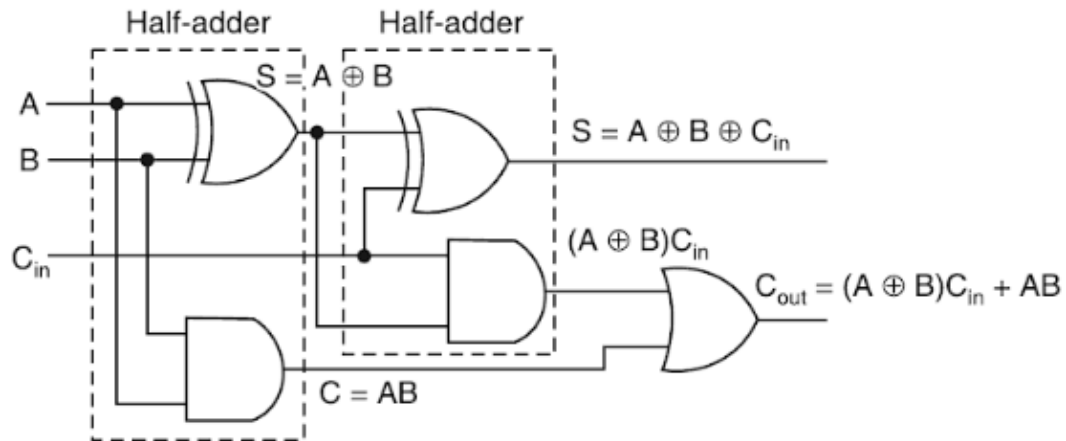
For sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

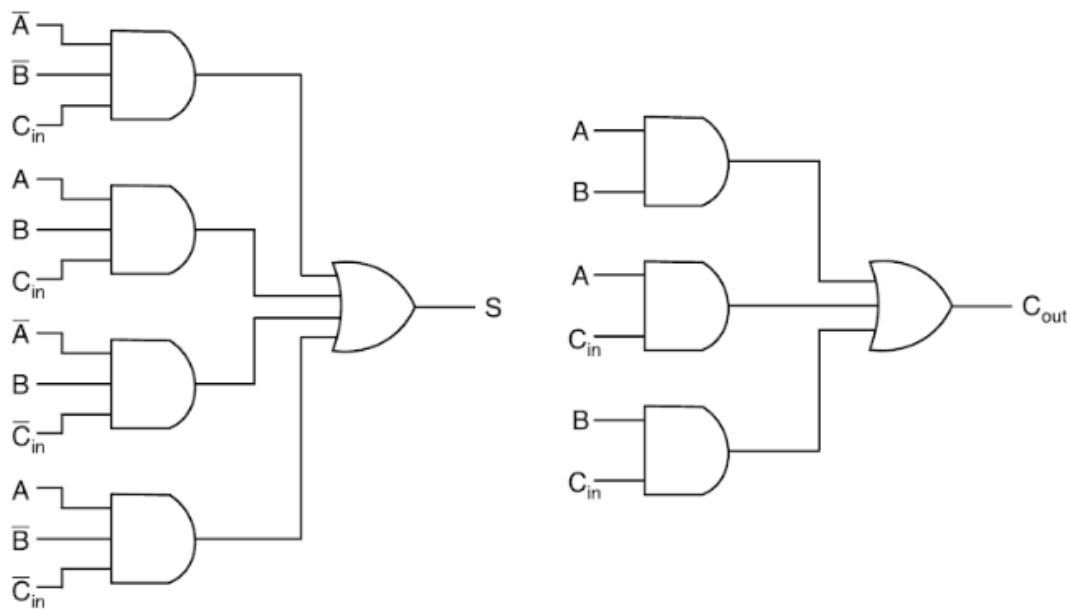
$$\text{Sum} = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

The sum term of the full-adder is the X-OR of A, B, and C_{in} , i.e., the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e., Two half adders) and one OR gate is as shown in the fig.

BASIC ELECTRONICS ENGINEERING



Realization of full-adder using AOI logic:



Realization of full-adder using NAND logic:

We know that

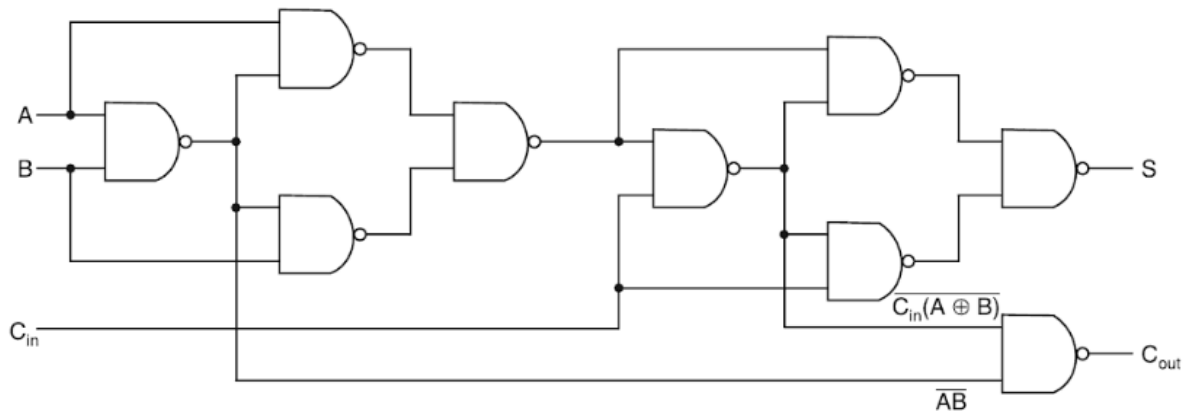
$$A \oplus B = \overline{\overline{A \cdot AB} \cdot \overline{B \cdot AB}}$$

Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot (A \oplus B)C_{in}} \cdot \overline{C_{in} \cdot (A \oplus B)C_{in}}}$$

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$

BASIC ELECTRONICS ENGINEERING



Realization of full-adder using NOR logic:

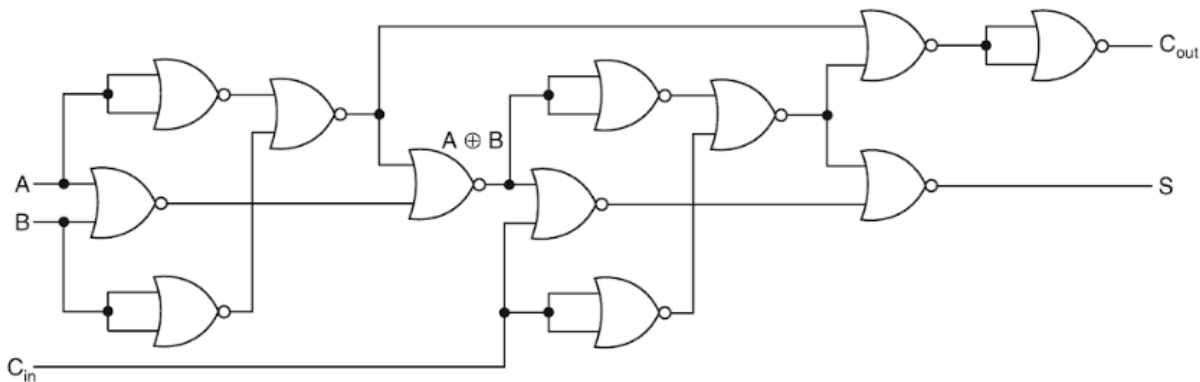
We know that

$$A \oplus B = \overline{\overline{A + B} + \overline{A} + \overline{B}}$$

Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{A \oplus B} + \overline{C_{in}} + \overline{A \oplus B} + \overline{C_{in}}}$$

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



NOTE:

$$\begin{aligned} \text{Sum} &= C_{in} \oplus (A \oplus B) \\ &= C_{in} \oplus (A \overline{B} + \overline{A} B) \\ &= C_{in} (\overline{A \overline{B} + \overline{A} B}) + \overline{C_{in}} (A \overline{B} + \overline{A} B) \\ &= C_{in} (\overline{A \overline{B}} \cdot \overline{\overline{A} B}) + \overline{C_{in}} (A \overline{B} + \overline{A} B) \\ &= C_{in} [(\overline{A} + B) \cdot (A + \overline{B})] + \overline{C_{in}} (A \overline{B} + \overline{A} B) \\ &= C_{in} (\overline{A} \overline{B} + A B) + \overline{C_{in}} (A \overline{B} + \overline{A} B) \\ &= \overline{A} \overline{B} C_{in} + A B C_{in} + A \overline{B} \overline{C_{in}} + \overline{A} B \overline{C_{in}} \\ &= \overline{A} \overline{B} C_{in} + \overline{A} B \overline{C_{in}} + A \overline{B} \overline{C_{in}} + A B C_{in} \end{aligned}$$

$$\begin{aligned} C_{out} &= AB + C_{in} (A \overline{B} + \overline{A} B) = AB + A \overline{B} C_{in} + \overline{A} B C_{in} \\ &= AB (C_{in} + 1) + A \overline{B} C_{in} + \overline{A} B C_{in} \quad \because C_{in} + 1 = 1 \\ &= AB C_{in} + AB + A \overline{B} C_{in} + \overline{A} B C_{in} \\ &= AB + A C_{in} (B + \overline{B}) + \overline{A} B C_{in} = AB + A C_{in} + \overline{A} B C_{in} \\ &= AB (C_{in} + 1) + A C_{in} + \overline{A} B C_{in} \quad \because C_{in} + 1 = 1 \\ &= A B C_{in} + AB + A C_{in} + \overline{A} B C_{in} \\ &= AB + A C_{in} + B C_{in} (A + \overline{A}) \\ &= AB + A C_{in} + B C_{in} \end{aligned}$$

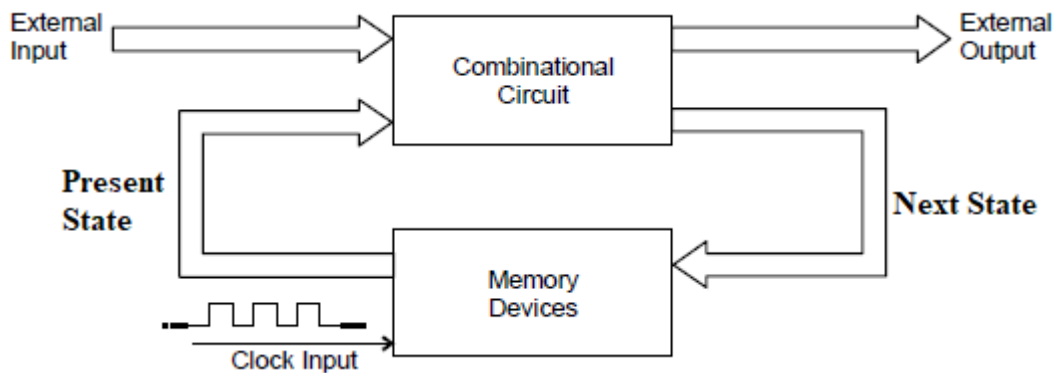
BASIC ELECTRONICS ENGINEERING

SEQUENTIAL LOGIC CIRCUITS

A sequential logic circuit is one whose outputs depend not only on its current inputs, but also on the past sequence of inputs.

A block diagram of a sequential circuit is shown in Fig. A Sequential circuit can be regarded as a collection of memory elements and combinational circuit, as shown in Fig. A feedback path is formed by using memory elements, input to which is the output of combinational circuit. The binary information stored in memory element at any given time is defined as the state of sequential circuit at that time. Present contents of memory elements are referred as the present state. The combinational circuit receive the signals from external input and from the memory output and determines the external output. They also determine the condition and binary values to change the state of memory. The new contents of the memory elements are referred as next state and depend upon the external input and present state.

Hence, a sequential circuit can be completely specified by a time sequence of inputs, outputs and the internal states. In general, clock is used to control the operation. The clock frequency determines the speed of operation of a sequential circuit.



COMPARISON BETWEEN COMBINATIONAL AND SEQUENTIAL CIRCUITS

COMBINATIONAL LOGIC CIRCUITS	SEQUENTIAL LOGIC CIRCUITS
Output is a function of the present inputs (Time Independent Logic).	Output is a function of clock, present inputs and the previous states of the system.
Do not have the ability to store data (state).	Have memory to store the present states that is sent as control input (enable) for the next operation.
It does not require any feedback. It simply outputs the input according to the logic designed.	It involves feedback from output to input that is stored in the memory for the next operation.
Used mainly for Arithmetic and Boolean operations.	Used for storing data (and hence used in RAM).
Logic gates are the elementary building blocks.	Flip flops (binary storage device) are the elementary building unit.
Independent of clock and hence does not require triggering to operate.	Clocked (Triggered for operation with electronic pulses).
Speed is fast	Speed is slow

BASIC ELECTRONICS ENGINEERING

This is time independent	This is time dependent
It is designed easy	It is designed tough as compared to combinational circuit
Example: Adder [$1+0=1$; Dependency only on present inputs i.e., 1 and 0].	Example: Counter [Previous O/P + 1 = Current O/P; Dependency on present input as well as previous state].

TYPES OF SEQUENTIAL CIRCUITS

Sequential circuits are broadly classified into two main categories, known as synchronous or clocked and asynchronous or unclocked sequential circuits, depending on the timing of their signals.

Synchronous Sequential circuits:

A sequential circuit whose behaviour can be defined from the knowledge of its signal at discrete instants of time is referred to as a synchronous sequential circuit. In these systems, the memory elements are affected only at discrete instants of time. The synchronization is achieved by a timing device known as a system clock, which generates a periodic train of clock pulses. The outputs are affected only with the application of a clock pulse. The rate at which the master clock generates pulses must be slow enough to permit the slowest circuit to respond. This limits the speed of all circuits. Synchronous circuits have gained considerable domination and wide popularity.

Asynchronous Sequential circuits:

A sequential circuit whose behaviour depends upon the sequence in which the input signals change is referred to as an asynchronous sequential circuit. The output will be affected whenever the input changes. The commonly used memory elements in these circuits are time-delay devices. There is no need to wait for a clock pulse. Therefore, in general, asynchronous circuits are faster than synchronous sequential circuits. However, in an asynchronous circuit, events are allowed to occur without any synchronization. And in such a case, the system becomes unstable. Since the designs of asynchronous circuits are more tedious and difficult, their uses are rather limited.

The clocked sequential circuits use a memory element known as Flip-Flop. A flip-flop is an electronic circuit used to store 1-bit of information, and thus forms a 1-bit memory cell. These circuits have two outputs, one giving the value of binary bit stored in it and the other gives the complemented value.

Differences between synchronous sequential circuit & Asynchronous sequential circuit

Sr. No.	Synchronous sequential circuits	Asynchronous sequential circuits
1.	In synchronous circuits, memory elements are clocked flip-flops.	In asynchronous circuits, memory elements are either unclocked flip-flops or time delay elements.
2.	In synchronous circuits, the change in input signals can affect memory element upon activation of clock signal.	In asynchronous circuits change in input signals can affect memory element at any instant of time.
3.	The maximum operating speed of clock depends on time delays involved.	Because of absence of clock, asynchronous circuits can operate faster than synchronous circuits.
4.	Easier to design.	More difficult to design.

BASIC ELECTRONICS ENGINEERING

MODES OF TRIGGERING

Flip-flops are synchronous sequential circuits. This type of circuit works with the application of a synchronization mechanism, which is termed as a clock. Based on the specific interval or point in the clock during or at which triggering of the flip-flop takes place, it can be classified into two different types—level triggering and edge triggering.

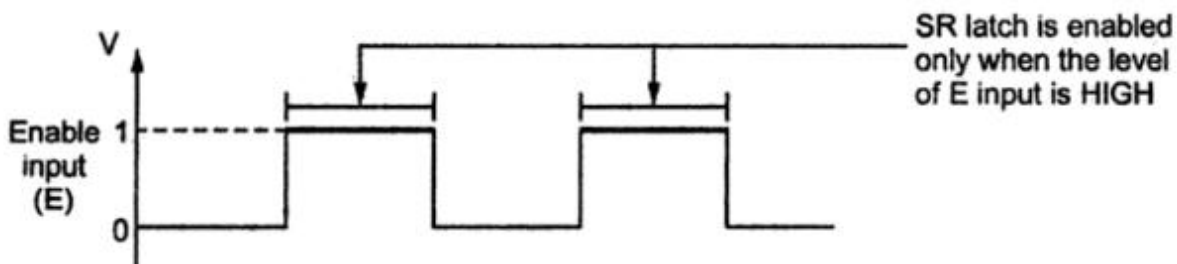
A clock pulse starts from an initial value of 0, goes momentarily to 1, and after a short interval, returns to the initial value.

a) Level Triggering of Flip-flops:

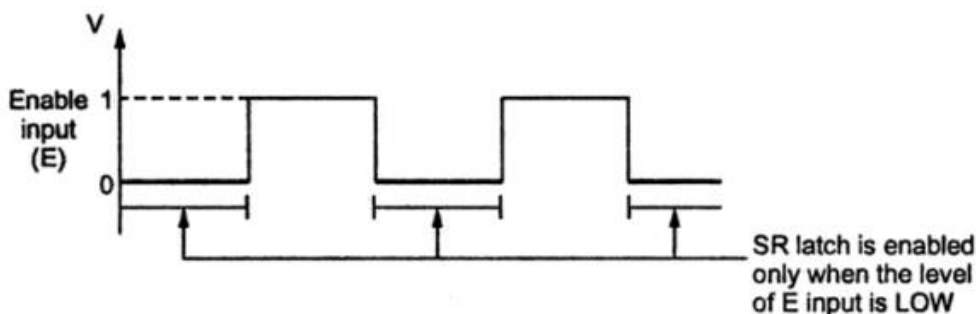
If a flip-flop gets enabled when a clock pulse goes HIGH and remains enabled throughout the duration of the clock pulse remaining HIGH, the flip-flop is said to be a level triggered flip-flop. If the flip-flop changes its state when the clock pulse is positive, it is termed as a positive level triggered flip-flop. On the other hand, if a NOT gate is introduced in the clock input terminal of the flip-flop, then the flip-flop changes its state when the clock pulse is negative, it is termed as a negative level triggered flip-flop.

The main drawback of level triggering is that, as long as the clock pulse is active, the flip-flop changes its state more than once or many times for the change in inputs. If the inputs do not change during one clock pulse, then the output remains stable. On the other hand, if the frequency of the input change is higher than the input clock frequency, the output of the flip-flop undergoes multiple changes as long as the clock remains active. This can be overcome by using either master-slave flip-flops or the edge-triggered flip-flop.

- **Positive level triggered :** The output of latch responds to the input changes only when its enable input is 1 (HIGH).



- **Negative level triggered :** The output of latch responds to the input changes only when its enable input is 0 (Low).



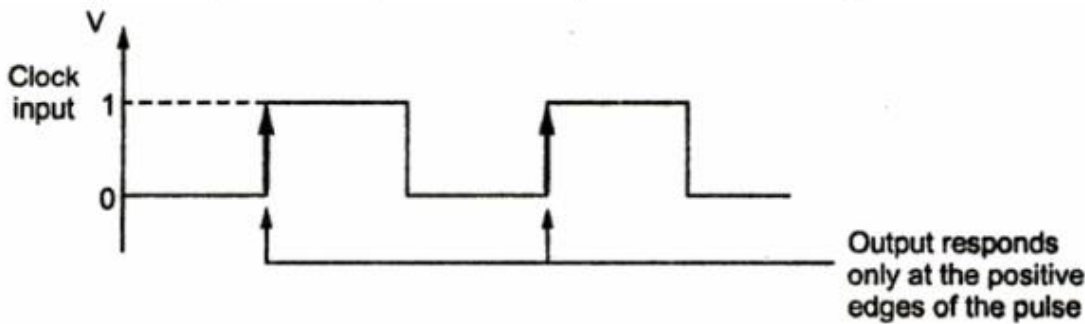
b) Edge Triggering of Flip-flops:

A clock pulse goes from 0 to 1 and then returns from 1 to 0. There are two types of edge triggering; they are the positive edge (0 to 1 transition) and the negative edge (1 to 0 transition).

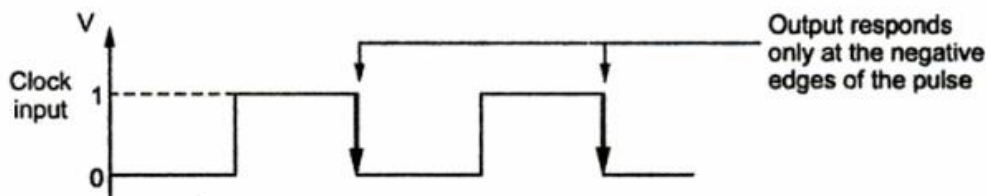
BASIC ELECTRONICS ENGINEERING

The term edge-triggered means that the flip-flop changes its state only at either the positive or negative edge of the clock pulse.

- **Positive edge triggering :** Here, the output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.



- **Negative edge triggering :** Here, the output responds to the changes in the input only at the negative edge of the clock pulse at the clock input.



CLOCKED S-R FLIPFLOP

a) Positive edge triggered SR flip-flop:

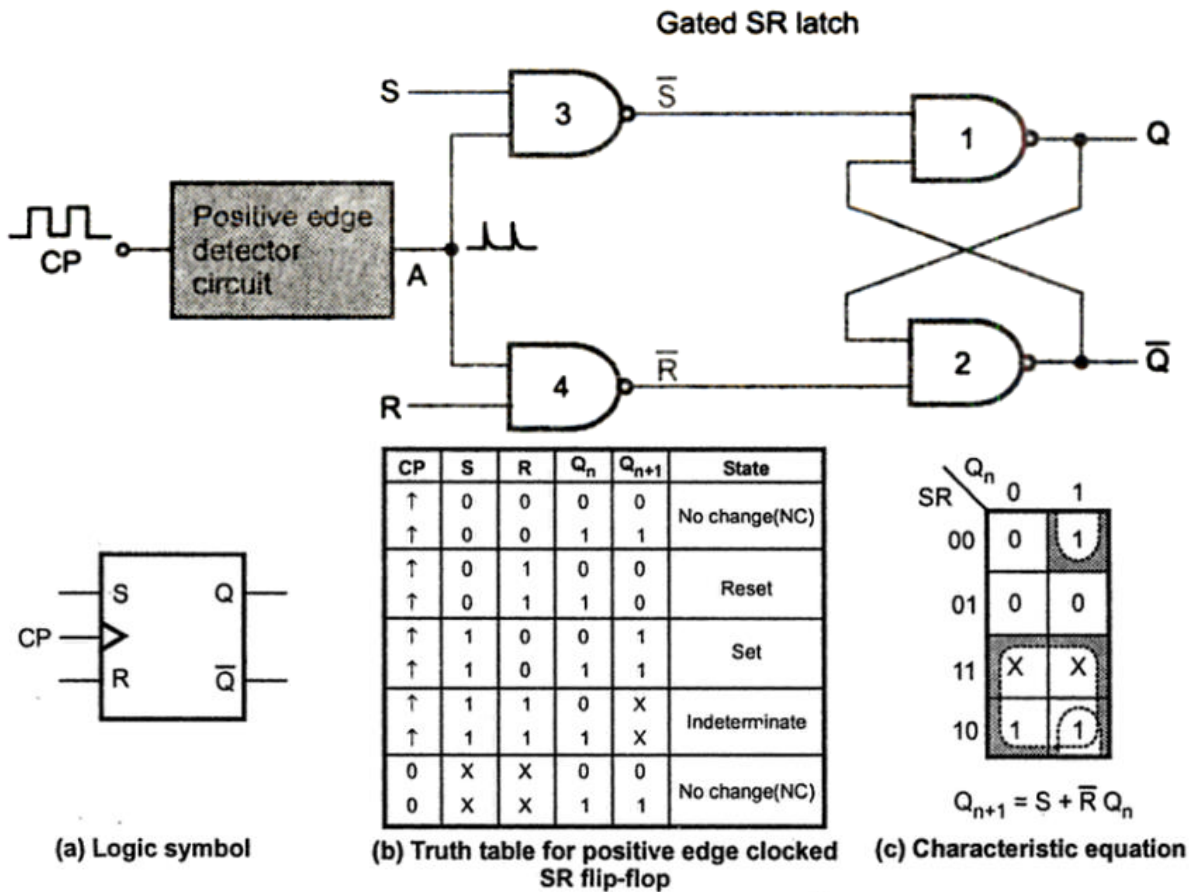
The Fig. shows the positive edge triggered clocked SR flip-flop. The circuit is similar to SR latch except enable signal is replaced by the clock pulse (CP) followed by the positive edge detector circuit. The edge detector circuit is a differentiator.

Without a clock pulse, the S and R inputs cannot affect the output. When S is HIGH and R is LOW, the Q output goes HIGH on the positive-going edge of the clock pulse and the flip-flop is SET. (If it is already in SET state, it remains SET). When S is LOW and R is HIGH, the Q output goes LOW on the positive-going edge of the clock pulse and the flip-flop is RESET, i.e. cleared. (If it is already in RESET state, it remains RESET). When both S and R are LOW, the output does not change from its prior state (If it is in SET state, it remains SET and if it is in RESET state, it remains RESET). When both S and R are HIGH simultaneously, an invalid condition exists. The basic operation described above is illustrated in Figure

The Fig. shows input and output waveforms for positive edge triggered clocked SR flip-flop. As shown in Fig., the circuit output responds to the S and R inputs only at the positive edges of the clock pulse. At any other instants of time, the SR flip-flop will not respond to the changes in input.

The Fig. shows the logic symbol and truth table of clocked SR flip-flop.

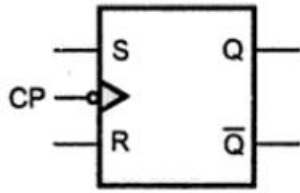
BASIC ELECTRONICS ENGINEERING



b) Negative edge triggered SR flip-flop:

In the negative edge triggered SR flip-flop, the negative edge detector circuit is used and the circuit output responds at the negative edges of the clock pulse. The Fig. shows the logic symbol, truth table, and input and output waveforms for negative edge triggered SR flip-flop. The bubble at the dock input indicates that the flip-flop is negative edge triggered.

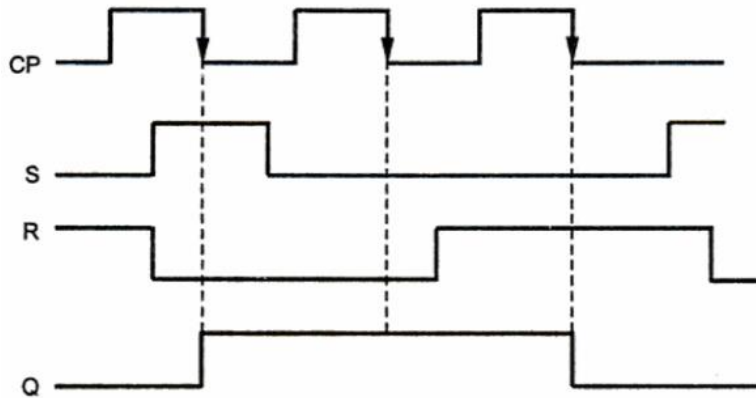
BASIC ELECTRONICS ENGINEERING



(a) Logic symbol

CP	S	R	Q_n	Q_{n+1}	State
↓	0	0	0	0	No change(NC)
↓	0	0	1	1	
↓	0	1	0	0	Reset
↓	0	1	1	0	
↓	1	0	0	1	Set
↓	1	0	1	1	
↓	1	1	0	X	Indeterminate
↓	1	1	1	X	
0	X	X	0	0	No change(NC)
0	X	X	1	1	

(b) Truth Table for negative edge clocked SR flip-flop



EXCITATION TABLE OF R-S FLIP-FLOP

During the design process we know, from the transition table, the sequence of states, i.e., the transition from each present state to its corresponding next state. From this information we wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need a table that lists the required inputs for a given change of state. Such a table is known as an excitation table of the flip-flop.

We can derive the excitation tables for flip-flops from their truth tables. The excitation table consists of two columns Q_n and Q_{n+1} , and a column for each input to show how the required transition can be achieved.

The following tables show the truth table and excitation tables for RS flip-flop, respectively. As shown in the table, there are four possible transitions from the present state to the next state. For each transition, the required input condition is derived from the information available in the truth table. Let us see the process by examining each case.

R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	*

(a) RS truth table

Q_n	Q_{n+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

(b) RS excitation table

BASIC ELECTRONICS ENGINEERING

0 → 0 Transition:

The present state of the flip-flop is 0 and is to remain 0 when a clock pulse is applied. Looking at truth table of RS flip-flop we can understand that, this can happen either when $R = S = 0$ (no-change condition) or when $R = 1$ and $S = 0$. Thus, S has to be at 0, but R can be at either level. The table indicates this with a "0" under S and an "X" (don't care) under R.

0 → 1 Transition:

The present state is 0 and is to change to 1. This can happen only when $S = 1$ and $R = 0$ (set condition). Therefore, S has to be 1 and R has to be 0 for this transition to occur.

1 → 0 Transition:

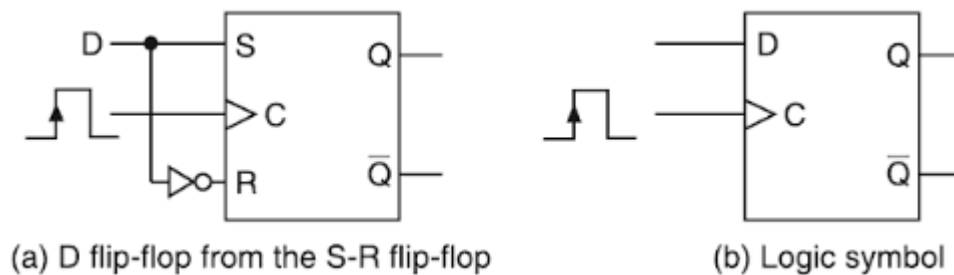
The present state is 1 and is to change to a 0. This can happen only when $S = 0$ and $R = 1$ (reset condition). Therefore, S has to be 0 and R has to be 1 for this transition to occur.

1 → 1 Transition:

The present state is 1 and is to remain 1. This can happen either when $S = 1$ and $R = 0$ (set condition) or when $S = 0$ and $R = 0$ (no change condition). Thus R has to be 0, but S can be at either level. The table indicates this with "X" under and "0" under R.

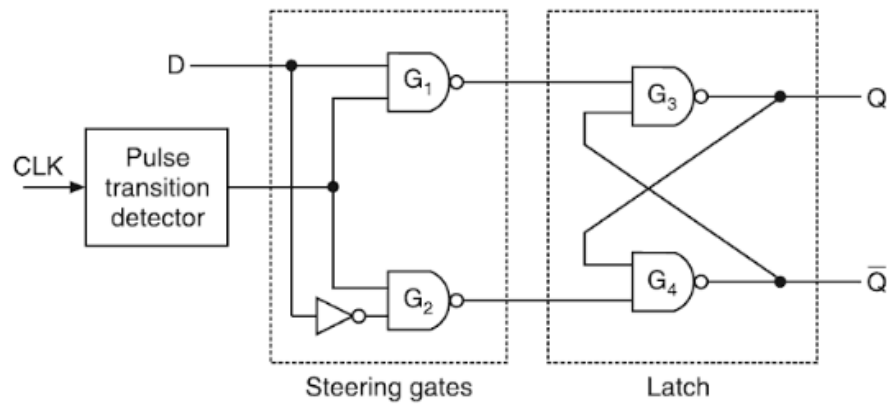
CLOCKED D FLIP-FLOP:

The edge-triggered D flip-flop has only one input terminal. The D flip-flop may be obtained from an S-R flip-flop by just putting one inverter between the S and R terminals. Figures show the logic symbol and the truth table of a positive edge-triggered D flip-flop. Note that, this flip-flop has only one synchronous control input in addition to the clock. This is called the D (data) input. The operation of the D flip-flop is very simple. The output Q will go to the same state that is present on the D input at the positive-going transition of the clock pulse. In other words, the level present at D will be stored in the flip-flop at the instant the positive-going transition occurs. That is, if D is a 1 and the clock is applied, Q goes to a 1 and \bar{Q} to a 0 at the rising edge of the pulse and thereafter remain so. If D is a 0 and the clock is applied, Q goes to a 0 and \bar{Q} to a 1 at the rising edge of the clock pulse and thereafter remain so.



The internal circuit of edge triggered D-Flipflop is shown in the fig.

BASIC ELECTRONICS ENGINEERING



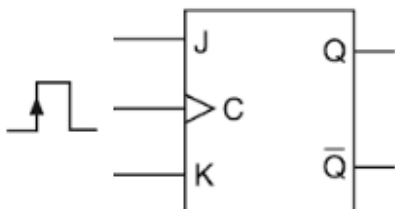
C	D	Q_n	Q_{n+1}	State
\uparrow	0	0	0	Reset
\uparrow	0	1	0	
\uparrow	1	0	1	Set
\uparrow	1	1	1	
0	x	0	0	No Change (NC)
0	x	1	1	

(c) Truth table

3) CLOCKED J-K FLIP-FLOP:

The J-K flip-flop is very versatile and also the most widely used. The J and K designations for the synchronous control inputs have no known significance.

The functioning of the J-K flip-flop is identical to that of the S-R flip-flop, except that it has no invalid state like that of the S-R flip-flop. The logic symbol and the truth table for a positive edge-triggered J-K flip-flop are shown in Figure



(a) Logic symbol

C	J	K	Q_n	Q_{n+1}	State
\uparrow	0	0	0	0	No Change (NC)
\uparrow	0	0	1	1	
\uparrow	0	1	0	0	Reset
\uparrow	0	1	1	0	
\uparrow	1	0	0	1	Set
\uparrow	1	0	1	1	
\uparrow	1	1	0	1	Toggle
\uparrow	1	1	1	0	
0	x	x	0	0	No Change (NC)
0	x	x	1	1	

(b) Truth table

BASIC ELECTRONICS ENGINEERING

When $J = 0$ and $K = 0$, no change of state takes place even if a clock pulse is applied.

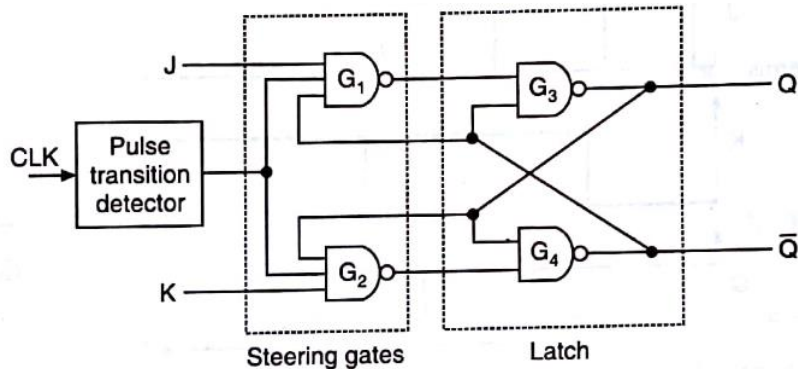
When $J = 0$ and $K = 1$, the flip-flop resets at the positive-going edge of the clock pulse.

When $J = 1$ and $K = 0$, the flip-flop sets at the positive-going edge of the clock pulse.

When $J = 1$ and $K = 1$, the flip-flop toggles, i.e. goes to the opposite state at the positive-going edge of the clock pulse. In this mode, the flip-flop toggles or changes state for each occurrence of the positive-going edge of the clock pulse.

A negative edge-triggered J-K flip-flop operates in the same way as a positive edge-triggered J-K flip-flop except that the change of state takes place at the negative-going edge of the clock pulse. In the truth-table of a negative edge-triggered J-K flip-flop the arrows point downwards.

The internal circuit of edge triggered JK-Flipflop is shown in the fig.



RACE AROUND CONDITION:

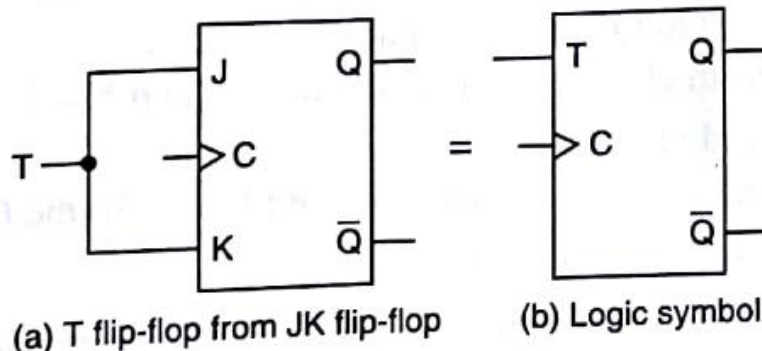
When $j = 1$ $k = 1$ and $clk = 1$; Q output will toggle as long as CLK is high. Thus the output will be unstable creating a race-around problem with this basic JK circuit.

This problem is avoided by ensuring that the clock input is at logic "1" only for a very short time, or to produce a more sophisticated JK flip-flop circuit called a Master-slave flip-flop.

4) CLOCKED T- FLIPFLOP:

A T flip-flop has a single control input, labelled T for toggle.

When T is HIGH, the flip-flop toggles on every new clock pulse. When T is LOW, the flip-flop remains in whatever state it was before. Although T flip-flops are not widely available commercially, it is easy to convert a J-K flip-flop to the functional equivalent of a T flip-flop by just connecting J and K together and labelling the common connection as T. Thus, when $T = 1$, we have $J = K = 1$, and the flip-flop toggles. When $T = 0$, we have $J = K = 0$, and so there is no change of state. The logic symbol and the truth table of a T flip-flop are shown in Figure



BASIC ELECTRONICS ENGINEERING

C	T	Q_n	Q_{n+1}	State
\uparrow	0	0	0	No Change (NC)
\uparrow	0	1	1	
\uparrow	1	0	1	Toggle
\uparrow	1	1	0	
0	x	0	0	No Change (NC)
0	x	1	1	

(c) Truth table

EXCITATION TABLE OF J-K FLIP-FLOP

The truth table and excitation table for JK flip-flop are shown in Table respectively. Let us examine each case.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

(a) JK truth table

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(b) JK excitation table

0 → 0 Transition : When both present state and next state are 0, the J input must remain at 0 and the K input can be either 0 and 1.

0 → 1 Transition : The present state is 0 and is to change to 1. This can happen either when $J = 1$ and $K = 0$ (set condition) or when $J = K = 1$ (toggle condition). Thus, J has to be 1, but K can be at either level for this transition to occur.

1 → 0 Transition : The present state is 1 and is to change to 0. This can happen either when $J = 0$ and $K = 1$ or when $J = K = 1$. Thus, K has to be 1 but J can be at either level.

1 → 1 Transition : When both present state and next are 1, the K input must remain at 0 while the J input can be 0 or 1.

EXCITATION TABLE OF 'D' FLIP-FLOP

The Table show the truth table and excitation table for D flip-flop, respectively. In D flip-flop, the next state is always equal to the D input and it is independent of the present state. Therefore, D must be 0 if Q_{n+1} has to be 0, and 1 if Q_{n+1} has to be 1, regardless of the value of Q_n .

BASIC ELECTRONICS ENGINEERING

D	Q_{n+1}
0	0
1	1

D Truth table

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

D Excitation table

EXCITATION TABLE OF 'T' FLIP-FLOP

The Table show the truth table and the excitation table for T flip-flop, respectively. We know that when input $T = 1$, the state of the flip-flop is complemented; when $T = 0$, the state of the flip-flop remains unchanged. Therefore, for $0 \rightarrow 0$ and $1 \rightarrow 1$ transitions T must be 0 and for $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions T must be 1.

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

T Truth table

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

T Excitation table

SHIFT REGISTERS

Registers are the devices used to store the data bits. The bits stored in such registers can be made to move within the registers and/or in/out of the registers by applying clock pulses. Such registers are called shift registers. An n-bit shift register can be formed by cascading 'n' flip-flops where each [flip-flop](#) stores a single bit of information. Here the clear line is used to reset each flip-flop which in turn clears the entire register.

Data may be available in parallel form or in serial form. Multi-bit data is said to be in parallel form when all the bits are available (accessible) simultaneously. The data is said to be in serial form when the data bits appear sequentially (one after the other, in time) at a single terminal. Data may also be transferred in parallel form or in serial form. Parallel data transfer is the simultaneous transmission of all bits of data from one device to another. Serial data transfer is the transmission of one bit of data at a time from one device to another. Serial data must be transmitted under the synchronization of a clock, since the clock provides the means to specify the time at which each new bit is sampled.

As a flip-flop (FF) can store only one bit of data, a 0 or a 1, it is referred to as a single-bit register. When more bits of data are to be stored, a number of FFs are used. A register is a set of FFs used to store binary data. The storage capacity of a register is the number of bits (Is and Os) of digital data it can retain. Loading a register means setting or resetting the individual FFs, i.e. inputting data into the register so that their states correspond to the bits of data to be stored.

BASIC ELECTRONICS ENGINEERING

Loading may be serial or parallel. In serial loading, data is transferred into the register in serial form, i.e. one bit at a time, whereas in parallel loading, the data is transferred into the register in parallel form meaning that all the FFs are triggered into their new states at the same time. Parallel input requires that the SET and/or RESET controls of every FF be accessible.

A register may output data either in serial form or in parallel form. Serial output means that the data is transferred out of the register, one bit at a time serially. Parallel output means that the entire data stored in the register is available in parallel form, and can be transferred out at the same time.

Shift registers are a type of logic circuits closely related to counters. They are used basically for the storage and transfer of digital data. The basic difference between a shift register and a counter is that, a shift register has no specified sequence of states except in certain very specialized applications, where as a counter has a specified sequence of states.

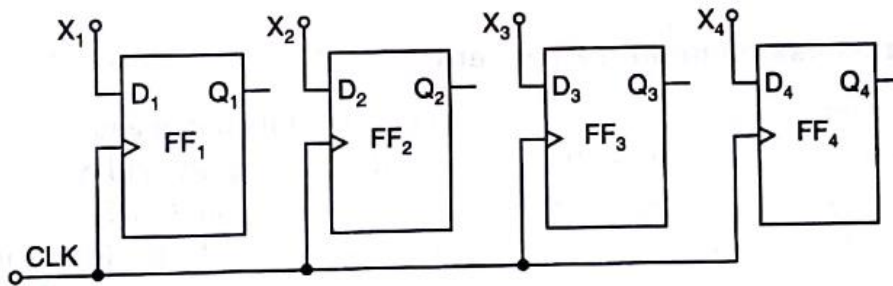
i) Buffer Registers:

Some registers do nothing more than storing a binary word. The buffer register is the simplest of registers. It simply stores the binary word. The buffer may be a controlled buffer. Most of the buffer registers use D flip-flops.

The Figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the input terminals, i.e. the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes:

$$Q_4 Q_3 Q_2 Q_1 = X_4 X_3 X_2 X_1$$
$$Q = X$$



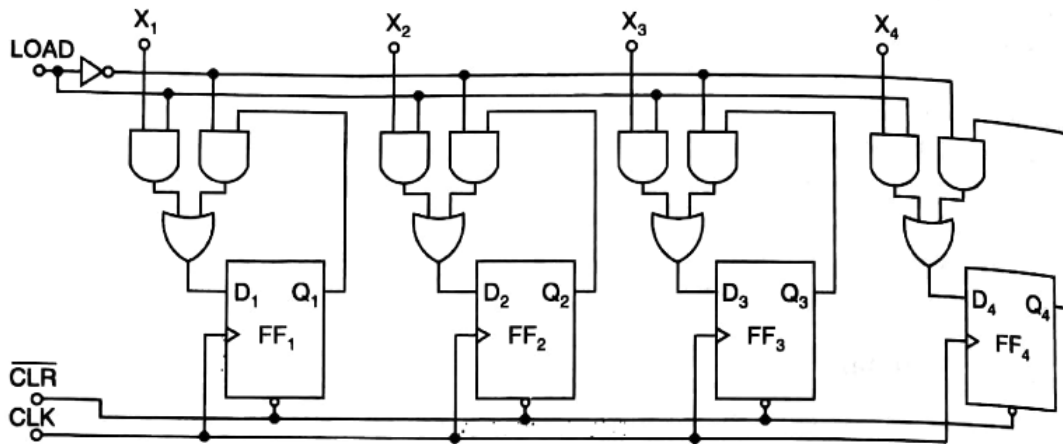
ii) Controlled Buffer Register :

The Figure shows a controlled buffer register. If \overline{CLR} goes LOW, all the FFs are RESET and the output becomes, $Q = 0000$.

When \overline{CLR} is HIGH, the register is ready for action. LOAD is the control input. When LOAD is HIGH, the data bits X can reach the D inputs of FFs. At the positive-going edge of the next clock pulse, the register is loaded, i.e.

$$Q_4 Q_3 Q_2 Q_1 = X_4 X_3 X_2 X_1$$
$$Q = X$$

BASIC ELECTRONICS ENGINEERING



When LOAD is LOW, the X bits cannot reach the FFs. At the same time, the inverted signal LOAD is HIGH. This forces each flip-flop output to feed back to its data input. Therefore, data is circulated or retained as each clock pulse arrives. In other words, the contents of the register remain unchanged in spite of the clock pulses. Longer buffer registers can be built by adding more FFs.

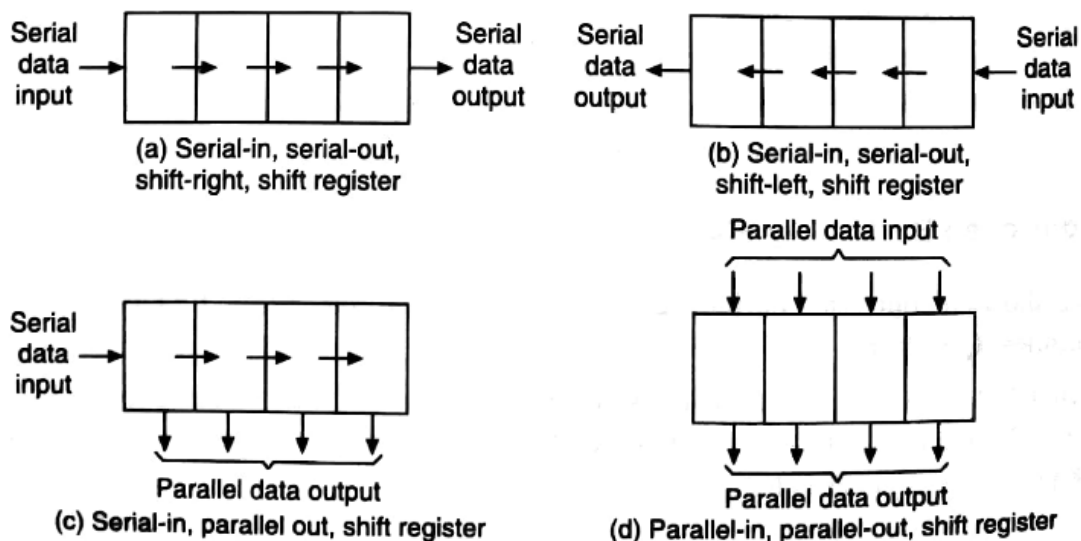
iii) Data Transmission in Shift Registers :

A number of FFs connected together such that data may be shifted into and shifted out of them is called a shift register. Data may be shifted into or out of the register either in serial form or in parallel form.

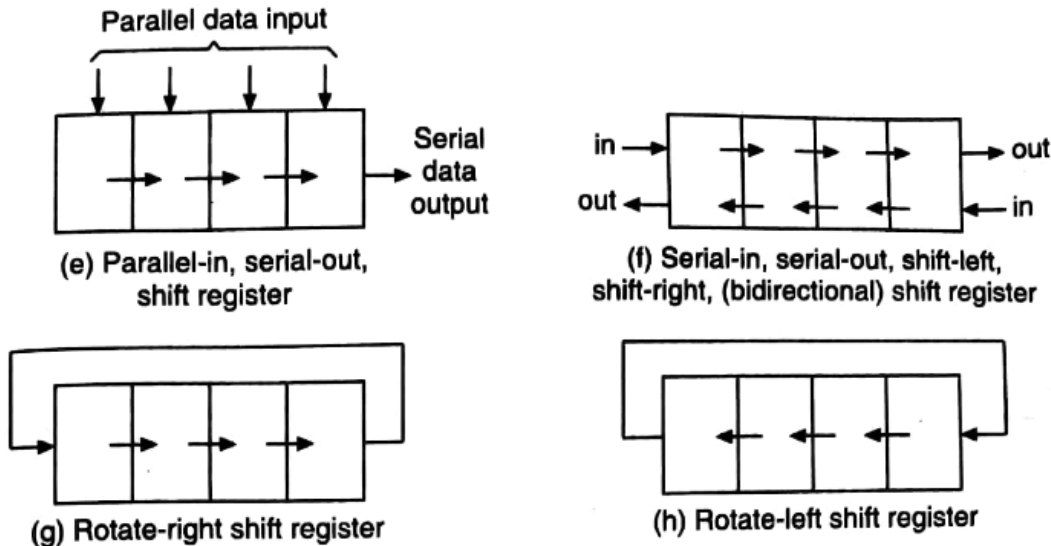
So, there are four basic types of shift registers:

- a) serial-in, serial-out (SISO)
- b) serial-in, parallel-out (SIPO)
- c) parallel-in, serial-out (PISO)
- d) parallel-in, parallel-out (PIPO)

The process of data shifting in these registers is illustrated in Figure. All of these configurations are commercially available as TTL MSU LSI circuits. Data may be rotated left or right. Data may be shifted from left to right or right to left at will, i.e. in a bidirectional way. Also, data may be shifted in serially (in either way) or in parallel and shifted out serially (in either way) or in parallel.



BASIC ELECTRONICS ENGINEERING

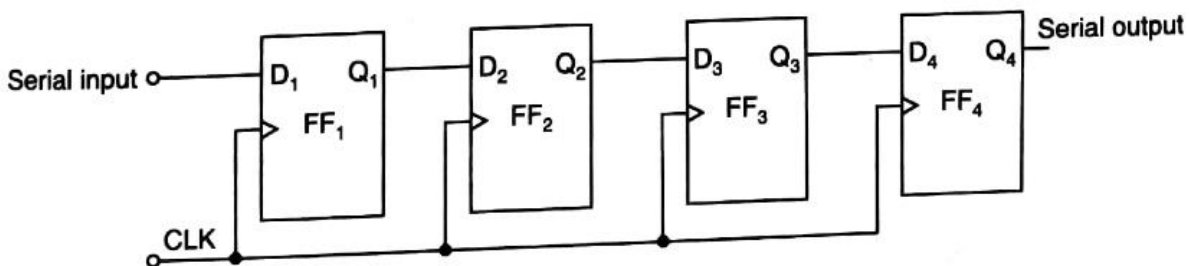


iv) Serial-in, Serial-out, Shift Register:

This type of shift register accepts data serially, i.e. one bit at a time, and also outputs data serially. The logic diagram of a 4-bit serial-in, serial-out, shift-right, shift register is shown in Figure.

With four stages, i.e. four FFs, the register can store upto four bits of data. Serial data is applied at the D input of the first FF. The Q output of the first FF is connected to the D input of the second FF, the Q output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF. The data is outputted from the Q terminal of the last FF.

When serial data is transferred into a register, each new bit is clocked into the first FF at the positive-going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. The bit that was stored by the second FF is transferred to the third FF, and so on. The bit that was stored by the last FF is shifted out.

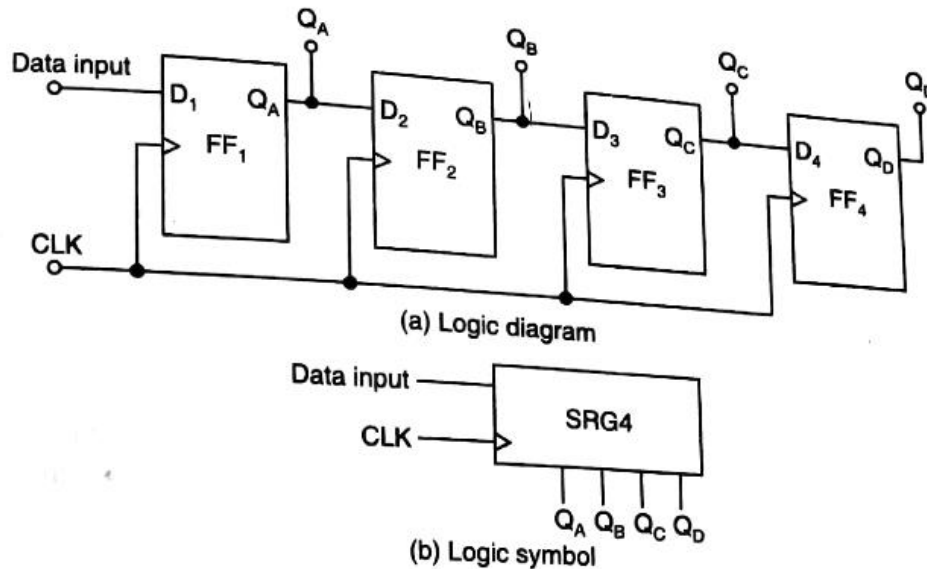


v) Serial-in, Parallel-out, Shift Register:

The Figure shows the logic diagram and the logic symbol of a 4-bit serial-in, parallel-out, shift register. In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output. The serial-in, parallel-out, shift register can be used as a serial-in, serial-out, shift register if the output is taken from the Q terminal of the last FF.

BASIC ELECTRONICS ENGINEERING

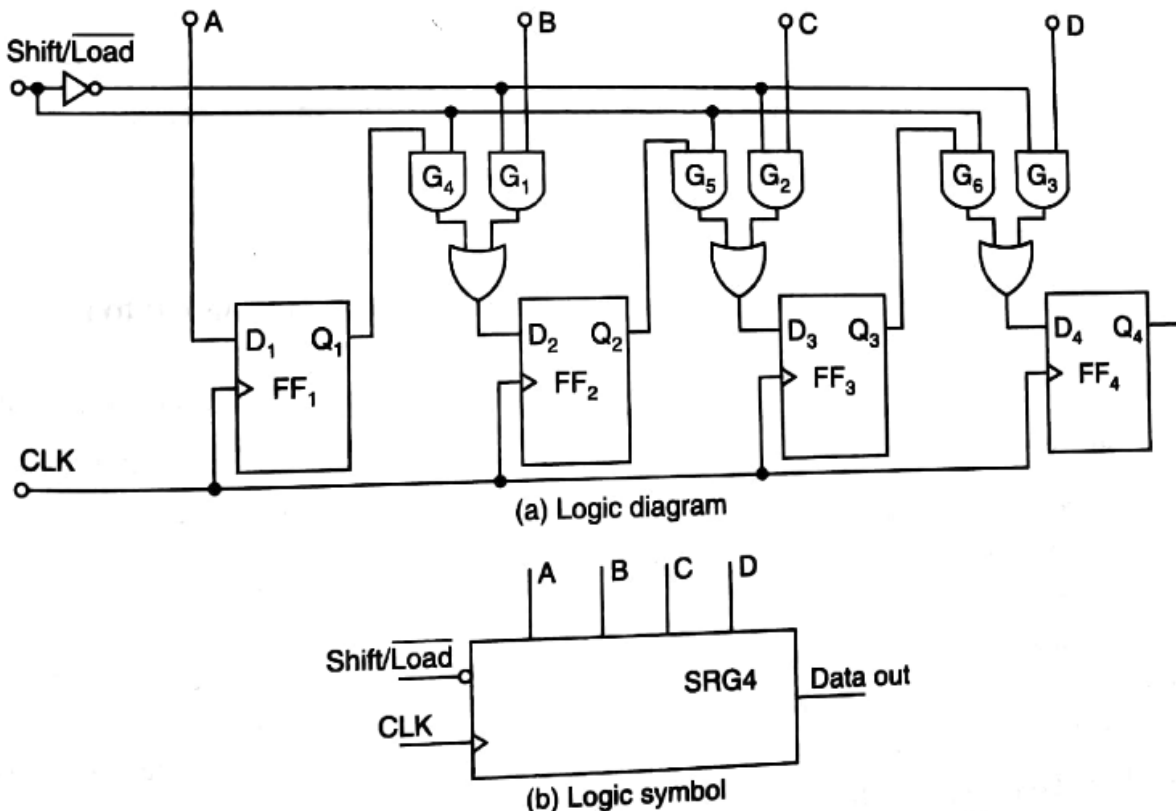


vi) Parallel-in, Serial-out, Shift Register:

For a parallel-in, serial-out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data inputs, but the data bits are transferred out of the register serially, i.e. on a bit-by-bit basis over a single line.

The Figure illustrates a 4-bit parallel-in, serial-out, shift register using D FFs. There are four data lines A, B, C, and D through which the data is entered into the register in parallel form. The signal Shift/LOAD allows

- (a) the data to be entered in parallel form into the register and
- (b) the data to be shifted out serially from terminal Q₄.



BASIC ELECTRONICS ENGINEERING

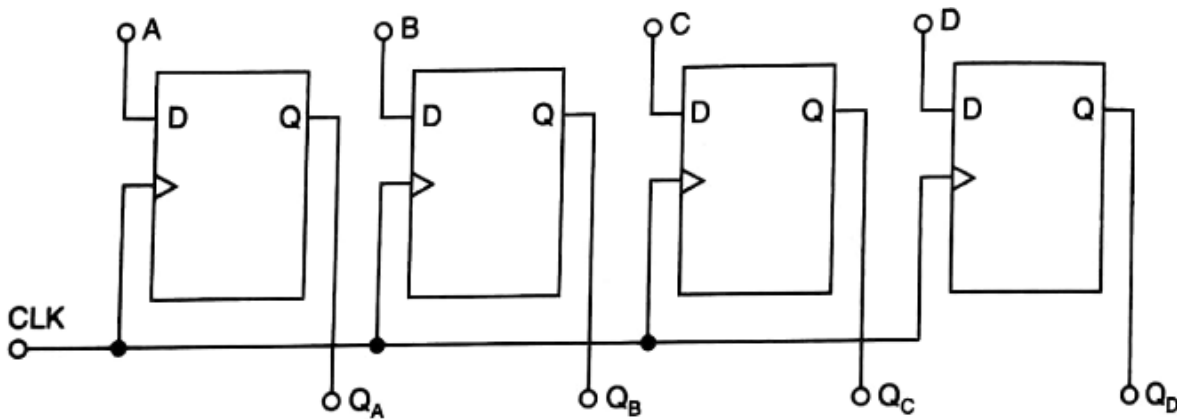
When Shift/ \overline{LOAD} line is HIGH, gates G_1 , G_2 , and G_3 are disabled, but gates G_4 , G_5 , and G_6 are enabled allowing the data bits to shift-right from one stage to the next. When Shift/ \overline{LOAD} line is LOW, gates G_4 , G_5 , and G_6 are disabled, whereas gates G_1 , G_2 , and G_3 are enabled allowing the data input to appear at the D inputs of the respective FFs.

When a clock pulse is applied, these data bits are shifted to the Q output terminals of the FFs and, therefore, data is inputted in one step. The OR gate allows either the normal shifting operation or the parallel data entry depending on which AND gates are enabled by the level on the Shift/ \overline{LOAD} input.

vii) Parallel-in, Parallel-out, Shift Register :

In a parallel-in, parallel-out, shift register, the data is entered into the register in parallel form and also the data is taken out of the register in parallel form. Immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs.

The Figure shows a 4-bit parallel-in, parallel-out, shift register using D FFs. Data is applied to the D input terminals of the FFs. When a clock pulse is applied, at the positive-going edge of that pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

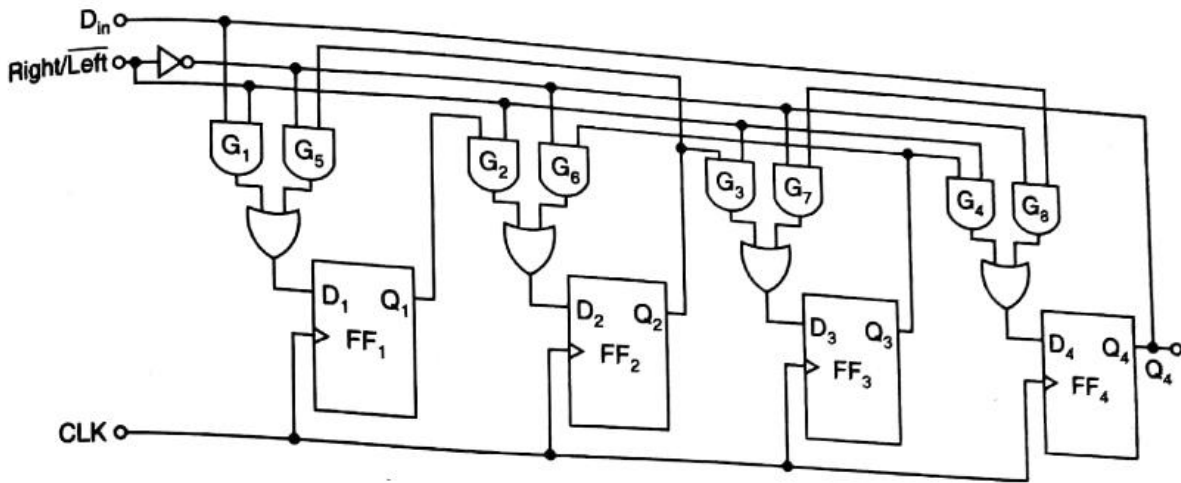


viii) Bidirectional Shift Register:

A bidirectional shift register is one in which the data bits can be shifted from left to right or from right to left.

The Figure shows the logic diagram of a 4-bit serial-in, serial-out, bidirectional (shift-left, shift-right) shift register. Right/ \overline{Left} is the mode signal. When Right/ \overline{Left} is a 1, the logic circuit works as a shift-right shift register. When Right/ \overline{Left} is a 0, it works as a shift-left shift register. The bidirectional operation is achieved by using the mode signal and two AND gates and one OR gate for each stage as shown in Figure.

BASIC ELECTRONICS ENGINEERING



A HIGH on the Right/Left control input enables the AND gates G_1 , G_2 , G_3 , and G_4 and disables the AND gates G_5 , G_6 , G_7 , and G_8 , and the state of Q output of each FF is passed through the gate to the D input of the following FF. When a clock pulse occurs, the data bits are then effectively shifted one place to the right.

A LOW on the Right/Left control input enables the AND gates G_5 , G_6 , G_7 , and G_8 and disables the AND gates G_1 , G_2 , G_3 , and G_4 and the Q output of each FF is passed through the gate to the D input of the preceding FF. When a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence the circuit works as a bidirectional shift register.

ix) Universal Shift Registers :

A register capable of shifting in one direction only is a unidirectional shift register. One that can shift in both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift register. So a universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be either in serial form or in parallel form.

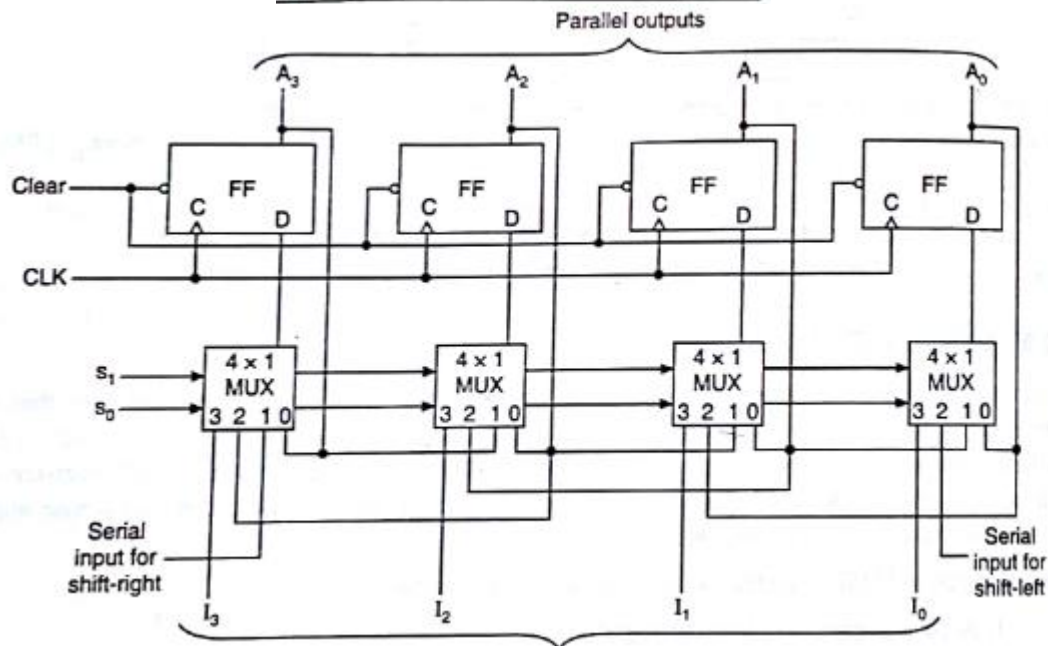
A universal shift register can be realized using multiplexers. The Figure shows the logic diagram of a 4-bit universal shift register. It consists of four D flip-flops and four multiplexers. The four multiplexers have two common selection inputs S_1 and S_0 . Input '0' in each multiplexer is selected when $S_1S_0 = 00$, input '1' is selected when $S_1S_0 = 01$, and input 2 is selected when $S_1S_0 = 10$ and input 3 is selected when $S_1S_0 = 11$. The selection inputs control the mode of operation of the register according to the function entries in Table. When $S_1S_0 = 0$, the present value of the register is applied to the D inputs of flip-flops. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs.

When $S_1S_0 = 01$, terminal '1' of the multiplexer inputs have a path to the D inputs of the flip-flops. This causes a shift-right operation, with the serial input transferred into flip-flop A_4 . When $S_1S_0 = 10$, a shift-left operation results with the other serial input going into flip-flop A_1 . Finally when $S_1S_0 = 11$, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge.

BASIC ELECTRONICS ENGINEERING

Function table for the register

Mode control		Register operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load



COUNTERS

A digital counter is a set of flip-flops (FFs) whose states change in response to pulses applied at the input to the counter. The FFs are interconnected such that their combined state at any time is the binary equivalent of the total number of pulses that have occurred up to that time. Thus, as its name implies, a counter is used to count pulses.

A counter can also be used as a frequency divider to obtain waveforms with frequencies that are specific fractions of the clock frequency. They are also used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters, etc.

Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters. The ripple counter is the simplest type of counter, the easiest to design and requires the least amount of hardware. In ripple counters, the FFs within the counter are not made to change the states at exactly the same time. This is because the FFs are not triggered simultaneously. The clock does not directly control the time at which every stage changes state. An asynchronous counter uses 'T' FFs to perform a counting function. The actual hardware used is usually J-K FFs connected in toggle mode, i.e. with Js and Ks connected to logic 1.

The asynchronous counter has a disadvantage, in so far as the unwanted spikes are concerned. This limitation is overcome in parallel counters. The asynchronous counter is called ripple counter because when the counter, for example, goes from 1111 to 0000, the first stage

BASIC ELECTRONICS ENGINEERING

causes the second to flip, the second causes the third to flip, and the third causes the fourth to flip, and so on. In other words, the transition of the first stage ripples through to the last stage.

If there is a gate that will AND during any state, a brief spike will be seen at the gate output every time the counter goes from 1111 to 0000.

Ripple counters are also called serial or series counters.

Synchronous counters are clocked such that each FF in the counter is triggered at the same time. This is accomplished by connecting the clock line to each stage of the counter. Synchronous counters are faster than asynchronous counters, because the propagation delay involved is less.

COMPARISON BETWEEN ASYNCHRONOUS COUNTERS & SYNCHRONOUS COUNTERS

S.No.	Asynchronous Counter	Synchronous Counter
1.	All flip-flops are not clocked simultaneously.	All flip-flops are clocked simultaneously.
2.	Output of first flip-flop drives the clock of the second flip-flop, the output of second drives the third and so on.	there is no interconnection between an output of one flip-flop and clock of next flip-flop.
3.	The settling time of asynchronous counter is cumulative sum of individual flip-flops.	The settling time of synchronous counter is equal to the highest settling time of all flip-flops.
4.	It is also known as a serial counter.	It is also known as a parallel counter.
5.	Its design and implementation are very simple.	Its design and implementation become tedious and complex as the number of states increases.
6.	It is slow in speed as compared to synchronous counter.	It is faster in speed as compared to asynchronous counter.

UP/DOWN COUNTER:

A counter may be an up-counter or a down-counter. An up-counter is a counter which counts in the upward direction, i.e. 0, 1, 2, 3,..., N.

A down-counter is a counter which counts in the downward direction, i.e. N, N- 1, N- 2, N- 3, ..., 1, 0.

Each of the counts of the counter is called the state of the counter. The number of states through which the counter passes before returning to the starting state is called the **modulus** of the counter. Hence, the modulus of a counter is equal to the total number of distinct states (counts) including zero that a counter can store.

In other words, the number of input pulses that causes the counter to reset to its initial count is called the modulus of the counter. Since a 2-bit counter has 4 states, it is called a mod-4 Counter. It divides the input clock signal frequency by 4; therefore, it is also called a divide-by-4

BASIC ELECTRONICS ENGINEERING

counter. It requires two FFs. Similarly, a 3-bit counter uses 3 FFs and has $2^3 = 8$ states. It divides the input clock frequency by 2^3 , i.e. 8.

In general, an n-bit counter will have 'n' FFs and 2^n states, and divides the input frequency by 2^n . Hence, it is a divide-by- 2^n counter.

MOD-N COUNTER:

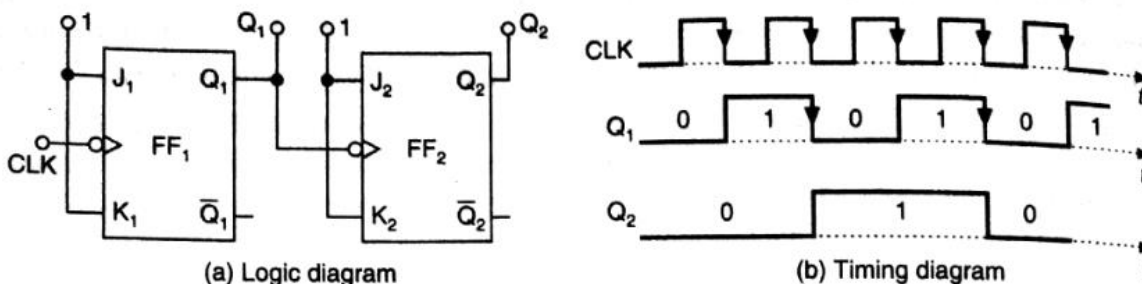
A counter may have a shortened modulus. This type of counter does not utilize all the possible states. Some of the states are unutilized, i.e. invalid. The number of Flip-flops required to construct a mod-N counter equals the smallest it for which $N \leq 2^n$. A mod-N counter divides the input frequency by N, hence, it is called a divide-by-N counter.

In an asynchronous counter, the invalid states are bypassed by providing a suitable feedback. In a synchronous counter, the invalid states are taken care of by treating the corresponding excitations as don't cares.

ASYNCHRONOUS COUNTERS

i) Two-bit ripple up-counter using negative edge-triggered flip-flops:

The 2-bit up-counter counts in the order 0, 1, 2, 3, 0, 1,..., i.e. 00, 01, 10, 11, 00, 01,...,etc. The Figure shows a 2-bit ripple up-counter, using negative edge-triggered J-K FFs, and its timing diagram.



The counter is initially reset to 00. When the first clock pulse is applied, FF₁ toggles at the negative-going edge of this pulse, therefore, Q₁ goes from LOW to HIGH. This becomes a positive-going signal at the clock input of FF₂. So, FF₂ is not affected, and hence, the state of the counter after one clock pulse is Q₁ = 1 and Q₂ = 0, i.e. 01.

At the negative-going edge of the second clock pulse, FF₁ toggles. So, Q₁ changes from HIGH to LOW and this negative-going signal applied to CLK of FF₂ activates FF₂, and hence, Q₂ goes from LOW to HIGH. Therefore, Q₁ = 0 and Q₂ = 1, i.e. 10 is the state of the counter after the second clock pulse.

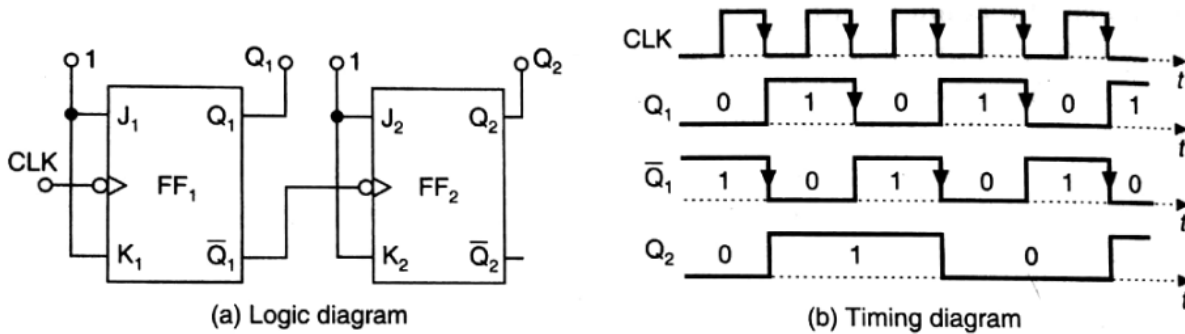
At the negative-going edge of the third clock pulse, FF₁ toggles. So Q₁ changes from 0 to 1. This becomes a positive-going signal to FF₂, hence, FF₂ is not affected. Therefore, Q₂ = 1 and Q₁ = 1, i.e. 11 is the state of the counter after the third clock pulse. At the negative-going edge of the fourth clock pulse, FF₁ toggles. So, Q₁ goes from 1 to 0. This negative-going signal at Q₁ toggles FF₂, hence, Q₂ also changes from 1 to 0. Therefore, Q₂ = 0 and Q₁ = 0 i.e. 00 is the state of the counter after the fourth clock pulse. For subsequent clock pulses, the counter goes through the

BASIC ELECTRONICS ENGINEERING

same sequence of states. So, it acts as a mod-4 counter with Q_1 as the LSB and Q_2 as the MSB. The counting sequence is thus 00, 01, 10, 11, 00, 01, ..., etc.

ii) Two-bit ripple down-counter using negative edge-triggered flip-flops:

A 2-bit down-counter counts in the order 0, 3, 2, 1, 0, 3, ..., i.e. 00, 11, 10, 01, 00, 11, etc. The Figure shows a 2-bit ripple down-counter, using negative-edge triggered J-K FFs, and its timing diagram.



For down counting, \bar{Q}_1 of FF₁ is connected to the clock of FF₂. Let initially all the FFs be reset, i.e. let the count be 00. At the negative-going edge of the first clock pulse, FF₁ toggles, so, Q_1 goes from 0 to 1 and \bar{Q}_1 goes from 1 to 0. This negative-going signal at \bar{Q}_1 applied to the clock input of FF₂, toggles FF₂ and, therefore, Q_2 goes from 0 to 1. So, after one clock pulse $Q_2 = 1$ and $Q_1 = 1$, i.e. the state of the counter is 11.

At the negative-going edge of the second clock pulse, Q_1 changes from 1 to 0 and \bar{Q}_1 from 0 to 1. This positive-going signal at \bar{Q}_1 , does not affect FF₂ and, therefore, Q_2 remains at 1. Hence, the state of the counter after the second clock pulse is 10.

At the negative-going edge of the third clock pulse, FF₁ toggles. So, Q_1 goes from 0 to 1 and \bar{Q}_1 from 1 to 0. This negative-going signal at \bar{Q}_1 toggles FF₂ and, so, Q_2 changes from 1 to 0. Hence, the state of the counter after the third clock pulse is 01.

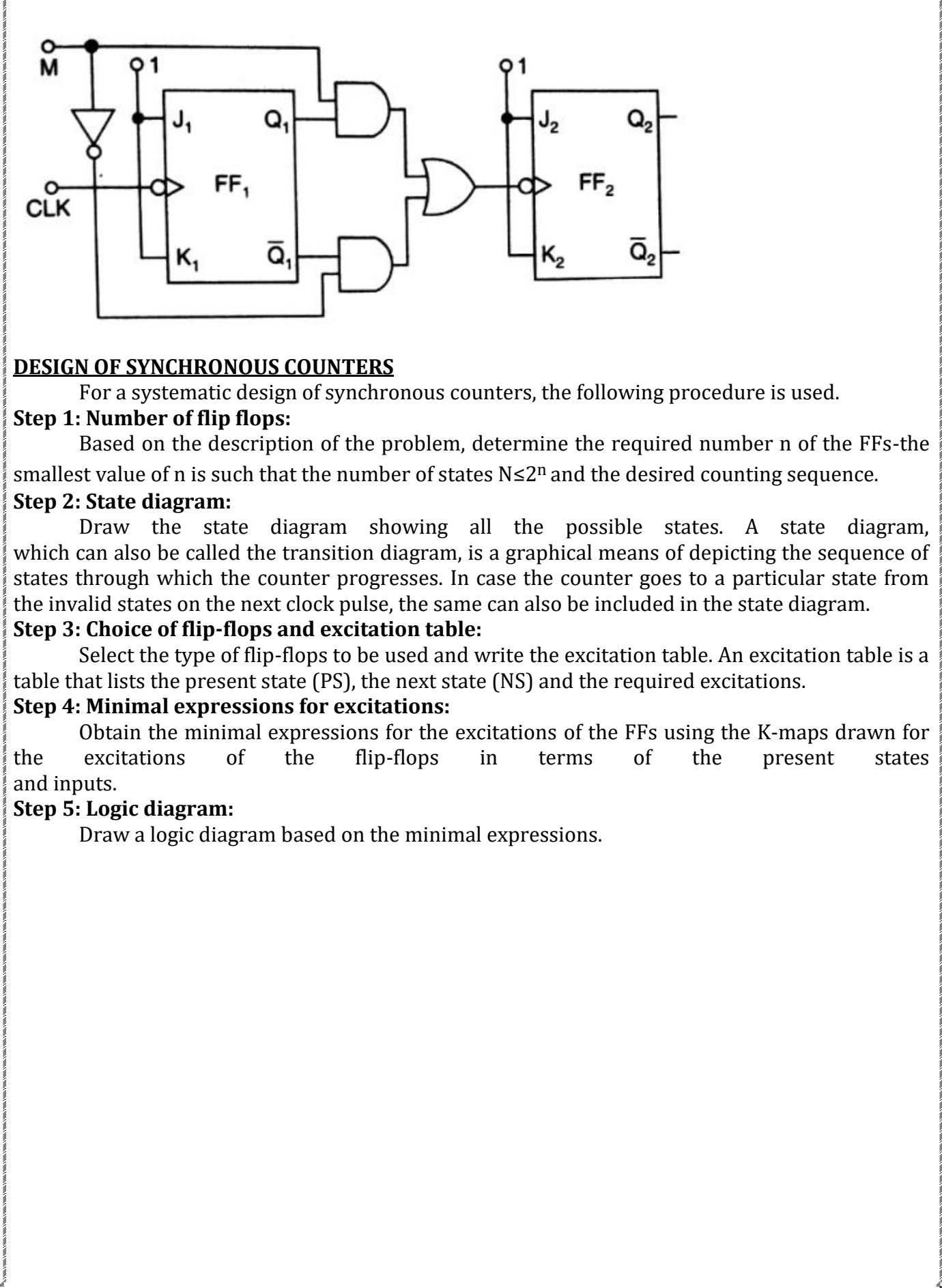
At the negative-going edge of the fourth clock pulse, FF₁ toggles. So, Q_1 goes from 1 to 0 and \bar{Q}_1 from 0 to 1. This positive-going signal at \bar{Q}_1 does not affect FF₂. So, Q_2 remains at a 0. Hence, the state of the counter after the fourth clock pulse is 00.

For subsequent clock pulses the counter goes through the same sequence of states, i.e. the counter counts in the order 00, 11, 10, 01, 00, and 11,

iii) Two-bit ripple up-down counter using negative edge-triggered flip-flops:

As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So a control signal or a mode signal M is required to choose the direction of count. When $M=1$ for up counting, Q_1 is transmitted to clock of FF₂ and when $M = 0$ for down counting, \bar{Q}_1 is transmitted to clock of FF₂. This is achieved by using two AND gates and one OR gate as shown in figure. The external clock signal is applied to FF₁.

$$\text{Clock signal to FF}_2 = (Q_1 \cdot \text{Up}) + (\bar{Q}_1 \cdot \text{Down}) = Q_1 M + \bar{Q}_1 \bar{M}$$



For a systematic design of synchronous counters, the following procedure is used.

Step 1: Number of flip flops:

Based on the description of the problem, determine the required number n of the FFs-the

smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence.

Draw the state diagram showing all the possible states. A state diagram,

which can also be called the transition diagram, is a graphical means of depicting the sequence of

Select the type of flip-flops to be used and write the excitation table. An excitation table is a

table that lists the present state (PS), the next state (NS) and the required excitations.

Obtain the minimal expressions for the excitations of the FFs using the K-maps drawn for

the excitations of the flip-flops in terms of the present states

Draw a logic diagram based on the minimal expressions.