



SRI G.C.S.R COLLEGE

(Affiliated to Dr. B. R. Ambedkar University, Srikakulam)

GMR Nagar, Rajam – 532127, Srikakulam (Dist.), A.P

T: +91 8941-251336, M: +91 89785 23866, F: +91 8941-251591,

www.srigcsrcollege.org



Department of ELECTRONICS

DIGITAL ELECTRONICS

II B.Sc. II – Semester

Electronics Paper-2

Name of the Student : _____

Roll Number : _____

Group : _____

Year/ Semester : _____

Prepared by:

K. Venugopala Rao M.Sc.(Tech), M. Tech

Lecturer in Electronics

SRI GCSR COLLEGE, GMR Nagar, Rajam

venugopalarao.k@sgsrc.edu.in,

+91-9603803168, +91-9182061933

Department of ELECTRONICS

Course Objectives

Students will learn:

1. Have a thorough understanding of the fundamental concepts and techniques used in digital electronics.
2. To understand number representation and conversion between different representation in digital electronic circuits.
3. To analyze logic processes and implement logical operations using combinational logic circuits.
4. To understand concepts of sequential circuits and to analyze sequential systems in terms of state machines.
5. To understand characteristics of memory and their classification.

After successful completion of the course student will be able to

1. Develop a digital logic and apply it to solve real life problems.
2. Analyze, design and implement combinational logic circuits.
3. Classify different semiconductor memories.
4. Analyze, design and implement sequential logic circuits.
5. To develop skill to build, and troubleshoot digital circuits.

Course Contents:**UNIT – I: NUMBER SYSTEM AND CODES**

Decimal, Binary, Hexadecimal, Octal, BCD, Conversions, Complements (1's, 2's, 9's and 10's), Addition, Subtraction, Gray, Excess-3 Code conversion from one to another.

UNIT – II: BOOLEAN ALGEBRA AND THEOREMS

Boolean Theorems, De-Morgan's laws. Digital logic gates, Multi-level NAND & NOR gates, Standard representation of logic functions (SOP and POS), Minimization Techniques (Karnaugh Map Method: 4,5 variables), don't care condition.

UNIT – III: COMBINATIONAL DIGITAL CIRCUITS

Adders-Half & full adder, Subtractor-Half and full subtractors, Parallel binary adder, Magnitude Comparator, Multiplexers (2:1,4:1) and De-multiplexers (1:2,1:4), Encoder (8-line-to-3-line) and Decoder (3-line-to-8-line). IC-LOGIC FAMILIES: TTL logic, DTL logic, RTL Logic, CMOS Logic families (NAND&NOR Gates), Bi-CMOS inverter

UNIT – IV: SEQUENTIAL DIGITAL CIRCUITS

Flip Flops: S-R FF, J-K FF, T and D type FFs, Master-Slave FFs, Excitation tables, Registers: - shift left register, shift right register, Counters - Asynchronous-Mod16, Mod-10, Mod-8, Down counter, Synchronous-4-bit & Ring counter

UNIT – V: MEMORY DEVICES

General Memory Operations, ROM, RAM (Static and Dynamic), PROM, EPROM, EEPROM, EAROM, PLA (Programmable logic Array), PAL (Programmable Array Logic)

UNIT – 1

NUMBER SYSTEM & CODES

1.1 INTRODUCTION TO ELECTRONIC SIGNALS

A **Signal** can be understood as "a representation that gives some information about the data present at the source from which it is produced." This is usually time varying. Hence, a signal can be a **source of energy which transmits some information**. This can easily be represented on a graph.

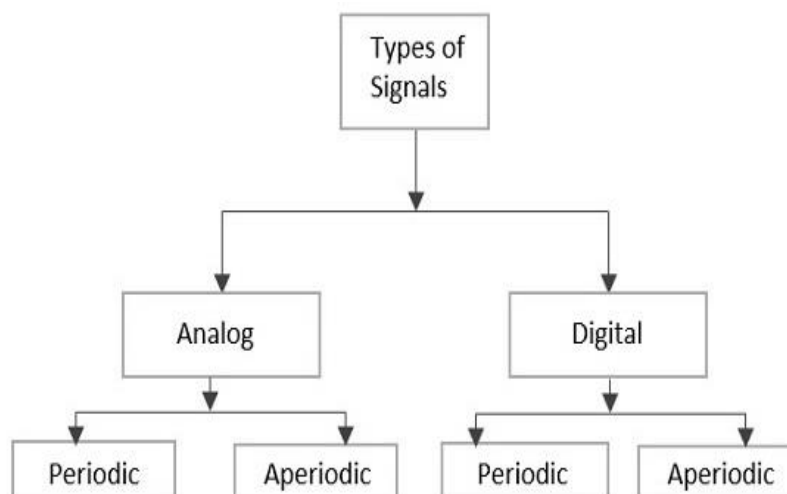
Examples

- An alarm gives a signal that it's time.
- A cooker whistle confirms that the food is cooked.
- A red light signals some danger.
- A traffic signal indicates your move.
- A phone rings signaling a call for you.

A signal can be of any type that conveys some information. This signal produced from an electronic equipment, is called as **Electronic Signal** or **Electrical Signal**. These are generally time variants.

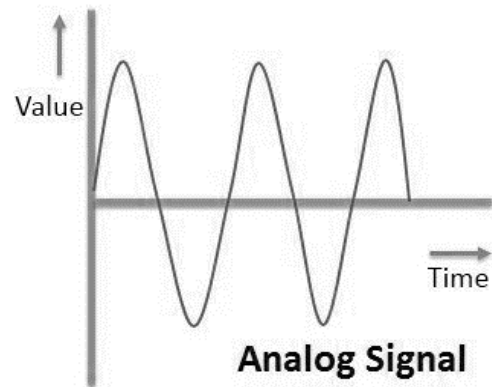
Types of Signals

Signals can be classified either as Analog or Digital, depending upon their characteristics. Analog and Digital signals can be further classified, as shown in the following image.

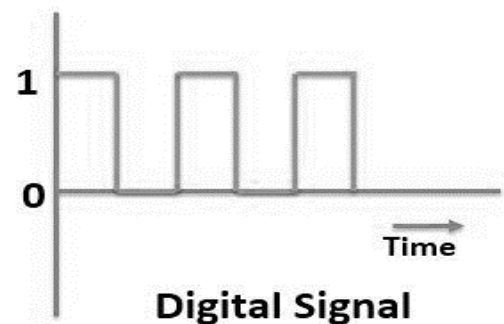


Analog Signal

A continuous time-varying signal, which represents a time-varying quantity, can be termed as an **Analog Signal**. This signal keeps on varying with respect to time, according to the instantaneous values of the quantity, which represents it.

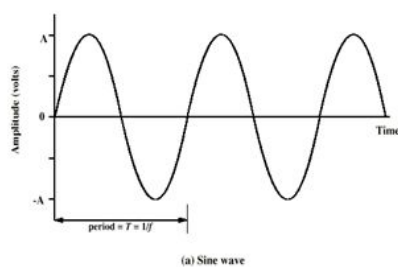
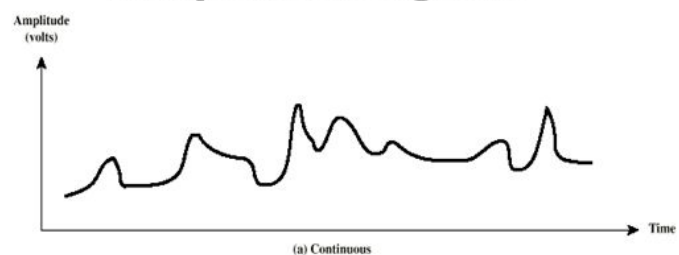
**Digital Signal**

A signal which is **discrete** in nature or which is **non-continuous** in form can be termed as a **Digital signal**. This signal has individual values, denoted separately, which are not based on previous values, as if they are derived at that instant of time.

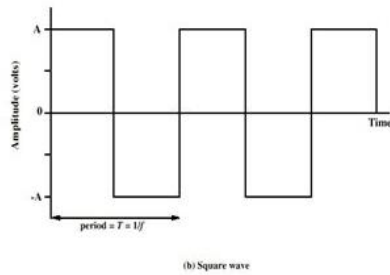
**Periodic Signal & Aperiodic (Non-periodic) Signal**

Any analog or digital signal, that repeats its pattern over a period of time, is called as a **Periodic Signal**. This signal has its pattern continued repeatedly and is easy to be assumed or to be calculated.

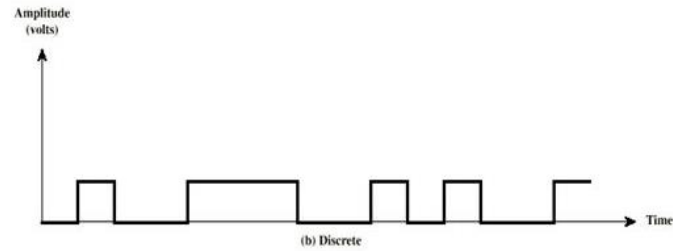
Any analog or digital signal, that doesn't repeat its pattern over a period of time, is called as **Aperiodic Signal**. This signal has its pattern continued but the pattern is not repeated and is not so easy to be assumed or to be calculated.

Periodic signals**Nonperiodic signals**

Periodic signals



Nonperiodic signals



1.2 INTRODUCTION TO NUMBER SYSTEMS

If base or radix of a number system is 'r', then the numbers present in that number system are ranging from zero to r-1. The total numbers present in that number system is 'r'. So, we will get various number systems, by choosing the values of radix as greater than or equal to two. In general, any number with decimal point can be represented by

$$\mathbf{a_{n-1} \dots a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \dots a_{-m}}$$

$$\mathbf{(r^{n-1} \times a_{n-1}) + \dots + (r^1 \times a_1) + (r^0 \times a_0) + (r^{-1} \times a_{-1}) + (r^{-2} \times a_{-2}) + (r^{-3} \times a_{-3}) + \dots + (r^{-m} \times a_{-m})}$$

Where r is the base or radix of the number system.

Where $a_{n-1}, a_2, a_1, a_0, a_{-1}, a_{-2}, a_{-3}, a_{-m}$ are called coefficients, whose values varies from 0 to r-1.

In this chapter, let us discuss about the **popular number systems** and how to represent a number in the respective number system. The following number systems are the most commonly used.

- Decimal Number system
- Binary Number system
- Octal Number system
- Hexadecimal Number system

1.2.1 Decimal Number System

The **base** or radix of Decimal number system is **10**. So, the numbers ranging from 0 to 9 are used in this number system. The part of the number that lies to the left of the **decimal point** is known as integer part. Similarly, the part of the number that lies to the right of the decimal point is known as fractional part.

In this number system, the successive positions to the left of the decimal point having weights of 10^0 , 10^1 , 10^2 , 10^3 and so on. Similarly, the successive positions to the right of the decimal point having weights of 10^{-1} , 10^{-2} , 10^{-3} and so on. That means, each position has specific weight, which is **power of base 10**

Example

Consider the **decimal number 1358.246**. Integer part of this number is 1358 and fractional part of this number is 0.246. The digits 8, 5, 3 and 1 have weights of 10^0 , 10^1 , 10^2 and 10^3 respectively. Similarly, the digits 2, 4 and 6 have weights of 10^{-1} , 10^{-2} and 10^{-3} respectively.

Mathematically, we can write it as

$$1358.246 = (1 \times 10^3) + (3 \times 10^2) + (5 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (4 \times 10^{-2}) + (6 \times 10^{-3})$$

After simplifying the right-hand side terms, we will get the decimal number, which is on left hand side.

1.2.2 Binary Number System

All digital circuits and systems use this binary number system. The **base** or radix of this number system is **2**. So, the numbers 0 and 1 are used in this number system.

The part of the number, which lies to the left of the **binary point** is known as integer part. Similarly, the part of the number, which lies to the right of the binary point is known as fractional part.

In this number system, the successive positions to the left of the binary point having weights of 2^0 , 2^1 , 2^2 , 2^3 and so on. Similarly, the successive positions to the right of the binary point having weights of 2^{-1} , 2^{-2} , 2^{-3} and so on. That means, each position has specific weight, which is **power of base 2**.

Example

Consider the **binary number 1101.011**. Integer part of this number is 1101 and fractional part of this number is 0.011. The digits 1, 0, 1 and 1 of integer part have weights of 2^0 , 2^1 , 2^2 , 2^3 respectively. Similarly, the digits 0, 1 and 1 of fractional part have weights of 2^{-1} , 2^{-2} , 2^{-3} respectively. **Mathematically**, we can write it as

$$1101.011 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of binary number on left hand side.

1.2.3 Octal Number System

The **base** or radix of octal number system is **8**. So, the numbers ranging from 0 to 7 are used in this number system. The part of the number that lies to the left of the **octal point** is known as integer part. Similarly, the part of the number that lies to the right of the octal point is known as fractional part.

In this number system, the successive positions to the left of the octal point having weights of $8^0, 8^1, 8^2, 8^3$ and so on. Similarly, the successive positions to the right of the octal point having weights of $8^{-1}, 8^{-2}, 8^{-3}$ and so on. That means, each position has specific weight, which is **power of base 8**.

Example

Consider the **octal number 1457.236**. Integer part of this number is 1457 and fractional part of this number is 0.236. The digits 7, 5, 4 and 1 have weights of $8^0, 8^1, 8^2$ and 8^3 respectively. Similarly, the digits 2, 3 and 6 have weights of $8^{-1}, 8^{-2}, 8^{-3}$ respectively.

Mathematically, we can write it as

$$1457.236 = (1 \times 8^3) + (4 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (2 \times 8^{-1}) + (3 \times 8^{-2}) + (6 \times 8^{-3})$$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of octal number on left hand side.

1.2.4 Hexadecimal Number System

The **base** or radix of Hexa-decimal number system is **16**. So, the numbers ranging from 0 to 9 and the letters from A to F are used in this number system. The decimal equivalent of Hexa-decimal digits from A to F are 10 to 15.

The part of the number, which lies to the left of the **hexadecimal point** is known as integer part. Similarly, the part of the number, which lies to the right of the Hexa-decimal point is known as fractional part.

In this number system, the successive positions to the left of the Hexa-decimal point having weights of $16^0, 16^1, 16^2, 16^3$ and so on. Similarly, the successive positions to the right of the Hexa-decimal point having weights of $16^{-1}, 16^{-2}, 16^{-3}$ and so on. That means, each position has specific weight, which is **power of base 16**.

Example

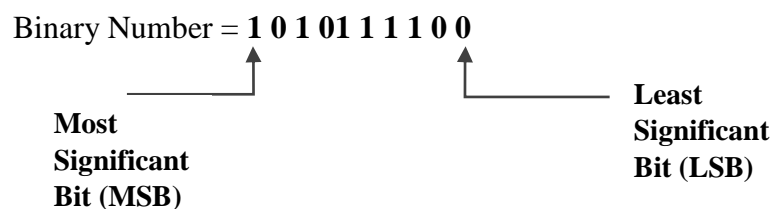
Consider the **Hexa-decimal number 1A05.2C**. Integer part of this number is 1A05 and fractional part of this number is 0.2C. The digits 5, 0, A and 1 have weights of 16^0 , 16^1 , 16^2 and 16^3 respectively. Similarly, the digits 2, C have weights of 16^{-1} , 16^{-2} .

Mathematically, we can write it as

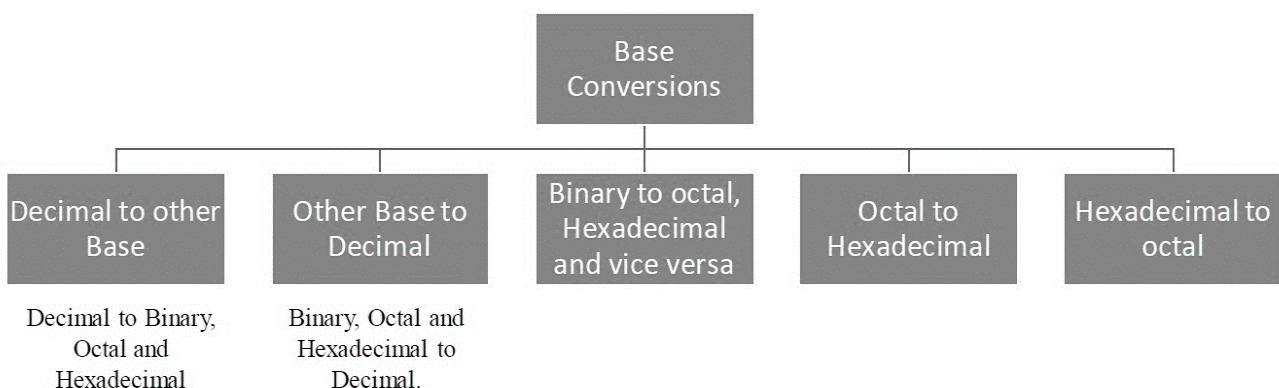
$$1A05.2C_4 = (1 \times 16^3) + (10 \times 16^2) + (0 \times 16^1) + (5 \times 16^0) + (2 \times 16^{-1}) + (12 \times 16^{-2})$$

After simplifying the right-hand side terms, we will get a decimal number, which is an equivalent of Hexa-decimal number on left hand side.

Note: In general, in any number the right-hand side number digit is known as least significant digit and left-hand side digit is known as most significant digit, for binary numbers it is known as least significant bit (LSB) and most significant bit (MSB).

**Summary:**

S. No	Number System	Base or radix (r)	Digits used
1.	Decimal Number System	r = 10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
2.	Binary Number System	r = 2	0, 1
3.	Octal Number System	r = 8	0, 1, 2, 3, 4, 5, 6, 7
4.	Hexadecimal Number System	r = 16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.3 NUMBER CONVERSION FROM ONE RADIX TO OTHER

1.3.1 Decimal to Other Base

If the decimal number contains both integer part and fractional part, then convert both the parts of decimal number into other base individually. Follow these steps for converting the decimal number into its equivalent number of any base 'r'.

1. Do division of integer part of decimal number and successive quotients with base 'r' and note down the remainders till the quotient is zero. Consider the remainders in reverse order to get the integer part of equivalent number of base 'r'. That means, first and last remainders denote the least significant digit and most significant digit respectively.
2. Do multiplication of fractional part of decimal number and successive fractions with base 'r' and note down the carry till the result is zero or the desired number of equivalent digits is obtained. Consider the normal sequence of carry in order to get the fractional part of equivalent number of base 'r'.

1.3.1(a) Decimal to Binary Conversion

The following two types of operations take place, while converting decimal number into its equivalent binary number.

1. Division of integer part and successive quotients with base 2.
2. Multiplication of fractional part and successive fractions with base 2.

Example

Consider the decimal number 58.25. Here, the integer part is 58 and fractional part is 0.25.

Step 1 – Division of 58 and successive quotients with base 2.

Operation	Quotient	Remainder
58/2	29	0 <i>LSB</i>
29/2	14	1
14/2	7	0
7/2	3	1
3/2	1	1
1/2	0	1 <i>MSB</i>

$$(58)_{10} = (111010)_2$$

Therefore, the integer part of equivalent binary number is 111010.

Step 2 – Multiplication of 0.25 and successive fractions with base 2.

Operation	Result	Carry
0.25 x 2	0.5	0
0.5 x 2	1.0	1
-	0.0	-

$$(0.25)_{10} = (0.01)_2$$

Therefore, the fractional part of equivalent binary number is 0.01

Therefore, the binary equivalent of decimal number $(58.25)_{10}$ is $(111010.01)_2$

1.3.1(b) Decimal to Octal Conversion

The following two types of operations take place, while converting decimal number into its equivalent octal number.

1. Division of integer part and successive quotients with base 8.
2. Multiplication of fractional part and successive fractions with base 8.

Example

Consider the decimal number 58.25. Here, the integer part is 58 and fractional part is 0.25.

Step 1 – Division of 58 and successive quotients with base 8.

Operation	Quotient	Remainder
58/8	7	2
7/8	0	7

$$(58)_{10} = (72)_8$$

Therefore, the integer part of equivalent octal number is 72.

Step 2 – Multiplication of 0.25 and successive fractions with base 8.

Operation	Result	Carry
0.25 x 8	2.00	2
-	0.00	-

$$(0.25)_{10} = (0.2)_8$$

Therefore, the fractional part of equivalent octal number is .2

Therefore, the octal equivalent of decimal number $(58.25)_{10}$ is $(72.2)_8$

1.3.1(c) Decimal to Hexadecimal Conversion

The following two types of operations take place, while converting decimal number into its equivalent hexa-decimal number.

1. Division of integer part and successive quotients with base 16.
2. Multiplication of fractional part and successive fractions with base 16.

Example

Consider the decimal number 58.25. Here, the integer part is 58 and decimal part is 0.25.

Step 1 – Division of 58 and successive quotients with base 16.

Operation	Quotient	Remainder
58/16	3	10=A
3/16	0	3

Therefore, the integer part $(58)_{10}$ of equivalent Hexa-decimal number is $(3A)_{16}$

Step 2 – Multiplication of 0.25 and successive fractions with base 16.

Operation	Result	Carry
0.25 x 16	4.00	4
-	0.00	-

Therefore, the fractional part $(0.25)_{10}$ of equivalent Hexa-decimal number is $(0.4)_{16}$

Therefore, the Hexa-decimal equivalent of decimal number $(58.25)_{10}$ is $(3A.4)_{16}$

1.3.2 Other Base to Decimal

Step 1 – Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 – Multiply the obtained column values (in step 1) by the digits in the corresponding columns.

Step 3 – Sum the products calculated in step 2. The total is the decimal equivalent of the given number.

Example-1Binary Number – 11101_2

Calculating Decimal Equivalent –

$$\begin{array}{cccccc} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \\ 1 & 1 & 1 & 0 & 1 & \end{array}$$

Step	Binary Number	Decimal Number
Step 1	$(11101)_2$	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	$(11101)_2$	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	$(11101)_2$	$(29)_{10}$

The decimal equivalent of $(11101)_2$ is = $(29)_{10}$

Example-2Binary Number – $(11101.011)_2$

Calculating Decimal Equivalent –

$$\begin{array}{cccccccc} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\ 1 & 1 & 1 & 0 & 1 & . & 0 & 1 & 1 \end{array}$$

Step	Binary Number	Decimal Number
Step 1	$(11101.011)_2$	$(1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$
Step 2	$(11101.011)_2$	$(16 + 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125)$
Step 3	$(11101.011)_2$	$(29.375)_{10}$

The decimal equivalent of $(11101)_2$ is = $(29.375)_{10}$

Example-3Octal Number – $(237.34)_8$

Calculating Decimal Equivalent –

$$\begin{array}{cccccc} 8^2 & 8^1 & 8^0 & 8^{-1} & 8^{-2} & \\ 2 & 3 & 7 & . & 3 & 4 \end{array}$$

Step	Octal Number	Decimal Number
Step 1	$(237.34)_8$	$(2 \times 8^2) + (3 \times 8^1) + (7 \times 8^0) + (3 \times 8^{-1}) + (4 \times 8^{-2})$
Step 2	$(237.34)_8$	$(128 + 24 + 7 + 0.375 + 0.0625)$
Step 3	$(237.34)_8$	$(159.4375)_{10}$

The decimal equivalent of $(237.43)_8$ is = $(159.4375)_{10}$

Example-4Hexadecimal Number – $(2AC.FE)_{16}$

Calculating Decimal Equivalent –

$$\begin{array}{cccccc} 16^2 & 16^1 & 16^0 & 16^{-1} & 16^{-2} & \\ 2 & A & C & .F & E & \end{array}$$

Step	Hexa Number	Decimal Number
Step 1	$(237.34)_8$	$(2 \times 16^2) + (A \times 16^1) + (C \times 16^0) + (F \times 16^{-1}) + (E \times 16^{-2})$ $(2 \times 16^2) + (10 \times 16^1) + (12 \times 16^0) + (15 \times 16^{-1}) + (14 \times 16^{-2})$
Step 2		$(512 + 160 + 12 + 0.9375 + 0.0546875)$
Step 3		$(684.9921875)_{10}$

The decimal equivalent of $(237.43)_8$ is $= (684.9921875)_{10}$

1.3.3 Binary to Octal and Hexadecimal Vice-versa**1.3.3 (a) Binary to Octal Number Conversion****For the integer part:**–Scan the binary number from *right to left*.

–Translate each group of three bits into the corresponding octal digit.

Add *leading* zeros if necessary.**For the fractional part:**–Scan the binary number from *left to right*.

–Translate each group of three bits into the corresponding octal digit.

Add *trailing* zeros if necessary.

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

Example-1Consider the **binary number 101110.01101****Step 1** – Make the groups of 3 bits on both sides of binary point.

$$101|110.011|01$$

Here, on right side of binary point, the last group is having only 2 bits. So, include one zero on extreme side in order to make it as group of 3 bits.

$$101|110.011|010$$

Step 2 – Write the octal digits corresponding to each group of 3 bits.

$$101|110.011|010 = 56.32$$

Therefore, the **octal equivalent** of binary number 101110.01101 is $(56.32)_8$

1.3.3 (b) Binary to Hexadecimal Number Conversion

For the integer part:

–Scan the binary number from *right to left*.

–Translate each group of four bits into the corresponding Hexadecimal digit.

Add *leading zeros* if necessary.

For the fractional part:

–Scan the binary number from *left to right*.

–Translate each group of four bits into the corresponding hexadecimal digit.

Add *trailing zeros* if necessary.

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

Example-2

Consider the **binary number 10111110.011011101**

Step 1 – Make the groups of 4 bits on both sides of binary point.

$$1011|1110.0110|1110|1$$

Here, on right side of binary point, the last group is having only 1 bits. So, include three zeros on extreme side in order to make it as group of 4 bits.

$$1011|1110.0110|1110|1000$$

Step 2 – Write the octal digits corresponding to each group of 3 bits.

$$10111110.011011101000 = BE.6E8$$

Therefore, the hexadecimal **equivalent** of binary number **10111110.011011101** is $(BE.6E8)_{16}$

1.3.3 (c) Octal, Hexadecimal to Binary Conversion

1. To convert an octal number into binary, write the 3-bit binary equivalent of each octal digit.
2. To convert a hexadecimal number into binary, write the 4-bit binary equivalent of each hexadecimal digit.

Example-1: Convert the octal number $(1342.56)_8$ to binary?

Given octal number = $(1\ 3\ 4\ 2\ .\ 5\ 6)_8$

Now write the 3-bit binary equivalent of each octal digit.

The binary equivalent is $= (001\ 011\ 100\ 010.\ 101\ 110)_2$

Example-2: Convert the Hexadecimal number $(1A3B.F4)_{16}$ to binary?

Given Hexadecimal number = $(1\ A\ 3\ B.\ F\ 4)_{16}$

Now write the 4-bit binary equivalent of each hexadecimal digit.

The binary equivalent is $(0001\ 1010\ 0011\ 1011.\ 1111\ 0100)_2$

1.3.4 Octal to Hexadecimal and vice versa**(i) Octal to Hexa-Decimal Conversion**

Follow these two steps for converting an octal number into its equivalent Hexa-decimal number.

1. Convert octal number into its equivalent binary number.
2. Convert the above binary number into its equivalent Hexa-decimal number.

(ii) Hexa-Decimal to Octal Conversion

Follow these two steps for converting Hexa-decimal number into its equivalent octal number.

1. Convert Hexa-decimal number into its equivalent binary number.
2. Convert the above binary number into its equivalent octal number.

Example-1: Convert $(7652.123)_8$ to Hexadecimal Number?

Step-1: First convert the octal number into binary by writing the 3-bit binary equivalent of each octal digit.

$(7652.123)_8 = (111\ 110\ 101\ 010.\ 001\ 010\ 011)_2$

Step-2: Convert the binary number obtained in step-1 into hexadecimal by partitioning the number into 4-bit group and writing the hexadecimal equivalent of each 4-bit binary.

$(111110101010.001010011)_2 = (1111\ 1010\ 1010.0010\ 1001\ 1000)_2 = (F\ A\ A.\ 2\ 9\ 8)_{16}$

Example-2: Convert $(FAA.298)_{16}$ to Octal Number?

Step-1: First convert the Hexadecimal number into binary by writing the 4-bit binary equivalent of each hexadecimal digit.

$$(FAA.298)_{16} = (1111\ 1010\ 1010.\ 0010\ 1001\ 1000)_2$$

Step-2: Convert the binary number obtained in step-1 into Octal by partitioning the number into 3-bit group and writing the octal equivalent of each 3-bit binary.

$$(111110101010.001010011000)_2 = (111\ 110\ 101\ 010.001\ 010\ 011\ 000)_2 = (7652.1230)_8$$

1.4 COMPLEMENTS OF A NUMBER SYSTEM

Complements are used in digital computers for

1. To represent negative numbers.
2. Simplifying the subtraction operation.

For example, to perform $A - B$

We can write $A + (-B)$

$A + (\text{Complement of } B)$

Here by performing addition operation between A and complement of B , we can obtain subtraction result.

There are two types of complements for each base- r number system.

1. Diminished radix complement ($(r-1)$'s complements)
2. Radix complement (r 's complements)

S.No	Number System	$(r-1)$'s Complements	r 's Complements
1.	Decimal Numbers	9's complements	10's complements
2.	Binary Numbers	1's complements	2's complements
3.	Octal Numbers	7's complements	8's complements
4.	Hexadecimal Numbers	15's complements	16's complements

1.4.1 r 's and $(r-1)$'s complements

The r 's and $(r-1)$'s complements are generalized representation of the complements; we have studied in previous subsections. r stands for radix or base of the number system, thus r 's complement is referred as radix complement and $(r-1)$'s complement is referred as diminished radix complement. Examples of r 's complements are 2's complement and 10's complement. Examples of $(r-1)$'s complement are 1's complement and 9's complement.

In a base-r system, the r's and (r - 1)'s complement of the number N having n digits, can be defined as:

$$(r - 1)'s \text{ complement of } N = (r^n - 1) - N$$

and

$$\begin{aligned} r's \text{ complement of } N &= r^n - N \\ &= (r - 1)'s \text{ complement of } N + 1 \end{aligned}$$

The (r - 1)'s complement can also be obtained by subtracting each digit of N from r-1. Using the above methodology, we can also define the 7's and 8's complement for octal system and 15's and 16's complement for hexadecimal system.

1.4.2 1's and 2's complements

These are the complements used for binary numbers. Their representation are very important as digital systems work on binary numbers only.

1's Complement

1's complement of a binary number is obtained simply by replacing each 1 by 0 and each 0 by 1. Alternately, 1's complement of a binary can be obtained by subtracting each bit from 1.

Example 1. Find 1's complement of (i) 011001 (ii) 00100111

Solution. (i) Replace each 1 by 0 and each 0 by 1

$$\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

So, 1's complement of 011001 is 100110.

(ii) Subtract each binary bit from 1.

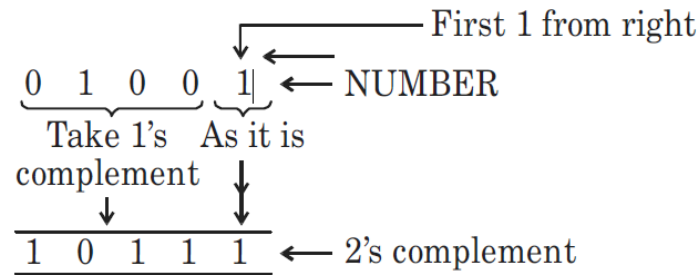
$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{array} \leftarrow 1's \text{ complement}$$

one can see that both the method gives same result.

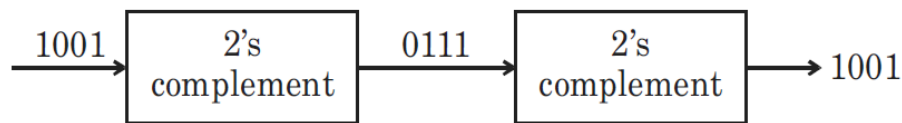
2's Complement

2's complement of a binary number can be obtained by adding 1 to its 1's complement.

(iii) Number = 01001



It is interesting to note that taking complement twice leaves the number as it is. This is illustrated in below Fig.



To represent a negative number using complements the process involves two steps.

(1) obtain the binary representation of equivalent positive number for given negative number. e.g., if given number is -2 then obtain binary representation of $+2$.

(2) Take the appropriate complement of representation obtained in step 1.

Example 3. Obtain 1's and 2's complement representation of -5 and -7 .

Solution. (i) -5

1. binary of $+5 = (0101)_2$
2. 1's complement of $(0101)_2 = (1010)_2 \leftarrow$ Represents $(-5)_{10}$
 2's complement of $(0101)_2 = (1011)_2 \leftarrow$ Represents $(-5)_{10}$

(ii) -7

1. binary of $+7 = (0111)_2$
2. 1's complement of $(0111)_2 = (1000)_2$ Represents $(-7)_{10}$
 2's complement of $(0111)_2 = (1001)_2$ Represents $(-7)_{10}$

Note that in above two examples, for positive numbers we obtained such a binary representation in which MSB is 0. e.g., for $+7$ we obtained $(0111)_2$ not just $(111)_2$. It is because for all positive numbers MSB must be 0 and for negative numbers MSB should be 1.

1.4.3 9's and 10's complements

9's and 10's complements are the methods used for the representation of decimal numbers. They are identical to the 1's and 2's complements used for binary numbers.

9's complement: 9's complement of a decimal number is defined as $(10^n - 1) - N$, where n is no. of digits and N is given decimal numbers. Alternately, 9's complement of a decimal number can be obtained by subtracting each digit from 9.

$$\boxed{\text{9's complement of } N = (10^n - 1) - N.}$$

Example 1. Find out the 9's complement of following decimal numbers.

(i) 459 (ii) 36 (iii) 1697

Solution. (i) By using $(10^n - 1) - N$; But, $n = 3$ in this case

$$\text{So, } (10^n - 1) - N = (10^3 - 1) - 459 = 540$$

$$\text{Thus, 9's complement of 459} = 540$$

(ii) By subtracting each digit from 9

$$\begin{array}{r} 9 \ 9 \\ -3 \ 6 \\ \hline 6 \ 3 \end{array}$$

So, 9's complement of 36 is 63.

(iii) We have $N = 1697$, so $n = 4$

$$\text{Thus, } 10^n - 1 = 10^4 - 1 = 9999$$

$$\begin{aligned} \text{So, } (10^n - 1) - N &= (10^4 - 1) - 1697 = 9999 - 1697 \\ &= 8302 \end{aligned}$$

$$\text{Thus, 9's complement of 1697} = 8302$$

10's complement: 10's complement of a decimal number is defined as $10^n - N$.

$$\boxed{\text{10's complement of } N = 10^n - N}$$

$$\begin{aligned} \text{but } 10^n - N &= (10^n - 1) - N + 1 \\ &= \text{9's complement of } N + 1 \end{aligned}$$

Thus, 10's complement of a decimal number can also be obtained by adding 1 to its 9's complement.

Example 2. Find out the 10's complement of following decimal numbers. (i) 459 (ii) 36.

Solution. (i) By using $10^n - N$; We have $N = 459$ so $n = 3$

$$\text{So, } 10^n - N = 10^3 - 459 = 541$$

So, 10's is complement of 459 = 541

(ii) By adding 1 to 9's complement

$$\begin{aligned} \text{9's complement of 36} &= 99 - 36 \\ &= 63 \end{aligned}$$

$$\begin{aligned} \text{Hence, 10's complement of 36} &= 63 + 1 \\ &= 64 \end{aligned}$$

1.5 BINARY ARITHMETIC

The binary arithmetic operations such as addition, subtraction, multiplication and division are similar to the decimal number system. Binary arithmetic's are simpler than decimal because they involve only two digits (bits) 1 and 0.

Binary Addition:

Rules for binary addition are summarized in the table shown in Fig. 1

<i>Augend</i>	<i>Addend</i>	<i>Sum</i>	<i>Carry</i>	<i>Result</i>
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Fig. 1 Rules for binary addition

As shown in 4th row adding 1 to 1 gives 1 carry which, is given to next binary position, similar to decimal system. This is explained in examples below:

Example 1. (i) Add 1010 and 0011 (ii) Add 0101 and 1111

Solution.

$$\begin{array}{r}
 \begin{array}{r}
 \leftarrow \text{Carry} \\
 1\ 0\ 1\ 0 \\
 +\ 0\ 0\ 1\ 1 \\
 \hline
 1\ 1\ 0\ 1
 \end{array}
 \qquad
 \begin{array}{r}
 1\ 1\ 1 \leftarrow \text{Carry} \\
 0\ 1\ 0\ 1 \\
 +\ 1\ 1\ 1\ 1 \\
 \hline
 1\ 0\ 1\ 0\ 0 \\
 \uparrow \\
 \text{Carry}
 \end{array}
 \end{array}$$

Binary Subtraction:

The rules for binary subtraction are summarized in the table shown in Fig. 2.

<i>Minuend</i>	<i>Subtrahend</i>	<i>Difference</i>	<i>Borrow</i>
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Fig. 2 Rules for binary subtraction

The process of subtraction is very similar to decimal system in which if a borrow is needed it is taken from next higher binary position, as shown in row 2.

Example 2. Subtract 0100 from 1011

Solution.

$$\begin{array}{r}
 \leftarrow \text{Borrow} \\
 1\ 0\ 1\ 1 \leftarrow \text{Minuend} \\
 -\ 0\ 1\ 0\ 0 \leftarrow \text{Subtrahend} \\
 \hline
 0\ 1\ 1\ 1 \leftarrow \text{Difference} \\
 \hline
 \uparrow\ \uparrow\ \uparrow\ \uparrow \\
 C_3\ C_2\ C_1\ C_0
 \end{array}$$

There is no problem in column C0 and C1. In column C2 we made 0 – 1, so result = 1 and borrow = 1. Then this borrow = 1 is marked in column C3. So, result in column C2 is 1. Then in column C3 we first made 1 – 0 to get result = 1 and then we subtracted borrow from result, thus, we get 0 in column C3.

“Thus, in subtraction, first subtract the subtrahend bit from minuend and then subtract borrow from the result.”

Example 3. Subtract 0110 from 1001

Solution.	1	1	0	1	← Borrow
	1	0	0	1	← Minuend
	– 0	1	1	0	← Subtrahend
	0	0	1	1	← Difference
	↑	↑	↑	↑	
	C ₃	C ₂	C ₁	C ₀	

Here, in column C₁ we get difference = 1 and borrow = 1. This borrow is marked in column C₂, and difference = 1 is shown in the column C₁. We now come to column C₂. Here by 0–1 we get difference = 1 and borrow = 1. Now this borrow is marked in column C₃. But in column C₂ already we have 9 borrow so this borrow = 1 is subtracted from difference = 1 which results in 0. Thus the difference = 0 is marked in column C₂.

In the similar way we process column C₃ and we get difference = 0 in column C₃.

Binary Multiplication:

Binary multiplication is also similar to decimal multiplication. In binary multiplication if multiplier bit is 0 then partial product is all 0 and if multiplier bit is 1 then partial product is 1.

Input A	Input B	Multiply (M) AxB
0	0	0
0	1	0
1	0	0
1	1	1

The same is illustrated in example below:

Example 4.

1 0 0 1	←	MULTIPLICAND
1 0 1	←	MULTIPLIER
1 0 0 1	←	Partial Product when multiplier bit = 1
0 0 0 0	←	Partial Product when multiplier bit = 0
1 0 0 1	←	Partial Product when multiplier bit = 0
1 0 1 1 0 1	←	FINAL PRODUCT

(ii) $3 - 7$:

binary of 3 = 0011

binary of 7 = 0111

Step 1. 1's complement of 0111 = 1000

Step 2. Perform addition of minuend and 1's complement of subtrahend

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ \leftarrow (3) \\ 1 \ 0 \ 0 \ 0 \ \leftarrow (-7 \text{ or } 1\text{'s complement of } +7) \\ \hline \text{Result} = \underline{1 \ 0 \ 1 \ 1} \end{array}$$

Step 3. No carry produced so no EAC operation.

Step 4. Since no carry produced in step 2, result is negative and is in complemented form. So we must take 1's complement of result to find correct magnitude of result.

1's complement of result $(1011)_2 = (0100)_2$

so, final result = $-(0100)_2$ or $-(4)_{10}$

Note that when (in example (ii)) the result was negative (step 2), MSB of the result was 1. When (in example (i)) the result was positive the MSB was 0. The same can be observed in 2's complement subtraction.

2's complement Subtraction Method of 2's complement is similar to 1's complement subtraction except the end around carry (EAC) is not needed. The rules are listed below:

1. Take 2's complement of subtrahend.
2. Add 2's complement of subtrahend to minuend.
3. If a carry is produced, then discard the carry and the result is positive. If no carry is produced result is negative and is in 2's complement form.
4. To obtain the result in familiar form find the 2's complement of the sum and add a minus sign in front of it.

Example 2. Perform following subtraction using 2's complement.

(i) $7 - 5$

(ii) $5 - 7$

Solution. (i) $7 - 5$: binary of 7 = $(0111)_2$ binary of 5 = $(0101)_2$ both the numbers should have equal length.

Step 1. 2's complement of subtrahend $(=0101)_2 = (1011)_2$.

Step 2. Perform addition of minuend and 2's complement of subtrahend.

$$\begin{array}{r} 0 \ 1 \ 1 \ 1 \ \leftarrow (7) \\ + \ 1 \ 0 \ 1 \ 1 \ \leftarrow (-5 \text{ or } 2\text{'s complement of } +5) \\ \hline \text{Final Carry} \rightarrow \underline{1 \ 0 \ 0 \ 1 \ 0} \\ \uparrow \\ \text{Discard the final carry} \end{array}$$

Step 3. Since a final carry is produced in step 2 (which is discarded) the result is positive. So,

$$\text{result} = (0010)_2 = (2)_{10}$$

(ii) $5 - 7$:

binary of 5 = $(0101)_2$

binary of 7 = $(0111)_2$

Step 1. 2's complement of subtrahend (= 0111) = 1001.

Step 2. Addition of minuend and 2's complement of subtrahend.

$$\begin{array}{r}
 0\ 1\ 0\ 1 \leftarrow 5 \\
 1\ 0\ 0\ 1 \leftarrow (-7 \text{ or } 2\text{'s complement of } +7) \\
 \hline
 1\ 1\ 1\ 0 \leftarrow \text{Result} \\
 \uparrow \\
 \text{No final carry}
 \end{array}$$

Step 3. Since final carry is not generated in step 2, the result is negative and is in 2's complement form. So we must take 2's complement of result obtained in step 2 to find correct magnitude of result.

2's complement of result $(1110)_2 = (0010)_2$

so, final result = $-(0010)_2 = -(2)_{10}$

1.7 BINARY CODES

The digital data is represented, stored and transmitted as group of bits. This group of bits is also called as binary code.

Binary codes can be classified into two types.

1. Weighted codes
2. Unweighted codes

If the code has positional weights, then it is said to be weighted code. Otherwise, it is an unweighted code. Weighted codes can be further classified as positively weighted codes and negatively weighted codes.

Binary Coded Decimal (BCD) Code:

It is a weighted binary code, which obey the positional weight principle. Each position of the number represents a specific weight. The weights associated with each digit are 8, 4, 2 and 1 from left to right side.

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

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

To represent a multi-digit decimal number in BCD, each decimal digit is represented by its 4-bit binary equivalent.

Ex: $(9)_{10} = (1001)_{\text{BCD}}$
 $(23)_{10} = (0010\ 0011)_{\text{BCD}}$
 $(679)_{10} = (0110\ 0111\ 1001)_{\text{BCD}}$

Advantages of BCD Codes

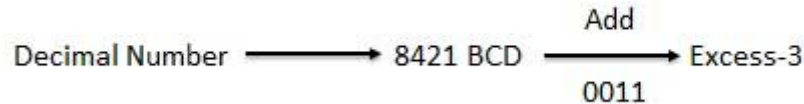
1. It is very similar to decimal system.
2. We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

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

Excess – 3 Code:

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



The following table shows the excess-3 codes for decimal digits 0-9.

Decimal	BCD	Excess-3
	8 4 2 1	BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

To represent a multi-digit decimal number in Excess-3, each decimal digit is first converted into BCD and then by adding $(0011)_2$ each BCD digit it is converted into excess-3 code.

Ex: $(9)_{10} = (1001)_{\text{BCD}} = (1100)_{\text{XS-3}}$
 $(23)_{10} = (0010\ 0011)_{\text{BCD}} = (0101\ 0110)_{\text{XS-3}}$
 $(679)_{10} = (0110\ 0111\ 1001)_{\text{BCD}} = (1001\ 1010\ 1100)_{\text{XS-3}}$

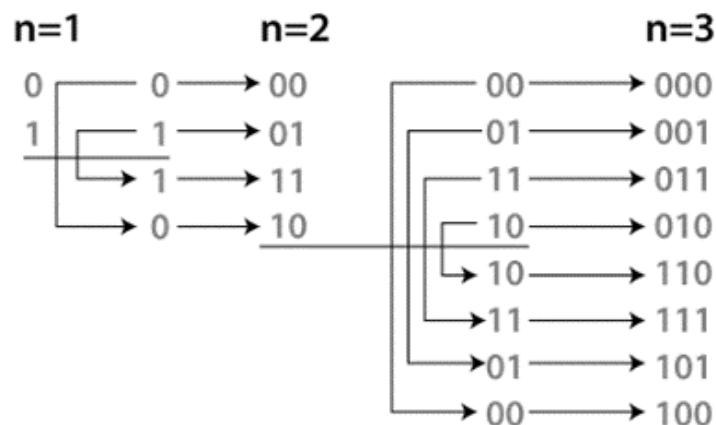
Gray Code:

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

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

Cyclic property:

n-bit Gray Code



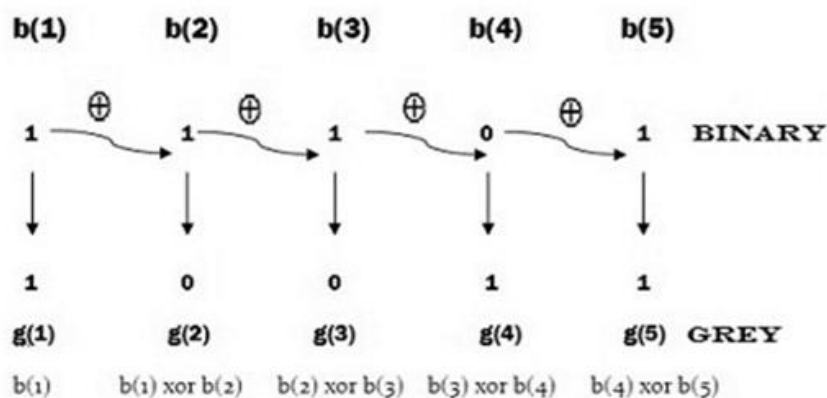
1.7.1 Binary to Grey code conversion:

This conversion method strongly follows the EX-OR gate operation between binary bits. The below steps & solved example may useful to know how to perform binary to gray code conversion.

1. To convert binary to grey code, bring down the most significant digit of the given binary number, because the first digit or most significant digit of the grey code number is same as the binary number.
2. To obtain the successive grey coded bits to produce the equivalent grey coded number for the given binary, perform ex-or operation between MSB and next binary digit. Continue the same up to LSB number.

Binary to Grey Code Conversion

Convert the binary 11101_2 to its equivalent Grey code

**1.7.2 Grey to Binary code conversion:**

This conversion method also follows the EX-OR gate operation between gray & binary bits. The below steps & solved example may useful to know how to perform gray code to binary conversion.

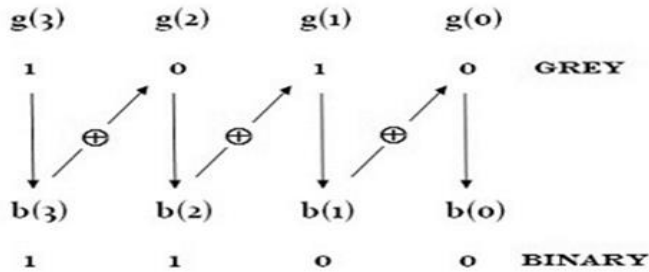
1. To convert gray code to binary, bring down the most significant digit of the given gray code number, because, the first digit or the most significant digit of the gray code number is same as the binary number.
2. To obtain the successive second binary bit, perform the EX-OR operation between the first bit or most significant digit of binary to the second bit of the given gray code.
3. To obtain the successive third binary bit, perform the EX-OR operation between the second bit of binary to the third MSD (most significant digit) of gray code and so on for the next successive binary bits conversion to find the equivalent.

Exclusive-OR– If the two bits EX-ORED are identical, the result is 0; if the two bits differ, the result is 1.

Exclusive-OR gate

Grey Code to Binary Conversion

Convert the Grey code 1010 to its equivalent Binary



i.e

$$b(3) = g(3)$$

$$b(2) = b(3) \oplus g(2)$$

$$b(1) = b(2) \oplus g(1)$$

$$b(0) = b(1) \oplus g(0)$$



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

UNIT – 2

BOOLEAN ALGEBRA AND THEOREMS

2.1 BINARY LOGIC

Binary logic deals with variables that take on two discrete values. The two values the variables assume may be called by different names (true and false, yes and no, etc.), but for our purpose, it is convenient to think in terms of bits and assign the values 1 and 0.

Here 0 and 1 are symbols that represent a range of voltages. A logic 0 represents anything between 0V to 1V, logic 1 represents a voltage range from 2V to 3V. The range of voltages may vary based on the logic family used to implement logic gates. The range of voltages for logic 0 and logic 1 are shown in below fig. 2.1.

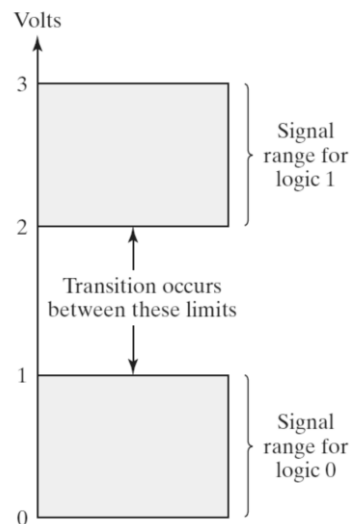


Fig. 2.1 Signal levels for binary logic values

Binary logic consists of binary variables and a set of logical operations. The variables are designated by letters of the alphabet, such as A, B, C, x, y, z, etc., with each variable having two and only two distinct possible values: 1 and 0. There are three basic logical operations: AND, OR, and NOT. Each operation produces a binary result, denoted by z.

1. AND: This operation is represented by a dot or by the absence of an operator. For example, $x \cdot y = z$ or $xy = z$ is read “x AND y is equal to z.” **The logical operation AND is interpreted to mean that $z = 1$ if and only if $x = 1$ and $y = 1$; otherwise, $z = 0$.** (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.) The result of the operation $x \cdot y$ is z.

2. OR: This operation is represented by a plus sign. For example, $x + y = z$ is read “x OR y is equal to z,” meaning that $z = 1$ if $x = 1$ or if $y = 1$ or if both $x = 1$ and $y = 1$. If both $x = 0$ and $y = 0$, then $z = 0$.

3. NOT: This operation is represented by a prime (sometimes by an overbar). For example, $x' = z$ (or $x = \bar{z}$) is read “not x is equal to z,” meaning that z is what x is not. In other words, if $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$. The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1, i.e., the result of complementing 1 is 0, and vice versa.

Truth Tables of Logical Operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

2.2 AXIOMATIC DEFINITION OF BOOLEAN ALGEBRA

Boolean algebra is an algebraic structure defined by a set of elements, $B = \{0, 1\}$ together with two binary operators, $+$ and \cdot , provided that the following (Huntington) postulates are satisfied:

1. (a) The structure is closed with respect to the operator $+$
- (b) The structure is closed with respect to the operator \cdot .

For all $a, b \in B$ such that $a + b \in B$, $a \cdot b \in B$.

2. (a) The element 0 is an identity element with respect to $+$; that is, $a + 0 = 0 + a = a$.
- (b) The element 1 is an identity element with respect to \cdot ; that is, $a \cdot 1 = 1 \cdot a = a$.

3. (a) The structure is commutative with respect to $+$; that is, $a + b = b + a$.
- (b) The structure is commutative with respect to \cdot ; that is, $a \cdot b = b \cdot a$.

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

4. (a) The operator \cdot is distributive over $+$; that is, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

(b) The operator $+$ is distributive over \cdot ; that is, $a + (b \cdot c) = (a + b) \cdot (a + c)$.

a	b	c	b+c	a.(b+c)	a.b	a.c	(a.b) + (a.c)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

a	b	c	b.c	a + (b.c)	a+b	a+c	(a+b).(a+c)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

5. For every element $a \in B$, there exists an element $a' \in B$ (called the complement of A) such that (a) $a + a' = 1$ and (b) $a.a' = 0$.

a	a'	a+a'
0	1	1
1	0	1

a	a'	a.a'
0	1	0
1	0	0

6. There exist at least two elements $a, b \in B$ such that $a \neq b$.

Postulate 6 is satisfied because the two-valued Boolean algebra has two elements, 1 and 0, with $1 \neq 0$.

2.3 BASIC THEOREMS AND PROPERTIES OF BOOLEAN ALGEBRA**DUALITY:**

The important property of Boolean algebra is called the duality principle and states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.

If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

1. $(A+B)+(CD)$ is the dual of $AB.(C+D)$
2. $A + 1 = 1$ is the dual of $A \cdot 0 = 0$
3. $A + A.B = A$ is the dual of $A \cdot (A + B) = A$
4. $A + A'.B = A + B$ is the dual of $A \cdot (A' + B) = A \cdot B$

Theorems of Boolean Algebra

Theorem 1	(a) $A + A = A$	(b) $A \cdot A = A$
Theorem 2	(a) $A + 1 = 1$	(b) $A \cdot 0 = 0$
Theorem 3: Involution	$(A')' = A$	
Theorem 4: Associative	(a) $A + (B + C) = (A + B) + C$	(b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Theorem 5: DeMorgan	(a) $(A + B)' = A' \cdot B'$	(b) $(AB)' = A' + B'$
Theorem 6: Absorption	(a) $A + AB = A$	(b) $A \cdot (A + B) = A$
Theorem 7	(a) $A + A'B = A + B$	(b) $A \cdot (A' + B) = A \cdot B$

Theorem 1(a): $A + A = A$

Proof: $A + A = (A+A).1$ By postulate 2(b): $A.1 = 1.A = A$
 $= (A+A). (A+A')$ 5(a): $A+A' = 1$
 $= A+AA'$ 4(b): $A+BC = (A+B). (A+C)$
 $= A+0$ 5(b): $A.A' = 0$
 $= A$ 2(a): $A+0 = 0+A = A$

Theorem 1(b): $A \cdot A = A$

Proof: $A \cdot A = (A \cdot A) + 0$ By postulate 2(a): $A+0 = 0+A = A$
 $= (A \cdot A) + (A \cdot A')$ 5(b): $A \cdot A' = 0$
 $= A \cdot (A+A')$ 4(a): $A \cdot (B+C) = (A \cdot B) + (A \cdot C)$
 $= A \cdot 1$ 5(a): $A+A' = 1$
 $= A$ 2(b): $A \cdot 1 = 1 \cdot A = A$

Note that theorem 1(b) is the dual of 1(a) and that each step of the proof in part (b) is the dual of part (a).

Theorem 2(a): $A + 1 = 1$

Proof: $A + 1 = 1 \cdot (A+1)$ By postulate 2(b): $A \cdot 1 = 1 \cdot A = A$
 $= (A+A') \cdot (A+1)$ 5(a): $A+A' = 1$
 $= A+A' \cdot 1$ 4(b): $A+BC = (A+B) \cdot (A+C)$
 $= A+A'$ 5(a): $A + A' = 1$
 $= 1$

Theorem 2(b): $A \cdot 0 = 0$. By taking the dual of 2(a): $A + 1 = 1$, $A \cdot 0 = 0$

Theorem 3: $(A')' = A$

The above theorem can be proved by using truth table method.

A	A'	(A')'
0	1	0
1	0	1

Theorem 4: (a) $A + (B + C) = (A + B) + C$ (b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

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

A	B	C	(B.C)	A.(B.C)	(A.B)	(A.B).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

Theorem 5: (a) $(A + B)' = A' \cdot B'$ (b) $(AB)' = A' + B'$

A	B	A'	B'	(A+B)	(A+B)'	A'.B'
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

A	B	A'	B'	(A.B)	(A.B)'	A'+B'
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

Theorem 6(a): $A+AB = A$

Proof: $A + AB = A.1 + AB$
 $= A \cdot (1+B)$
 $= A \cdot 1$
 $= A$

By postulate 2(b): $A.1 = 1.A = A$
 4(a): $A.(B+C) = AB + AC$
 By theorem 2(a): $A+1 = 1$
 2(b): $A.1 = 1.A = A$

Theorem 6(b): $A.(A+B) = A$, by taking the dual of 6(a): $A+AB = A$, $A.(A+B) = A$ **Theorem 7(a):** $A + A'B = A + B$

Proof: $A + A'B = A+AB + A'B$
 $= A + (B.(A+A'))$
 $= A+B.1$
 $= A+B$

By Theorem 6(a): $A+AB = A$
 By postulate 4(a): $A.(B+C) = AB+AC$
 5(a): $A+A' = 1$
 2(b): $A.1 = 1.A = A$

Theorem 7(b): $A.(A'+B) = AB$, by taking the dual of theorem 7(a); $A+A'B = A+B$,
 $A.(A'+B) = AB$

2.4 LOGIC GATES

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

2.4.1 Basic Gates:

There are three basic gates, namely AND, OR and NOT gates.

AND gate:

A logical AND gate produces a high output when all of its inputs are at high, otherwise its output is low. It is optional to represent the **Logical AND** with the symbol '·'.

The following shows the truth table and logic symbol of 2-input AND gate.

Boolean Expression	AND Symbol ↑ $A \cdot B = Y$															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Y														
0	0	0														
0	1	0														
1	0	0														
1	1	1														

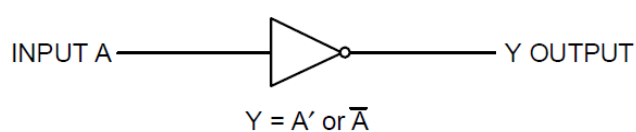
OR gate:

An OR gate output is high when at least one of its input or all of its inputs are at high, otherwise its output is low. This **logical OR** is represented with the symbol '+'. The following shows the truth table and logic symbol of 2-input OR gate.

Boolean Expression	OR Symbol ↑ $A + B = Y$															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1
A	B	Y														
0	0	0														
0	1	1														
1	0	1														
1	1	1														

NOT gate:

A NOT gate is a digital circuit that has single input and single output. The output of a NOT gate is 0 when its input is 1, the output is 1 when its input is 0. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter. The following table shows the **truth table** and logic symbol of NOT gate.



A	Y = A'
0	1
1	0

2.4.2 Universal Gates:

NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function and any logic gate operation by using these gates.

NAND gate

NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs.

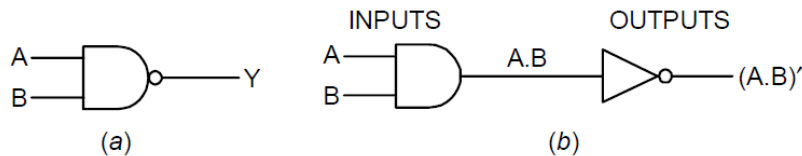


Fig. 2.2 (a) NAND gate logic symbol (b) AND gate and inverter being used to produce the NAND gate

Here A, B are the inputs and Y is the output of two input NAND gate. When both inputs are '1', the output, Y is '0'. If at least one of the inputs is zero, then the output, Y is '1'. This is just opposite to that of two input AND gate operation.

Boolean Expression	\downarrow NOT Symbol or $\overline{AB} = Y$ $A \cdot B = Y$ \uparrow AND Symbol or $(AB)' = Y$															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0
A	B	Y														
0	0	1														
0	1	1														
1	0	1														
1	1	0														

NOR gate

NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

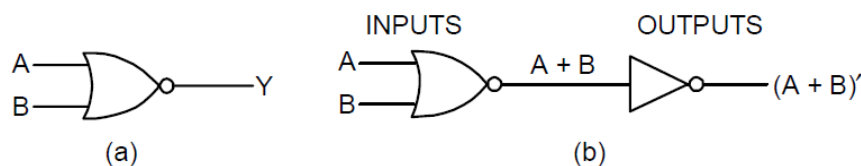
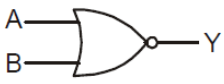


Fig. 2.3 (a) NOR gate logic symbol (b) Boolean expression for the output of NOR gate.

Here A, B are the inputs and Y is the output. If both inputs are '0', then the output, Y is '1'. If at least one of the inputs is '1', then the output, Y is '0'. This is just opposite to that of two input OR gate operation.

Boolean Expression	$\overline{A + B} = Y \quad \text{or} \quad (A + B)' = Y$ <p style="text-align: center;"> <small>↓ NOT Symbol</small> <small>↑ OR Symbol</small> </p>															
Logic Symbol																
Truth Table	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Y														
0	0	1														
0	1	0														
1	0	0														
1	1	0														

2.4.3 Exclusive-OR & Exclusive-NOR Gates:

Exclusive-OR gate:

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. 2.4 (a) with its Boolean expression. The symbol \oplus means the terms are XORed together.

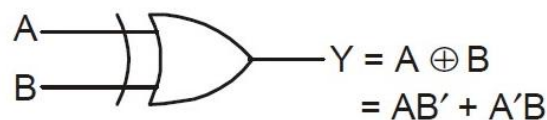


Fig. 2.4 (a) Exclusive-OR gate logic symbol.

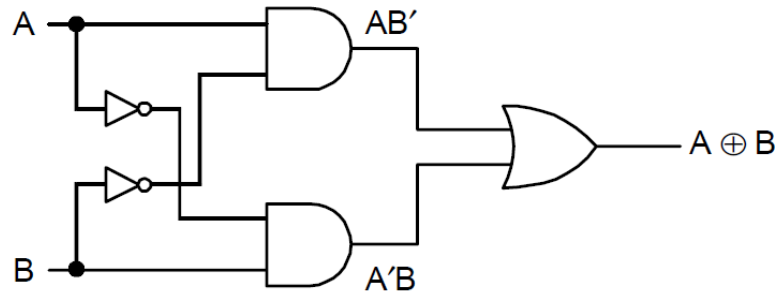
The truth table for XOR gate is shown in Fig. 2.4(b). Notice that if any but not all the inputs are 1, then the output will be 1. ‘The unique characteristic of the XOR gates that it produces a HIGH output only when the odd no. of HIGH inputs is present.’

INPUTS		OUTPUTS
A	B	$A \oplus B = AB' + A'B$
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 2.4 (b)

The XOR gate using AND-OR-NOT gates:

we know $A \oplus B = AB' + A'B$



The Exclusive 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 is shown in Fig. 2.5(a).

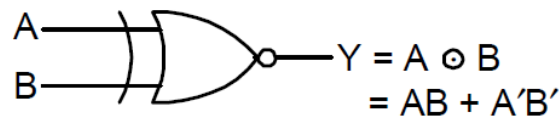


Fig. 2.5 (a)

Observe that it is the XOR symbol with the added invert bubble on the output side. The Boolean expression for XNOR is therefore, the invert of XOR function denoted by symbol \odot

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

'The unique output of the XNOR gate is a LOW only when an odd number of inputs are HIGH'. The truth table for XNOR gate is shown in Fig. 2.5 (b).

INPUTS		OUTPUTS
A	B	$A \odot B = AB + A'B'$
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 2.5 (b)

2.5 BOOLEAN FUNCTIONS

A binary variable can take the value of 0 or 1. A Boolean function is an expression formed with binary variable, the two binary operators OR and AND, the unary operator NOT, parentheses and an equal sign. For a given value of the variables, the function can be either 0 or 1. Consider, for example, the Boolean function

$$F_1 = xy'z$$

The function F is equal to 1 when $x = 1$, $y = 0$ and $z = 1$; otherwise $F_1 = 0$. This is an example of a Boolean function represented as an algebraic expression. A Boolean function may also be represented in a truth table. To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of n binary variables, and a column showing the combinations for which the function is equal to 1 or 0 as discussed previously. As shown in Table 2.1, there are eight possible distinct combinations for assigning bits to three variables. The table 2.1 shows that the function F_1 is equal to 1 only when $x = 1$, $y = 0$ and $z = 1$ and equal to 0 otherwise.

Consider now the function

$$F_2 = x'y'z + x'yz + xy'$$

$F_2 = 1$ if $x = 0, y = 0, z = 1$ or

$x = 0, y = 1, z = 1$ or

$x = 1, y = 0, z = 0$ or

$x = 1, y = 0, z = 1$

$F_2 = 0$, otherwise.

Table 2.1

x	y	z	F_1	F_2	F_3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	0

The number of rows in the table is 2^n , where n is the number of binary variables in the function.

The question now arises, is an algebraic expression of a given Boolean function unique? Or, is it possible to find two algebraic expressions that specify the same function? The answer is yes. Consider for example a third function.

$$F_3 = xy' + x'z$$

$F_3 = 1$ if $x = 1, y = 0, z = 0$ or

$x = 1, y = 0, z = 1$ or

$x = 0, y = 0, z = 1$ or

$x = 0, y = 1, z = 1$

$F_3 = 0$, otherwise.

From table, we find that F_3 is same as F_2 since both have identical 1's and 0's for each combination of values of the three binary variables. In general, two functions of n-binary variables are said to be equal if they have the same value for all possible 2^n combinations of the n-variables.

As a matter of fact, the manipulation of Boolean algebra is applied mostly to the problem of finding simpler expressions for the same function.

2.5.1 Transformation of Boolean Function into Logic Diagram:

A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR and NOT gates. Now we shall implement the three functions ($F_1 = xy'z$, $F_2 = x'y'z + x'yz + xy'$ and $F_3 = xy' + x'z$) discussed above as shown in Fig. 2.7.

1. Here we are using inverters (NOT gates) for complementing a single variable. In general, however, it is assumed that we have both the normal and complement forms available.
2. There is an AND gate for each product term in the expression.
3. An OR gate is used to combine two or more terms.

From the Fig. 2.7, it is obvious that the implementation of F_3 requires fewer gates and fewer inputs than F_2 . Since F_3 and F_2 are equal Boolean functions, it is more economical to implement F_3 form than the F_2 form. To find simpler circuits, we must know how to manipulate Boolean functions to obtain equal and simpler expression.

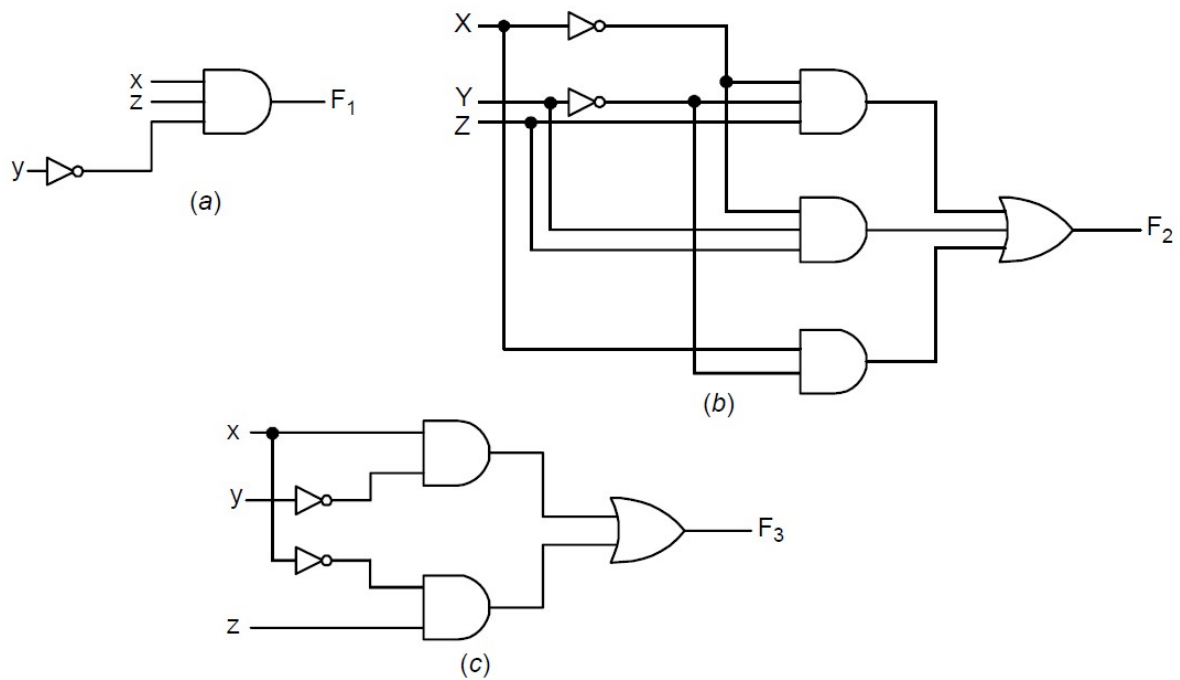


Fig. 2.7

2.5.2 Complement of a Function:

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F . The complement of a function may be derived algebraically through De Morgan's theorem. De Morgan's theorem can be extended to three or more variables. The three-variable form of the De Morgan's theorem is derived below:

$$\begin{aligned}
 (A + B + C)' &= (A + X)' && \text{Let } B + C = X \\
 &= A'X' && \text{by theorem 5(a)} \\
 &= A' \cdot (B + C)' && \text{substituting } B + C = X \\
 &= A' \cdot (B'C') && \text{theorem 5(a)} \\
 &= A'B'C'
 \end{aligned}$$

This theorem can be generalized as

$$(A + B + C + D + \dots + F)' = A'B'C'D' \dots F'$$

and its DUAL

$$(ABCD \dots F)' = A' + B' + C' + D' + \dots + F'$$

The generalized form of De Morgan's theorem states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

Example. Determine the complements of the following function:

$$F_1 = AB' + C'D$$

Solution.

$$F_1 = AB' + C'D$$

$$F_1' = (AB' + C'D)'$$

$$= (AB')' \cdot (C'D)'$$

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

2.6 REPRESENTATION OF BOOLEAN FUNCTIONS

Boolean functions (logical functions) are generally expressed in terms of logical variables. Any logical variable can take on only one of the two values 0 and 1 or any logical variable (binary variable) may appear either in its normal form (A) or in its complemented form (A'). An arbitrary logic function can be expressed in the following forms:

(i) Sum of Products (SOP)

(ii) Product of Sums (POS)

Product Term

The AND function is referred to as product. The logical product of several variables on which a function depends is considered to be a product term. The variables in a product term can appear either in complemented or uncomplemented (normal) form. For example, $AB'C$ is a product term.

Sum Term

The OR function is generally used to refer a sum. The logical sum of several variables on which a function depends is considered to be a sum term. Variables in a sum term also can appear either in normal or complemented form. For example, $A + B + C'$, is a sum term.

Sum of Products (SOP)

The logic sum of two or more product terms is called a 'sum of product' expression. It is basically an OR operation of AND operated variables such as $F = A'B + B'C + A'BC$.

Product of Sums (POS)

The logical product of two or more sum terms is called a 'product of sum' expression. It is basically an AND operation of OR operated variables such as $F = (A' + B) \cdot (B' + C) \cdot (A' + B + C)$.

2.6.1 Minterm and Maxterm Realization:

Consider two binary variables A and B combined with an AND operation. Since each variable may appear in either form (normal or complemented), there are four combinations, that are possible— $AB, A'B, AB', A'B'$.

Each of these four AND terms represent one of the four distinct combinations and is called a minterm, or a standard product or fundamental product. Now consider three variables—A, B and C. For a three variable function there are 8 minterms as shown in Table 2.2(a). (Since there are 8 combinations possible). The binary numbers from 0 to 7 are listed under three variables.

Each minterm is obtained from an AND term of the three variables, with each variable being primed (complemented form) if the corresponding bit of the binary number is a 0 and unprimed (normal form) if a 1. The symbol is m_j , where j denotes the decimal equivalent of the binary number of the minterm designated.

In a similar manner, n variables can be combined to form 2^n minterms. The 2^n different minterms may be determined by a method similar to the one shown in table for three variables.

Similarly, n -variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called maxterms or standard sums.

Each maxterm is obtained from an OR term of the n -variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1.

It is interesting to note that each maxterm is the complement of its corresponding minterm and vice versa. Now we have reached to a level where we are able to understand two very important properties of Boolean algebra through an example.

Table 2.2(a) Minterm and Maxterm for three binary variables

Decimal Eqt.	A	B	C	MINTERMS		MAXTERMS	
				Term	Designation	Term	Designation
0	0	0	0	$A'B'C'$	m_0	$A + B + C$	M_0
1	0	0	1	$A'B'C$	m_1	$A + B + C'$	M_1
2	0	1	0	$A'BC'$	m_2	$A + B' + C$	M_2
3	0	1	1	$A'BC$	m_3	$A + B' + C'$	M_3
4	1	0	0	$AB'C'$	m_4	$A' + B + C$	M_4
5	1	0	1	$AB'C$	m_5	$A' + B + C'$	M_5
6	1	1	0	ABC'	m_6	$A' + B' + C$	M_6
7	1	1	1	ABC	m_7	$A' + B' + C'$	M_7

Let we have a TV that is connected with three switches. TV becomes 'ON' only when at least two of the three switches are 'ON' (or high) and in all other conditions TV is 'OFF' (or low).

Let the three switches are represented by three variable A, B and C. The output of TV is represented by F. Since there are three switches (three variables), there are 8 distinct combinations possible that is shown in truth table (TT). (Table 2.2(b)).

Table 2.2(b)

SWITCHES			TV (o/p)
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

HIGH (ON) \rightarrow 1LOW (OFF) \rightarrow 0.

The TV becomes 'ON' at four combinations. These are 011, 101, 110 and 111. We can say that F is determined by expressing the combinations $A'BC$, $AB'C$, ABC' and ABC . Since each of these minterms result in $F = 1$, we should have

$$\begin{aligned} F &= A'BC + AB'C + ABC' + ABC \\ &= m_3 + m_5 + m_6 + m_7. \end{aligned}$$

This demonstrates an important property of Boolean algebra that 'Any Boolean function can be expressed as sum of minterms or as 'Sum of product'.

As mentioned, any TRUTH-TABLE INPUT/OUTPUT specifications can be expressed in a SOP expression. To facilitate this a shorthand symbology has been developed to specify such expressions. This is done by giving each row (MINTERM) in the TRUTH-TABLE a decimal number that is equivalent to the binary code of that row, and specifying the expression thus:

$$F = \Sigma(m_3, m_5, m_6, m_7)$$

which reads: $F =$ the sum-of-products of MINTERMS 3, 5, 6 and 7. This shorthand notation can be further shortened by the following acceptable symbology:

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

Now, continuing with the same example, consider the complement of Boolean function that can be read from Truth-table by forming a minterm for each combination that produces 0 in the function and by ORing

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

Now, if we take the complement of F' , we get F .

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

$$= M_0 \quad M_1 \quad M_2 \quad M_4$$

This demonstrates a second important property of the Boolean algebra that ‘Any Boolean function can be expressed as product-of-maxterms or as product of sums. The procedure for obtaining the product of maxterms directly from Truth-table is as; Form a maxterm for each combination of the variables that produces a 0 in the function, and then form the AND of all those functions. Output will be equal to F because in case of maxterms 0 is unprimed.




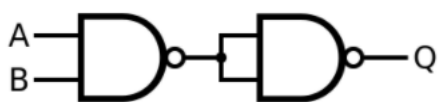

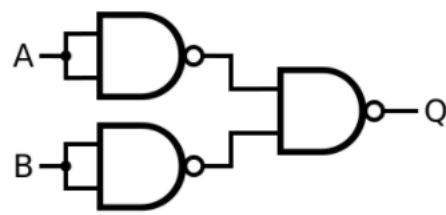
The shortened symbols for POS expressions is as follows—


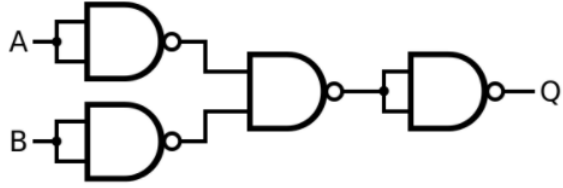

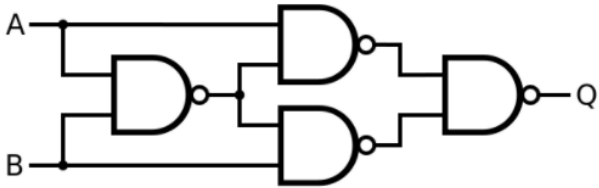
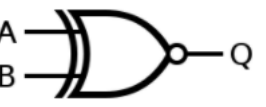
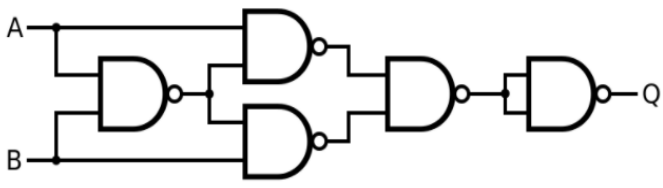
$$F = \Pi(M_0, M_1, M_2, M_4)$$

$$F = \Pi(0, 1, 2, 4)$$




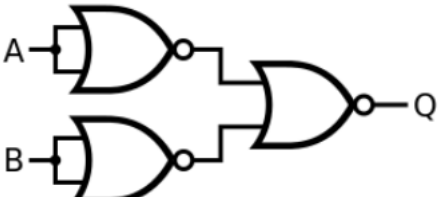
Boolean functions expressed as sum of minterms (sum of product terms) SOP or product of maxterms, (Product of sum terms) POS are said to be in CANONICAL form or STANDARD form.


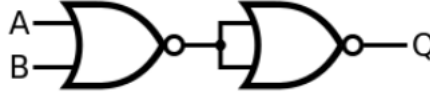

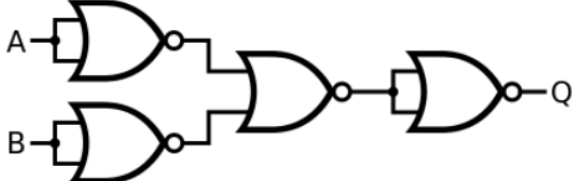

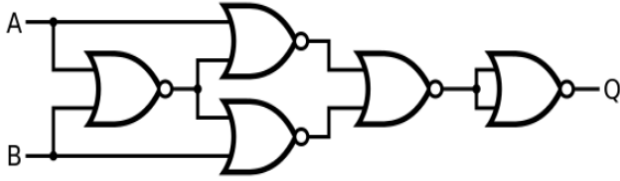
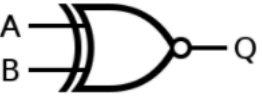
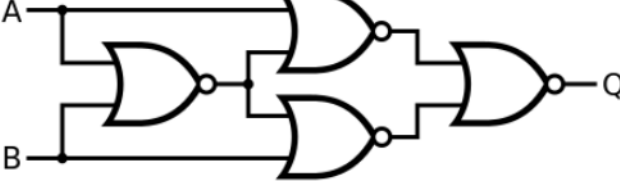
2.7 THE NAND GATE AS A UNIVERSAL GATE

Logic Function	Symbol	Circuit using NAND gates only
Inverter or NOT	 $Q = A'$	 $Q = A'$
AND	 $Q = A.B$	 $Q = A.B$
OR	 $Q = A+B$	 $Q = A+B$

Logic Function	Symbol	Circuit using NAND gates only
NOR	 $Q = (A+B)'$	 $Q = (A+B)'$
EX-OR	 $Q = A'B+AB'$	 $Q = A'B+AB'$
EX-NOR	 $Q = AB+A'B'$	 $Q = AB+A'B'$

2.8 THE NOR GATE AS A UNIVERSAL GATE

Logic Function	Symbol	Circuit using NOR gates only
Inverter or NOT	 $Q = A'$	 $Q = A'$
AND	 $Q = A.B$	 $Q = A.B$

Logic Function	Symbol	Circuit using NOR gates only
OR	 $Q = A+B$	 $Q = A+B$
NAND	 $Q = (AB)'$	 $Q = (AB)'$
EX-OR	 $Q = A'B+AB'$	 $Q = A'B+AB'$
EX-NOR	 $Q = AB+A'B'$	 $Q = AB+A'B'$

2.9 KARNAUGH MAPS (K-MAPS)

The map method provides simple straight forward procedure for minimizing Boolean functions that may be regarded as pictorial form of truth table. K-map orders and displays the minterms in a geometrical pattern such that the application of the logic adjacency theorem becomes obvious.

1. The K-map is a diagram made up of squares. Each square represents one minterm.
2. Since any function can be expressed as a sum of minterms, it follows that a Boolean function can be recognized from a map by the area enclosed by those squares. Whose minterms are included in the operation.

3. By various patterns, we can derive alternative algebraic expression for the same operation, from which we can select the simplest one. (One that has minimum member of literals).

Now, let us start with a two variable K map.

2.9.1 Two and Three Variable K Map:

If we examine a two variable truth table, Fig. 2.8(a) we can make some general observations that support the geometric layout of K Map shown in Fig. 2.8(b).

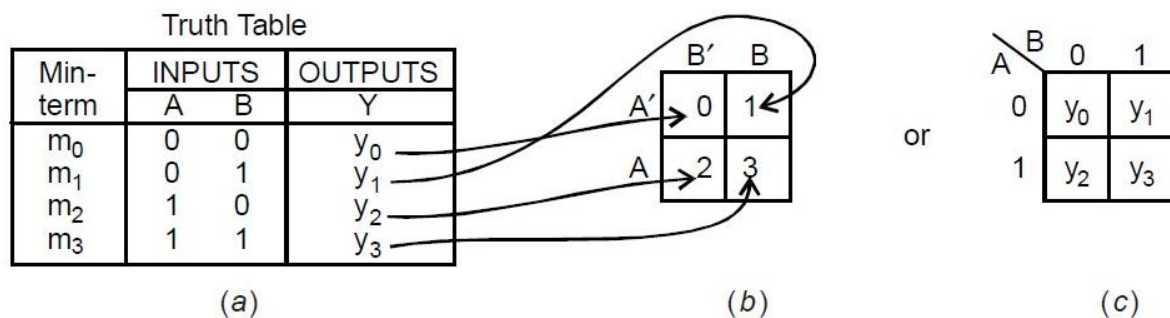


Fig. 2.8

The four squares (0, 1, 2, 3) represent the four possible combinations of A and B in a two variable truth table. Square 1 in the K-map; then, stands for $A'B'$, square 2 for $A'B$, and so forth. The map is redrawn in Fig. 2.8(c) to show the relation between the squares and the two variables. The 0's and 1's marked for each row and each column designate the values of variables A and B respectively. A appears primed in row 0 and unprimed in row 1. Similarly, B appears primed in column 0 and unprimed in column 1.

Now let us map a Boolean Function $Y = A + B$.

Method I—(i) Draw the truth table of given function. [Fig. 2.9(a)]

Min-term	A	B	Y
m_0	0	0	0
m_1	0	1	1
m_2	1	0	1
m_3	1	1	1

Fig. 2.9(a)

(ii) Draw a two variable K map and fill those squares with a 1 for which the value of minterm in the function is equal to 1. [Fig. 2.9(b)]

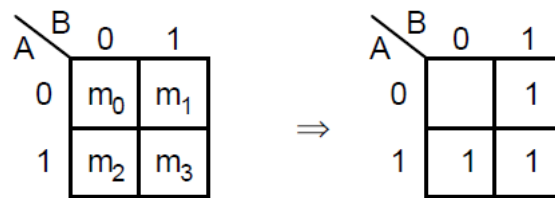


Fig. 2.9(b)

The empty square in the map represents the value of minterms [m_0 (or $A'B'$)] that is equal to zero in the function. Thus, actually this empty square represents zero.

Now examine a three variable truth table shown in Fig. 2.10 (a).

Min term	INPUTS			OUTPUT
	A	B	C	Y
m_0	0	0	0	y_0
m_1	0	0	1	y_1
m_2	0	1	0	y_2
m_3	0	1	1	y_3
m_4	1	0	0	y_4
m_5	1	0	1	y_5
m_6	1	1	0	y_6
m_7	1	1	1	y_7

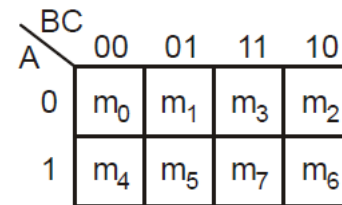


Fig. 2.10(b)

Fig. 2.10(a)

Here we need a K-map with 8 squares represent all the combination (Minterms) of input variables A, B and C distinctly. A three-variable map is shown in Fig. 2.10 (b).

It is very important to realize that in the three variable K-map of Fig. 2.10 (b), the minterms are not arranged in a binary sequence, but similar to 'Gray Code' sequence. The gray code sequence is a unit distance sequence that means only one-bit changes in listing sequence.

Our basic objective in using K-map is the simplify the Boolean function to minimum number of literals. The gray code sequencing greatly helps in applying. 'Logic Adjacency theorem' to adjacent squares that reduces number of literals.

The map in Fig. 2.10 (b) is redrawn in Fig. 2.10 (c) that will be helpful to make the pictures clear.

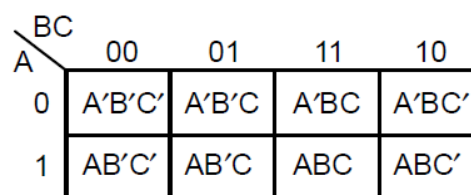


Fig. 2.10(c)

We can see that any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in other.

For example, m_3 ($A'BC$) and m_7 (ABC) are two adjacent squares. Variable A is primed in m_3 and unprimed in m_7 , whereas the other two variables are same in both squares. Now applying 'logic adjacency' theorem, it is simplified to a single AND term of only two literals. To clarify this, consider the sum of m_3 and $m_7 \rightarrow m_3 + m_7 = A'BC + ABC = BC(A + A') = BC$.

2.9.2 Boolean Expression Minimization Using K-Map:

1. Construct the K-map as discussed. Enter 1 in those squares corresponding to the minterms for which function value is 1. Leave empty the remaining squares. Now in following steps the square means the square with a value 1.
2. Examine the map for squares that cannot be combined with any other squares and form group of such single squares.
3. Now, look for squares which are adjacent to only one other square and form groups containing only two squares and which are not part of any group of 4 or 8 squares. A group of two squares is called a pair.
4. Next, group the squares which result in groups of 4 squares but are not part of an 8-squares group. A group of 4 squares is called a quad.
5. Group the squares which result in groups of 8 squares. A group of 8 squares is called octet.
6. Form more pairs, quads and outlets to include those squares that have not yet been grouped, and use only a minimum no. of groups. There can be overlapping of groups if they include common squares.
7. Omit any redundant group.
8. Form the logical sum of all the terms generated by each group.

Using Logic Adjacency Theorem, we can conclude that,

— a group of two squares eliminates one variable,

— a group of four squares eliminates two variable and a group of eight squares eliminates three variables.

Example 1. Simplify the Boolean for $F = AB + AB' + A'B$. Using two variable K-map.

This function can also be written as, $F(A, B) = \Sigma(1, 2, 3)$

Step 1. Make a two variable K-map and enter 1 in squares corresponding to minterms present in the expression and leave empty the remaining squares.

	B	0	1
A	0	m_0	m_1
	1	m_2	m_3

Step 2. There are no 1's which are not adjacent to other 1's. So, this step is discarded.

	B	0	1
A	0		1
	1	1	1

Step 3. m_1 is adjacent to m_3 , therefore, forms a group of two squares and is not part of any group of 4 squares. [A group of 8 squares is not possible in this case].

	B	0	1
A	0		1
	1	1	1

Similarly, m_2 is also adjacent to m_3 therefore forms another group of two squares and is not a part of any group of 4 squares.

Step 4 and 5. Discarded because these in no quad or octet.

Step 6. All the 1's have already been grouped.

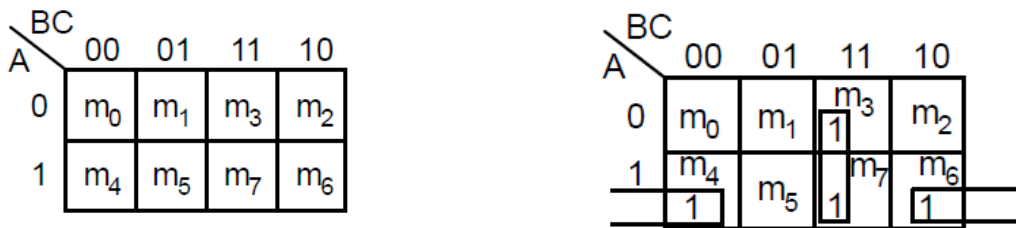
There is an overlapping of groups because they include common minterm m_3 .

Step 7. There is no redundant group.

Step 8. The terms generated by the two groups are 'OR' operated together to obtain the expression for F as follows:

$F = A$ \downarrow From group $m_2 m_3$ \downarrow This row is corresponding to the value of A is equal to 1.	$+$	B \downarrow From group $m_1 m_3$ \downarrow This column is corresponding to the value of B equal to 1.
----------------------------------------------------------------------------------------------------------------------------------------------	-----	---------------------------------------------------------------------------------------------------------------------------------------

Example 2. Simplify the Boolean Function $F(A, B, C) = \Sigma(3, 4, 6, 7)$.



$F = AC'$ \downarrow From group $m_4 m_6$ The row is corresponding to the value of A = 1 and in the two columns (00 \rightarrow $B'C'$ and the 10 $\rightarrow BC'$), the value C = 0 $\Rightarrow C'$ is common $= AC'$	$+$	BC \downarrow From group $m_3 m_7$. Correspond to both rows (A = 0 and A = 1) so A is omitted, and single column (B = 1 and C = 1), <i>i.e.</i> , BC.
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.9.3 Four and Five Variable K Map:

The map for Boolean functions of four binary variables (w, x, y, z) is shown in Fig. 2.11. In Fig. 2.11(a) are listed the 16 minterms and the squares assigned to each. In Fig. 2.11(b), the map is redrawn to show the relationship between the squares and the four variables.

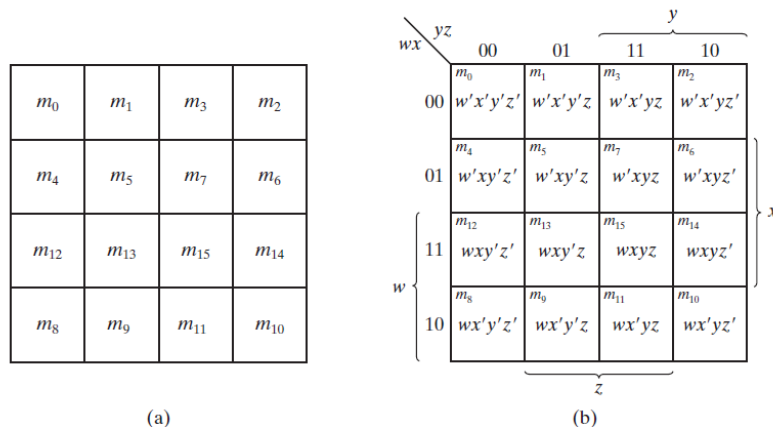
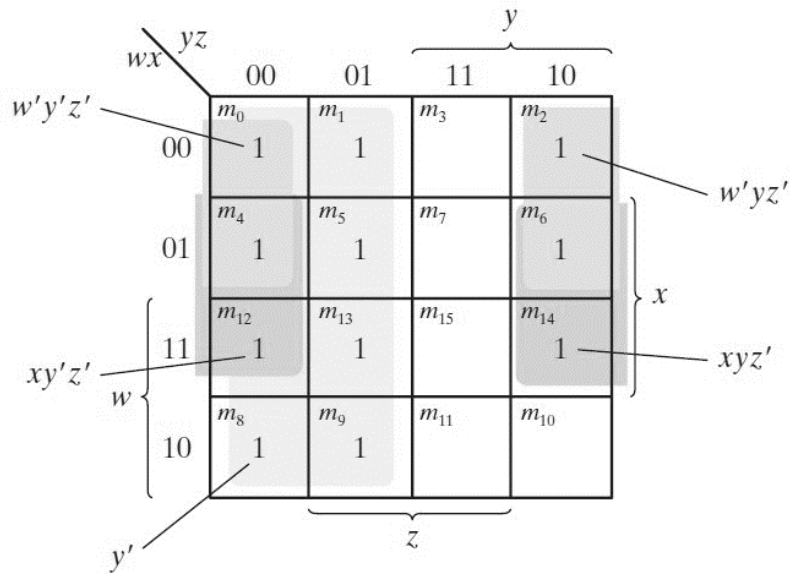


Fig. 2.11

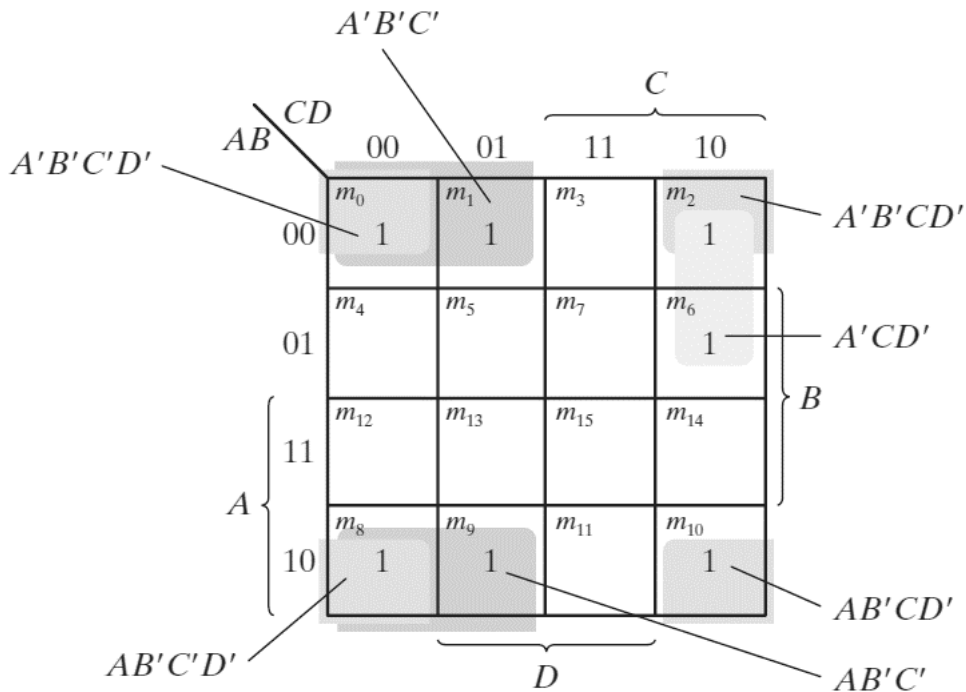
Example 3. Simplify the Boolean Function $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

$$F = y' + w'z' + xz'$$



Note: $w'y'z' + w'yz' = w'z'$
 $xy'z' + xyz' = xz'$

Example 4. Simplify the Boolean Function $F = A'B'C' + B'CD' + A'BCD' + AB'C'$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
 $AB'C'D' + AB'CD' = AB'D'$
 $A'B'D' + AB'D' = B'D'$
 $A'B'C' + AB'C' = B'C'$

UNIT – 3**COMBINATIONAL DIGITAL CIRCUITS****3.1 INTRODUCTION**

Combinational logic circuits are circuits in which the output at any time depends upon the combination of input signals present at that instant only, and does not depend on any past conditions. The block diagram of a combinational circuit with m inputs and n outputs is shown in Fig. 3.1.

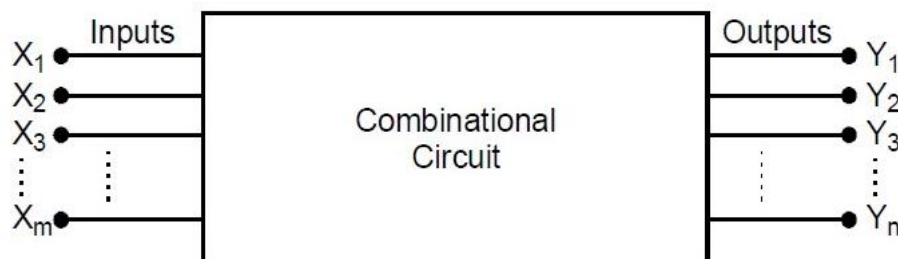


Fig. 3.1 Block diagram of combinational logic circuit

In particular, the output of particular circuit does not depend upon any past inputs or outputs i.e., the output signals of combinational circuits are not feedback to the input of the circuit. Moreover, in a combinational circuit, for a change in the input, the output appears immediately, except for the propagation delay through circuit gates.

The combinational circuit block can be considered as a network of logic gates that accept signals from inputs and generate signals to outputs. For m -input variables, there are 2^m possible combinations of binary input values. Each input combination to the combinational circuit exhibits a distinct (unique) output. Thus, a combinational circuit can be described by n -Boolean functions, one for each input combination, in terms of m -input variables with n is always less than or equal to 2^m . [$n < 2^m$].

3.2 COMBINATIONAL CIRCUIT DESIGN PROCEDURE

It involves following steps:

Step 1: From the word description of the problem, identify the inputs and outputs and draw a block diagram.

Step 2: Make a truth table based on problem statement which completely describes the operations of circuit for different combinations of inputs.

Step 3: Simplified output functions are obtained by algebraic manipulation, k-map method.

Step 4: Implement the simplified expression using logic gates.

To explain the procedure, let us take an example that we have already been used in unit-2.

Example: A TV is connected through three switches. The TV becomes 'on' when at least two switches are in 'ON' position; In all other conditions, TV is 'OFF'.

Solution. Step 1: The TV is connected with 3 switches; thus, there are three inputs to TV, represented by variables say A, B and C. The o/p of TV is represented by variable say, F. The block diagram is shown in Fig. 3.2:

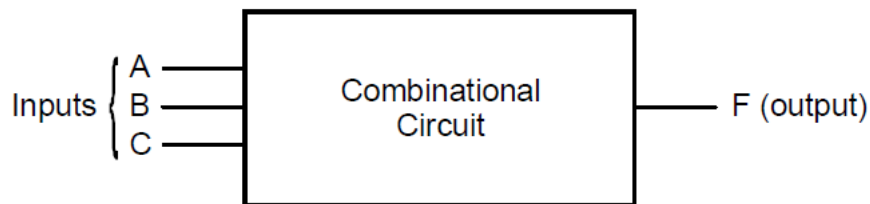


Fig. 3.2

Step 2:

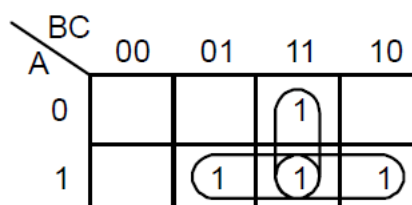
<i>TV switches ← INPUTS</i>			<i>OUTPUTS</i>
<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i>
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

0 → switch off

1 → switch on

It means for the input combinations in which there are two or more 1's, the output $F = 1$ (TV is ON) and for rest combinations, output $F = 0$ (TV is OFF).

Step 3: In general, in simplifying Boolean functions up to four variables, the best method is K-map technique. Thus, using a 3 variable K-map, we can simplify the function obtained in step 2. We get $F = AB + AC + BC$



Step 4: For implementation we need three 'AND' gates and one 'OR' gate as shown in Fig. 3.3.

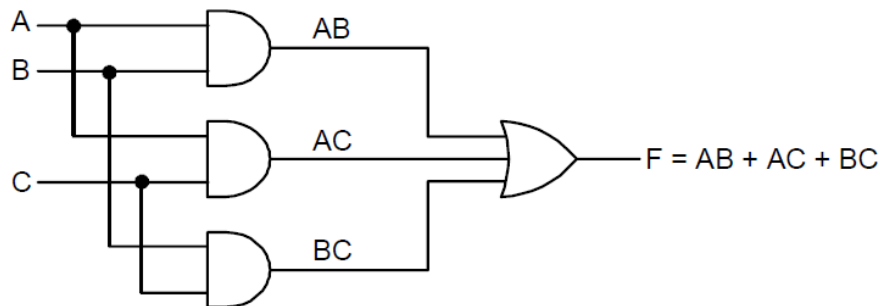


Fig. 3.3

3.3 ARITHMETIC CIRCUITS

The logic circuits which are used for performing the digital arithmetic operations such as addition, subtraction, multiplication and division are called arithmetic circuits.

3.3.1 Adders

The most common arithmetic operation in digital systems is the addition of two binary digits. The combinational circuit that performs this operation is called as adder.

Half Adder

1. A half adder is a combinational logic circuit which is used to add two 1-bit information. It has two inputs A and B. that are two 1-bit numbers, and two output sum (S) and carry (C) produced by addition of two bits. Fig. 3.4. shows a half adder (HA).

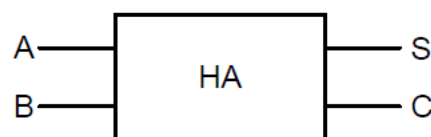
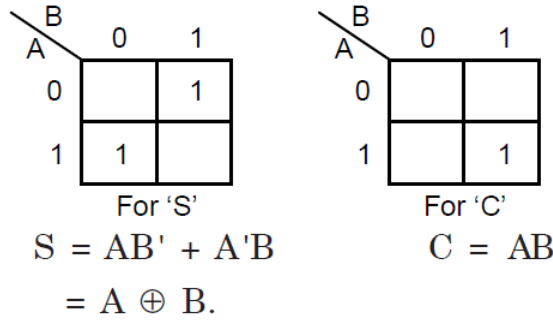


Fig. 3.4

2. Truth Table

<i>Inputs</i>		<i>Outputs</i>	
<i>A</i>	<i>B</i>	<i>S</i>	<i>C</i>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

3. Using a two-variable k-map, separately for both outputs S and C.



4. Logical Implementation.

(i) Using Basic gates (as shown in Fig. 3.5(a)).

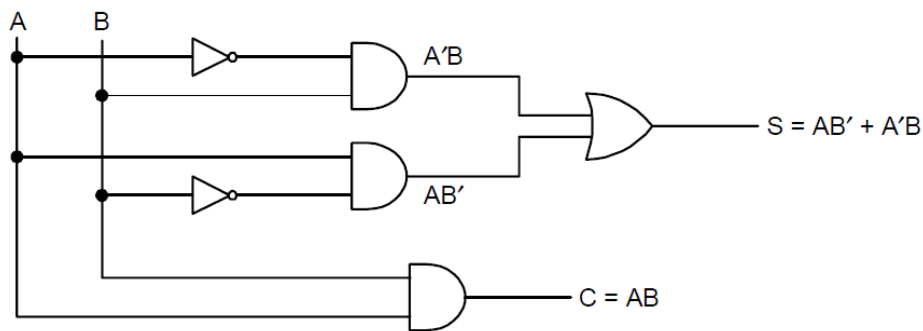


Fig. 3.5(a)

(ii) Using XOR gate as shown in Fig. 3.5 (b).

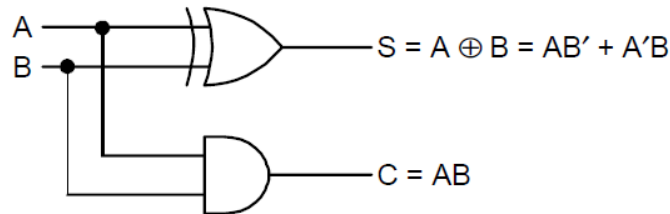


Fig. 3.5(b)

Full Adder

Full adder is a combinational circuit that performs the addition of three binary digits.

1. Fig. 3.6 shows a full adder (FA). It has three inputs A, B and C and two outputs S and C₀ produced by addition of three input bits. Carry output is designated C₀ just to avoid confusion between with i/p variable C.

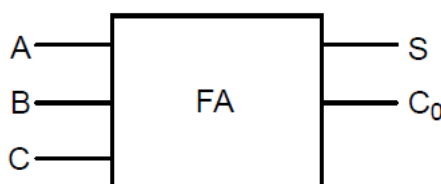
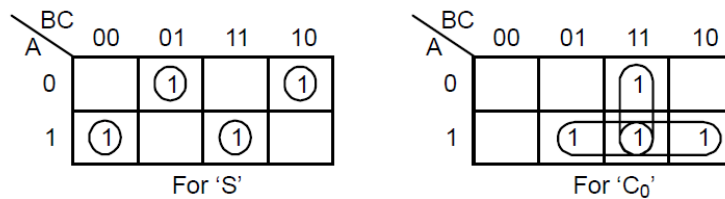


Fig. 3.6

2. **Truth Table:** The eight possible combinations of three input variables with their respective outputs is shown. We observe that when all the three inputs are 1, the sum and carry both outputs, are 1.

Inputs			Output	
A	B	C	S	C ₀
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

3. Using a three-variable map for both outputs.



$$S = ABC + AB'C' + A'BC' + A'B'C \text{ and } C_0 = AB + AC + BC.$$

4. Logical Implementation. (i) Using basic gates as shown in Fig. 3.7.

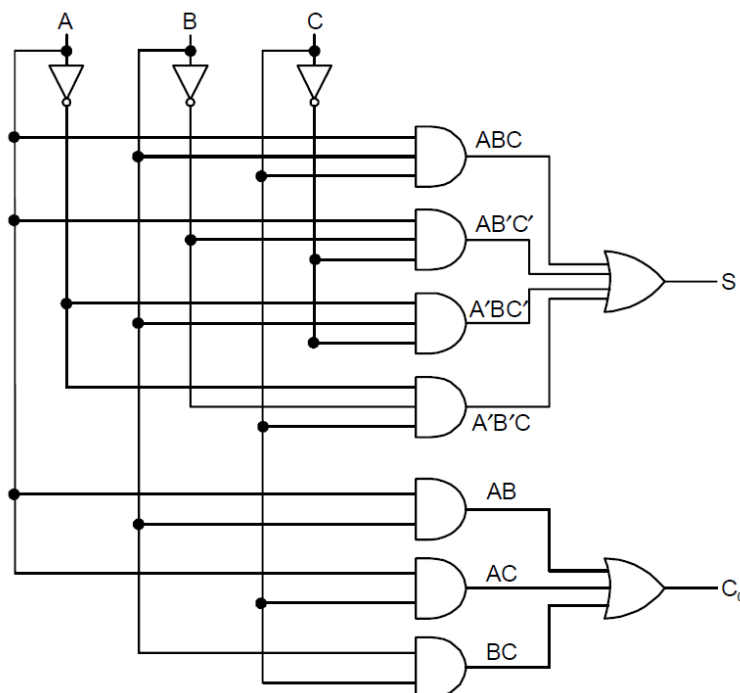


Fig. 3.7

(ii) A 'Full Adder' can also be implemented using two half adders and an 'OR' Gate as shown in Fig. 3.8.

$$\begin{aligned}
 \text{The Sum } S &= ABC + AB'C' + A'BC' + A'B'C \\
 &= ABC + A'B'C + AB'C' + A'BC' \\
 &= C(AB + A'B') + C'(AB' + A'B) \\
 &= C(AB' + A'B)' + C'(AB' + A'B) \\
 &= (A \oplus B) \oplus C
 \end{aligned}$$

$$\begin{aligned}
 \text{and the carry } C_0 &= AB + AC + BC \\
 &= AB + C(A + B) \\
 &= AB + C(A + B)(A + A')(B + B') \\
 &= AB + C[AB + AB' + A'B] \\
 &= AB + ABC + C(AB' + A'B) \\
 &= AB(1 + C) + C(A \oplus B) \\
 &= AB + C(A \oplus B)
 \end{aligned}$$

Therefore, $S = (A \oplus B) \oplus C$ and $C_0 = AB + C(A \oplus B)$

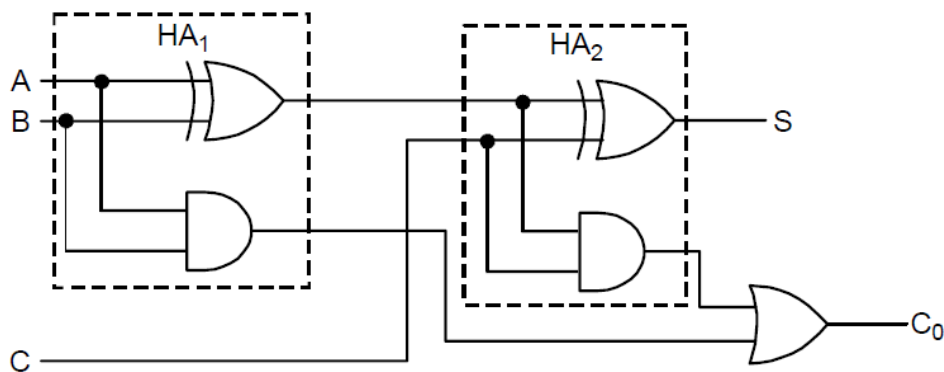
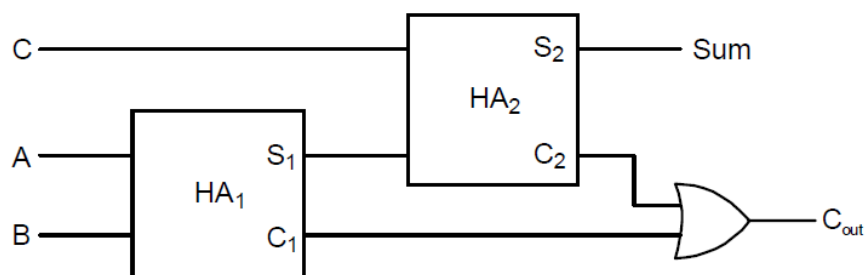


Fig. 3.8

Block Diagram representation of a full adder using two half adders:



S_1 and C_1 are outputs of first half adder (HA_1)

S_2 and C_2 are outputs of second half adder (HA_2)

A, B and C are inputs of Full adder.

Sum and C_{out} are outputs of full adder.

3.3.2 Subtractors

The logic circuits used for binary subtraction, are known as ‘binary subtractors’.

Half Subtractor

The half subtractor is a combinational circuit which is used to perform the subtraction of two bits.

- Fig. 3.9 shows a half subtractor. (HS) It has two inputs, A (minuend) and B (subtrahend) and two outputs D (difference) and B₀ (Borrow). [The symbol for borrow (B₀) is taken to avoid confusion with input variable B] produced by subtractor of two bits.

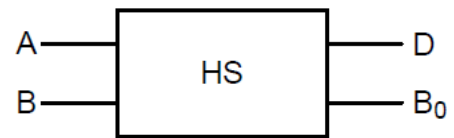


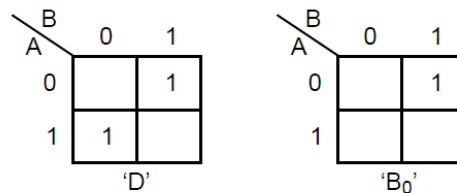
Fig. 3.9 Half subtractor

2. Truth Table

The difference output is 0 if A = B and 1 if A ≠ B; the borrow output is 1 whenever A < B. If A < B, the subtraction is done by borrowing 1 from the next higher order bit.

Inputs		Outputs	
A	B	D	B ₀
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

3. Using a two-variable map, for outputs D and B.



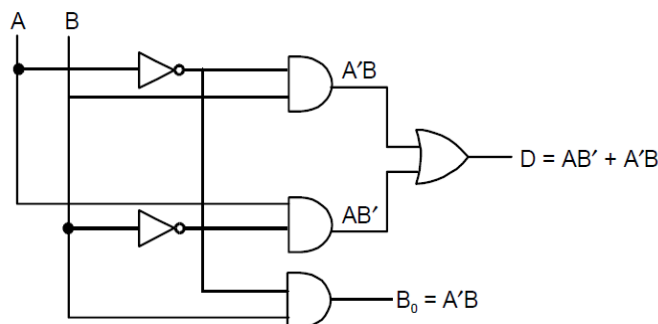
$$D = AB' + A'B$$

$$= A \oplus B$$

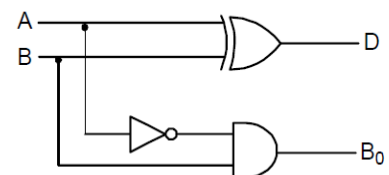
$$B_0 = A'B$$

4. Logical Implementation shown in Fig. 3.10

3.10(a) Using Basic gates



3.10(b) Using XOR gate



Full subtractor

Full subtractor is a combinational circuit that performs the subtraction of three binary digits.

1. Fig. 3.11 shows a full subtractor (FS). It has three inputs A, B and C and two outputs D and B₀ produced by subtraction of three input bits.
2. **Truth Table**

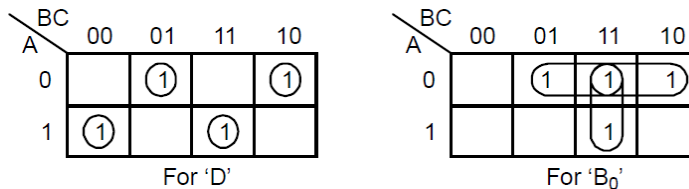


Fig. 3.11 Full subtractor

The eight possible combinations of three input variables with their respective outputs is shown. We observe that when all the three inputs are 1, the difference and borrow both outputs are 1.

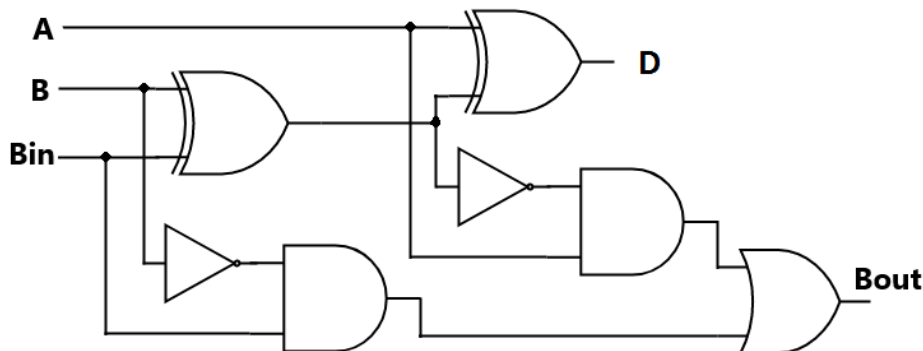
Inputs			Output	
A	B	C	B ₀	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

3. Using a three-variable map for both outputs.



$$D = ABC + AB'C' + A'BC' + A'B'C, B_0 = A'B + A'C + BC$$

4. Logical implementation (i) Using basic gates:



(ii) A 'full subtractor' can also be implemented using two 'half subtractors' and an 'OR' gate as shown in Fig. 3.12.

The difference

$$\begin{aligned}
 'D' &= ABC + AB'C' + A'BC' + A'B'C \\
 &= ABC + A'B'C + AB'C' + A'BC' \\
 &= C (AB + A'B') + C' (AB' + A'B) \\
 &= C (AB' + A'B)' + C' (AB' + A'B) \\
 &= C (A \oplus B)' + C' (A \oplus B) \\
 &= (A \oplus B) \oplus C
 \end{aligned}$$

and the borrow

$$\begin{aligned}
 B_0 &= A'B + A'C + BC \\
 &= A'B + C (A' + B) \\
 &= A'B + C (A' + B) (A + A') (B + B') \\
 &= A'B + C [A'B + AB + A'B'] \\
 &= A'B + A'BC + C (AB + A'B') \\
 &= A'B (C + 1) + C (A \oplus B)' \\
 &= A'B + C (A \oplus B)'
 \end{aligned}$$

$$D = (A \oplus B) \oplus C \text{ and } B_0 = A'B + C (A \oplus B)'$$

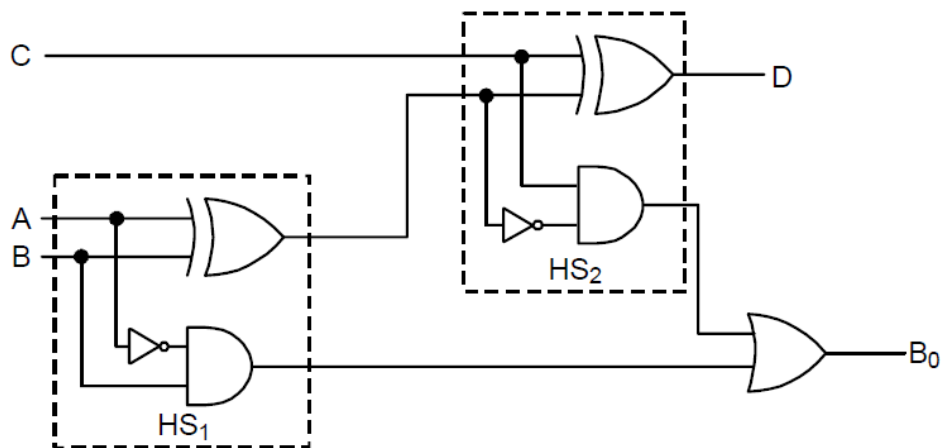


Fig. 3.12 (a)

Block Diagram Representation of a full subtractor using two half subtractors:

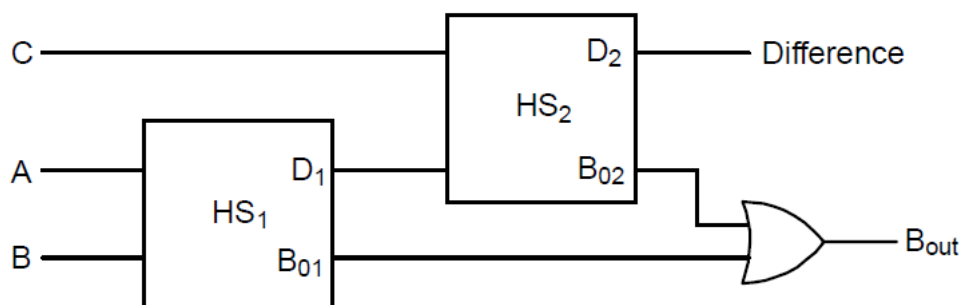


Fig. 3.12(b)

Parallel Adder (4-bit parallel adder)

Full adder is a combinational circuit that adds three binary digits. When we add two numbers of any length, the terms we have to deal with are:

Input carry, Augend, Addend, sum and output carry. We simply start adding two binary digits from LSB (rightmost positioned bits). At this position, the input carry is always equal to zero. After addition, we get sum and output carry. This output carry works as the input carry to the next higher positioned augend and addend bits. Next, we add augend and addend bits along with the input carry that again produces sum and output carry. The process repeats upto MSB position (leftmost positioned bits).

We observe that in the process of addition we are actually adding three digits – the input carry, the augend bit and the addend bit. And, we are getting two outputs the sum and the output carry.

This can be illustrated by the following example. Let the 4-bits words to be added be represented by:

$$A_3 A_2 A_1 A_0 = 1 1 0 1 \text{ and } B_3 B_2 B_1 B_0 = 0 0 1 1.$$

Significant Place	3	2	1	0	
Input Carry	1	1	1	0	← Carry-In
Augend Word	1	1	0	1	
Addend Word	0	0	1	1	
Sum	0	0	0	0	
Output carry	1	1	1	1	
Carry-out					

Now, if we compare this with the full adder circuit, we can easily observe that the two inputs (A and B) are augend and addend bits with the third input (c) as the input carry. Similarly, two outputs are sum (s) and output carry (C0).

To add two n-bit numbers, the parallel method uses n full adder circuits and all bits of addend and augend bits are applied simultaneously. The output carry from one full adder is connected to the input carry of the full adder one position to its left.

The 4-bit adder using full adder circuit is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in Fig. 3.13.

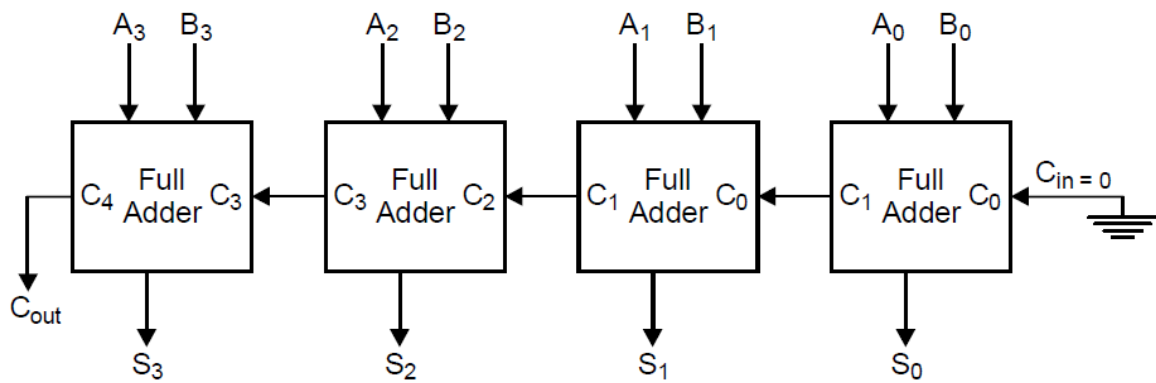


Fig. 3.13 4-bit binary parallel adder

The addition operation is illustrated in the following example. Let the 4-bit words to be added be represented by $A_3A_2A_1A_0 = 1\ 0\ 1\ 0$ and $B_3B_2B_1B_0 = 0\ 0\ 1\ 1$.

Subscript i	3	2	1	0
Input carry C_i	0	1	0	0
Augend A_i	1	0	1	0
Addend B_i	0	0	1	1
Sum S_i	1	1	0	1
Output carry C_{i+1}	0	0	1	0

← Significant place.

An alternative block diagram representation of a 4-bit binary parallel adder is shown in Fig.3.14

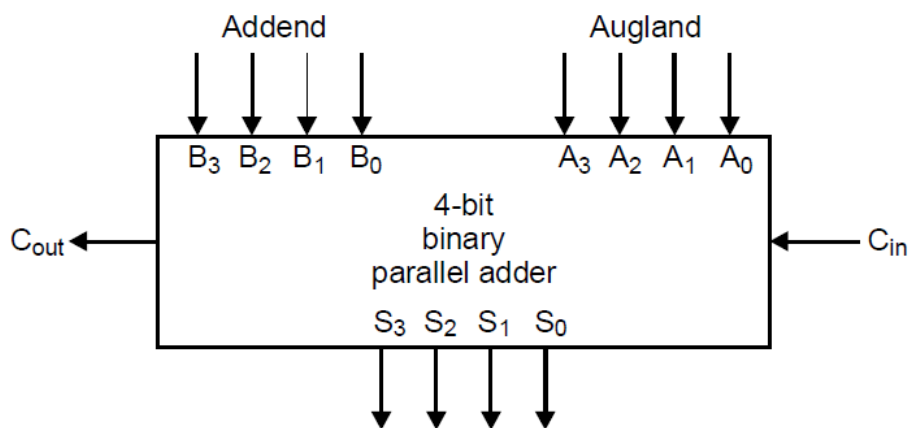


Fig. 3.14 4-bit binary parallel adder

3.4 DECODERS

A decoder is a combinational logic circuit that converts coded input into coded output, where both input and output coders are different. It has n -inputs and 2^n -output lines. The n -input lines carry binary information and the 2^n -output lines carry 1-out of m -bit form code i.e., at any time out of 2^n output lines one output is activated.

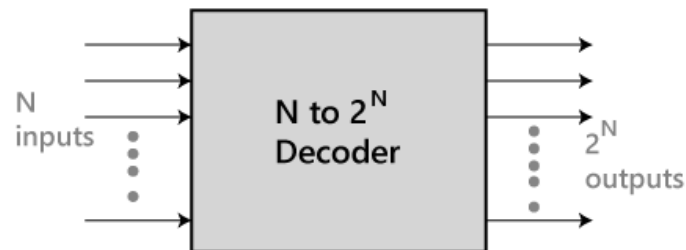


Fig. 3.15 Decoder block diagram

There are various types of decoders based on the input and output lines, they are as follows:

2 to 4 Decoder

In a 2-to-4-line decoder, there are two inputs namely A_1 and A_0 and four outputs i.e., Y_0 , Y_1 , Y_2 and Y_3 . It has an enable signal E as input which is used to control the operation of the decoder (just like an on/off switch). It is an active high enable, means any one of the four output lines get activated based on the input value when this enable signal value is 1, otherwise the output is un-decoded output.

The following Fig. 3.16 shows the block diagram and truth table of a 2-to-4-line decoder.

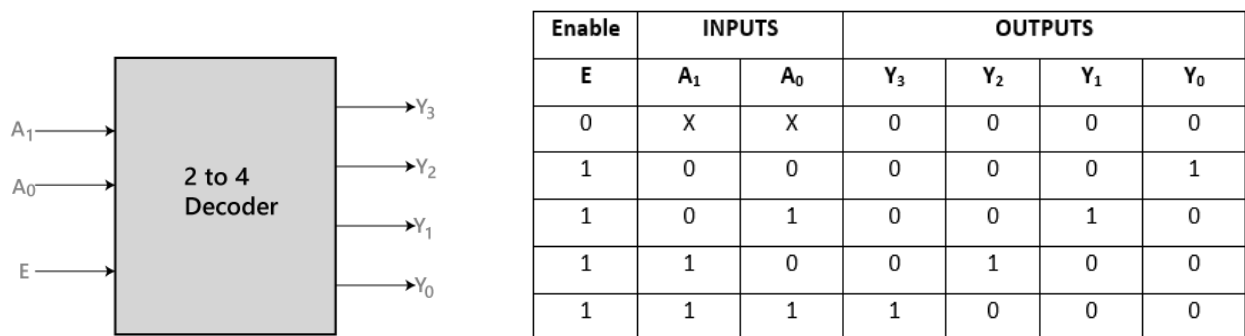


Fig. 3.16 Block diagram and truth table of a 2-to-4-line decoder

The logical expression of the term Y_0 , Y_1 , Y_2 , and Y_3 is as follows:

$$Y_3 = E \cdot A_1 \cdot A_0$$

$$Y_2 = E \cdot A_1 \cdot A_0'$$

$$Y_1 = E \cdot A_1' \cdot A_0$$

$$Y_0 = E \cdot A_1' \cdot A_0'$$

Logical circuit of the above expressions is given below Fig. 3.17

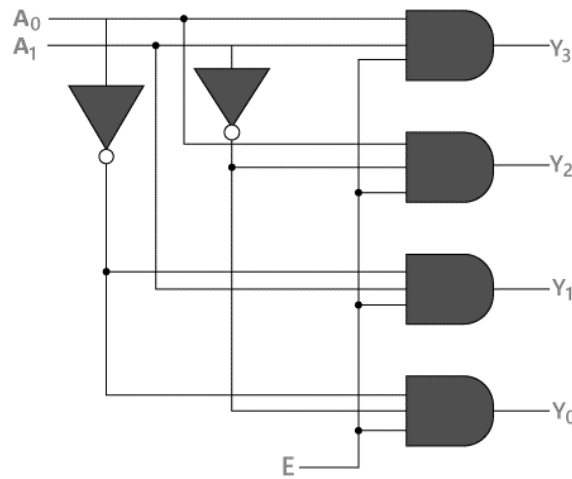


Fig. 3.17. A 2 to 4 decoder logic diagram

3 to 8 Decoder

In a 3-to-8-line decoder, there are three inputs namely A_2 , A_1 and A_0 and eight outputs i.e., Y_0 , Y_1 , Y_2 , Y_3 , Y_4 , Y_5 , Y_6 and Y_7 . It has an enable signal E as input which is used to control the operation of the decoder (just like an on/off switch). It is an active high enable, means any one of the eight output lines get activated based on the input value when this enable signal value is 1, otherwise the output is un-decoded output.

The following Fig. 3.18 shows the block diagram and truth table of a 3-to-8-line decoder.

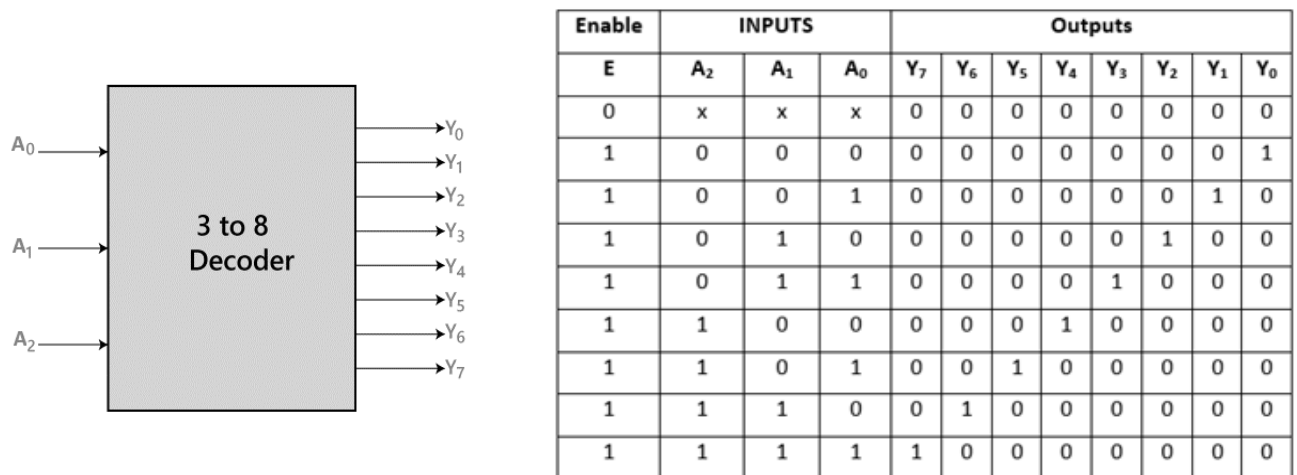


Fig. 3.18 Block diagram and truth table of a 3-to-8-line decoder

The logical expression of the term Y_0 , Y_1 , Y_2 , Y_3 , Y_4 , Y_5 , Y_6 , and Y_7 is as follows:

$$\begin{aligned}
 Y_0 &= A_0'.A_1'.A_2' & Y_1 &= A_0.A_1'.A_2' & Y_2 &= A_0'.A_1.A_2' & Y_3 &= A_0.A_1.A_2' \\
 Y_4 &= A_0'.A_1'.A_2 & Y_5 &= A_0.A_1'.A_2 & Y_6 &= A_0'.A_1.A_2 & Y_7 &= A_0.A_1.A_2
 \end{aligned}$$

Logical circuit of the above expressions is given below Fig. 3.19

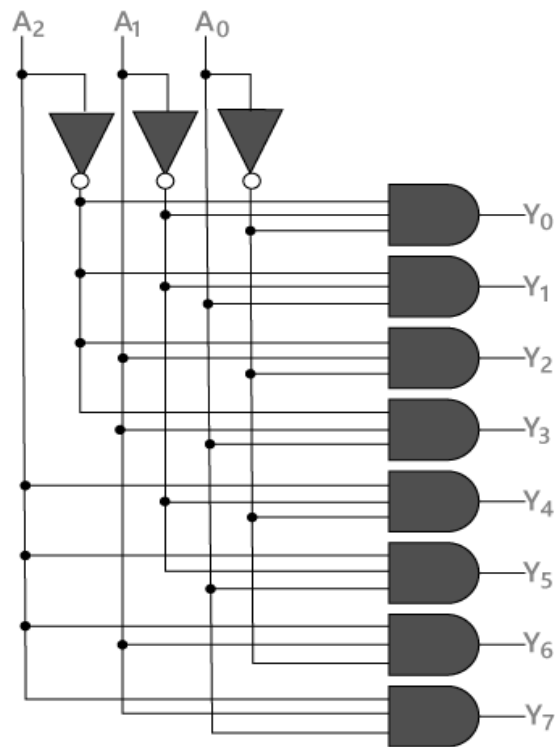


Fig. 3.19. A 3 to 8 decoder logic diagram

3.5 ENCODERS

An encoder is a combinational logic circuit, that does the reverse operation of a decoder. It has 2^n -inputs and n -outputs lines. The input code is 1-out of m -bit form and the output code is binary. At any time one input signal is set to 1 and the output is equal to its binary value.

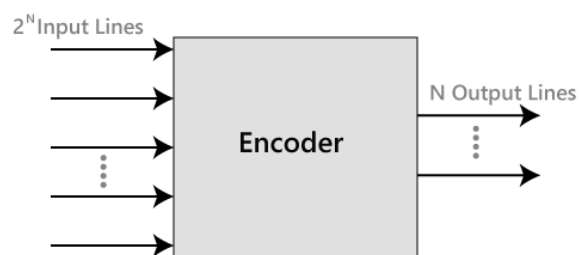


Fig. 3.20 Encoder block diagram

There are various types of encoders based on the input and output lines, they are as follows:

4-to-2-line Encoder

In 4-to-2-line encoder, there are total of four inputs, i.e., Y_0 , Y_1 , Y_2 , and Y_3 , and two outputs, i.e., A_0 and A_1 . In 4-input lines, one input-line is set to true at a time to get the respective binary code in the output side. Below are the block diagram and the truth table of the 4-to-2-line encoder

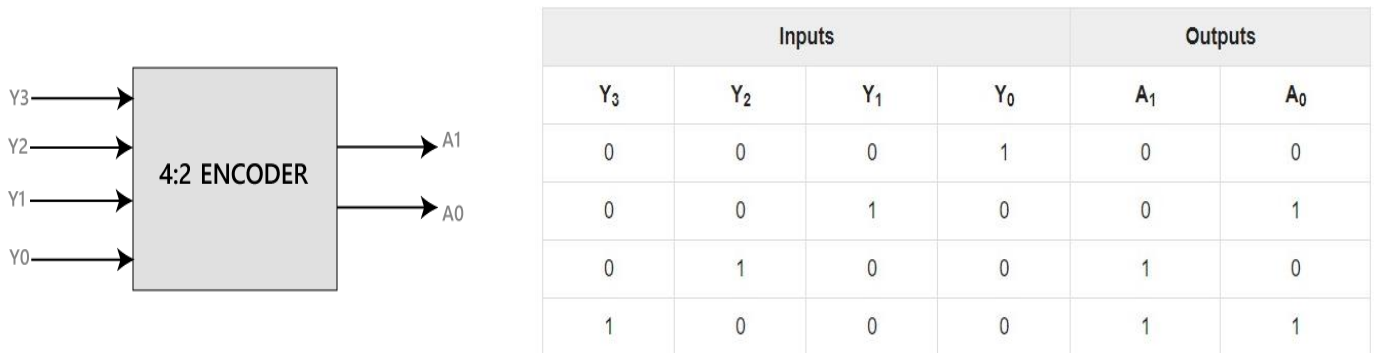


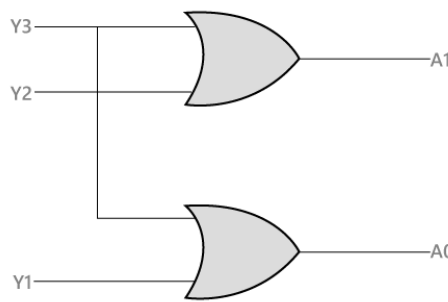
Fig. 3.21 Block diagram and truth table of a 4-to-2-line Encoder

The logical expression of the term A₀ and A₁ is as follows:

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

Logical circuit of the above expressions is given below:



8-to-3-line Encoder

The 8-to-3-line Encoder is also known as Octal to Binary Encoder. In 8-to-3-line encoder, there is a total of eight inputs, i.e., Y₀, Y₁, Y₂, Y₃, Y₄, Y₅, Y₆, and Y₇ and three outputs, i.e., A₀, A₁, and A₂. In 8-input lines, one input-line is set to true at a time to get the respective binary code in the output side. Below are the block diagram and the truth table of the 8-to-3-line encoder.

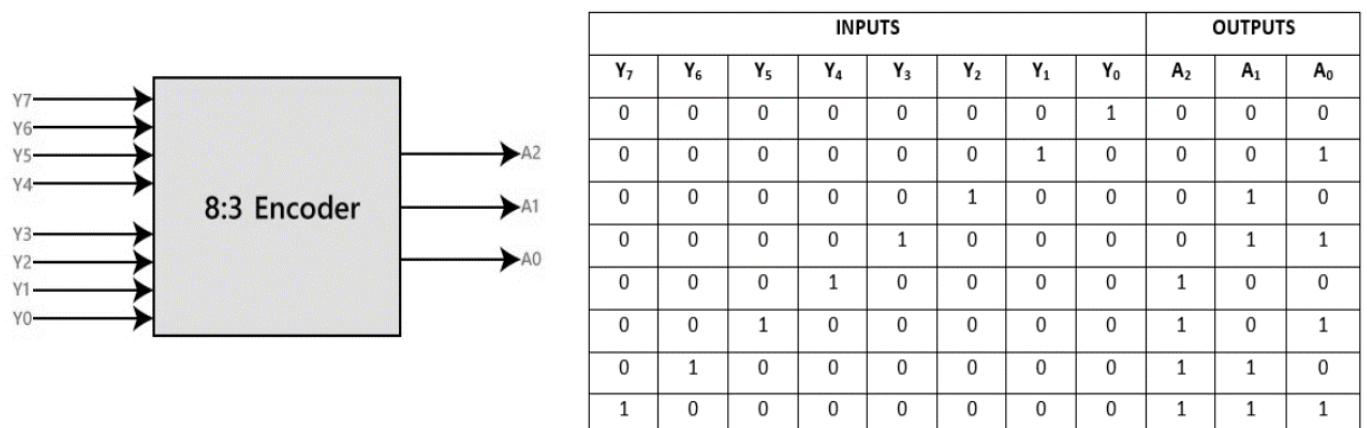


Fig. 3.22 Block diagram and truth table of a 3-to-3-line Encoder

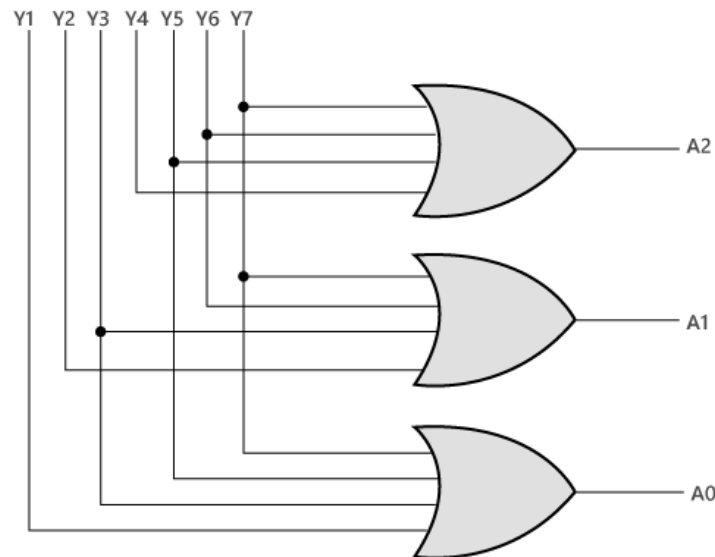
The logical expression of the term A_0 , A_1 , and A_2 are as follows:

$$A_2 = Y_4 + Y_5 + Y_6 + Y_7$$

$$A_1 = Y_2 + Y_3 + Y_6 + Y_7$$

$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

Logical circuit of the above expressions is given below:



3.6 MULTIPLEXERS (MUX)

A multiplexer is a combinational circuit that has 2^n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from many (2^n) input lines and directed to one output line. On the basis of the values of the selection lines (n), one of these data inputs will be connected to the output.

Unlike encoder and decoder, there are n -selection lines and 2^n input lines. So, there is a total of 2^N possible combinations of inputs. A multiplexer is also called as Mux. There are various types of the multiplexer which are as follows.

2×1 Multiplexer

In 2×1 multiplexer, there are only two inputs, i.e., A_0 and A_1 , one selection line, i.e., S and single outputs, i.e., Y . On the basis of the combination of inputs which are present at the selection line S , one of these two inputs will be connected to the output. The block diagram and the truth table of the 2×1 multiplexer is given below.

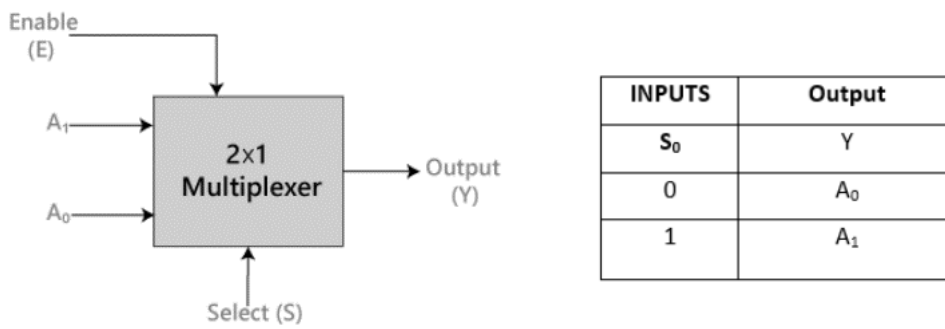


Fig. 3.23 Block diagram and truth table of 2X1 mux

The logical expression of the term Y is as follows:

$$Y = S'.A_0 + S.A_1$$

Logical circuit of the above expression is given below Fig. 3.24.

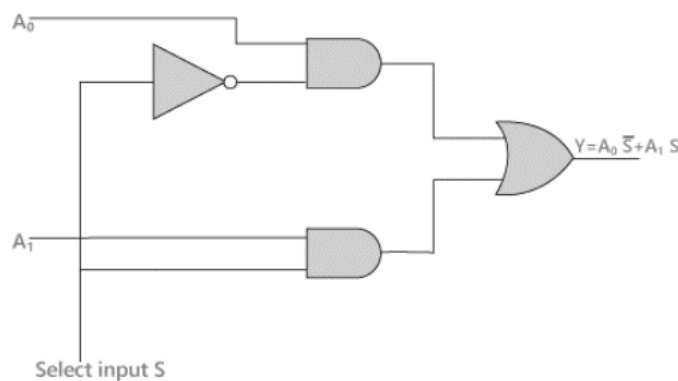


Fig. 3.24 2-to-1 Mux Logic Diagram

4x1 Multiplexer

In the 4x1 multiplexer, there is a total of four inputs, i.e., A₀, A₁, A₂, and A₃, two selection lines, i.e., S₀ and S₁ and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S₀ and S₁, one of these four inputs are connected to the output.

The block diagram and the truth table of the 4x1 multiplexer is given below fig 3.25.

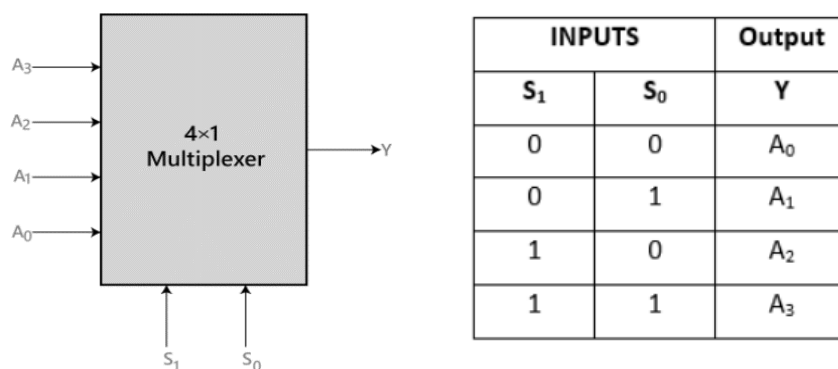


Fig. 3.25 Block diagram and truth table of 4X1 mux

The logical expression of the term Y is as follows:

$$Y = S_1' S_0' A_0 + S_1' S_0 A_1 + S_1 S_0' A_2 + S_1 S_0 A_3$$

Logical circuit of the above expression is given below Fig. 3.26.

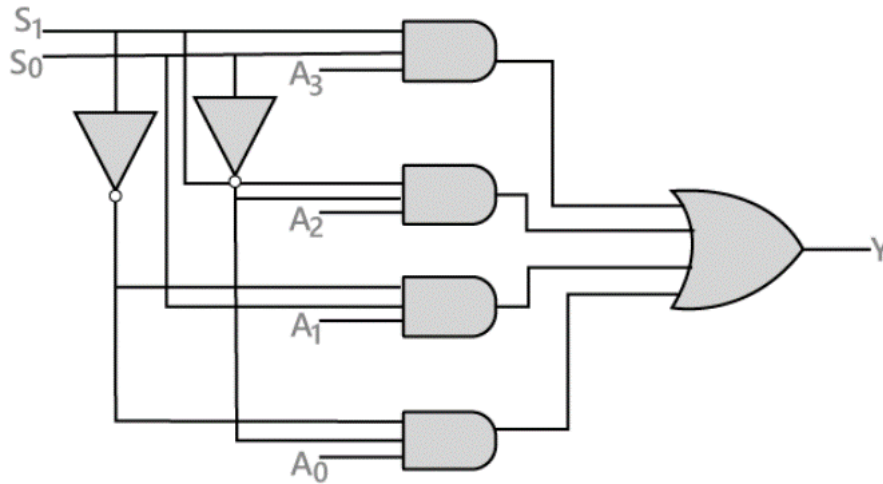


Fig. 3.26 4X1 Mux Logic diagram

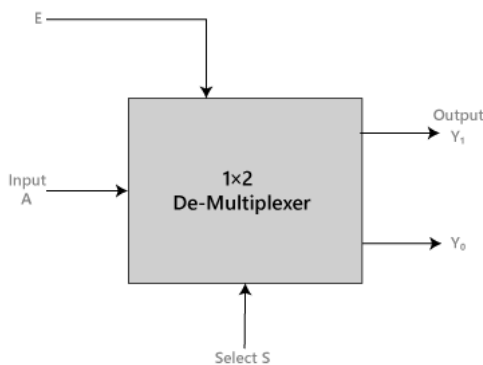
3.7 DE-MULTIPLEXERS (DE-MUX)

A De-multiplexer is a combinational circuit that has only one input line and 2^N output lines. Simply, the de-multiplexer is a single-input and multi-output combinational circuit. The information is received from the single input lines and directed to the one of many output lines. On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

Unlike encoder and decoder, there are n-selection lines and 2^n outputs. So, there is a total of 2^n possible combinations of inputs. De-multiplexer is also treated as De-mux. There are various types of De-multiplexer which are as follows.

1x2 De-multiplexer

In the 1-to-2 De-multiplexer, there are only two outputs, i.e., Y_0 , and Y_1 , one selection line, i.e., S, and single input, i.e., A. On the basis of the selection value, the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 de-multiplexer is given below.



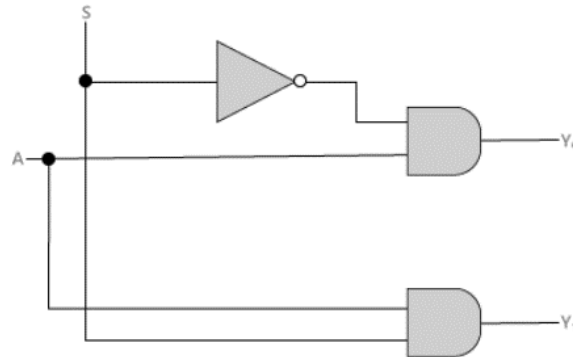
INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

The logical expression of the outputs Y_0 and Y_1 is as follows:

$$Y_0 = S'.A$$

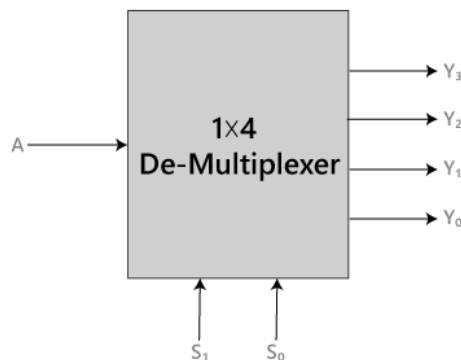
$$Y_1 = S.A$$

Logical circuit of the above expressions is given below:



1x4 De-multiplexer

In 1-to-4 De-multiplexer, there are total of four outputs, i.e., Y_0 , Y_1 , Y_2 , and Y_3 , two selection lines, i.e., S_0 and S_1 and single input, i.e., A . On the basis of the combination of inputs which are present at the selection lines S_0 and S_1 , the input be connected to one of the outputs. The block diagram and the truth table of the 1x4 multiplexer is given below.



INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

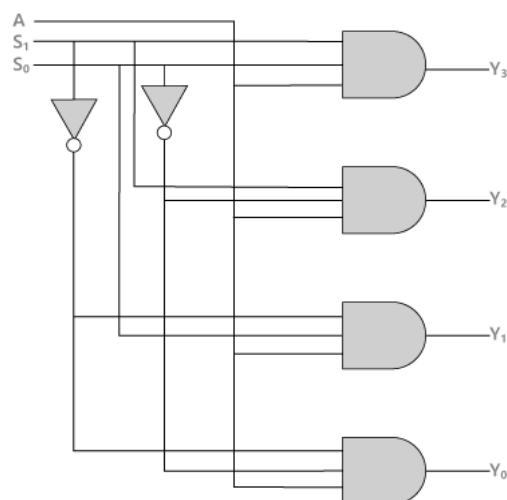
The logical expression of the terms Y and Logical circuit of the expressions is given below:

$$Y_0 = S_1'S_0'.A$$

$$Y_1 = S_1'S_0.A$$

$$Y_2 = S_1S_0'.A$$

$$Y_3 = S_1S_0.A$$



3.8 MAGNITUDE COMPARATOR

A magnitude comparator is a combinational circuit designed primarily to compare the relative magnitude of the two binary numbers A and B. Naturally, the result of this comparison is specified by three binary variables that indicate, whether $A > B$, $A = B$ or $A < B$. The block diagram of a single bit magnitude comparator is shown in Fig. 3.27.

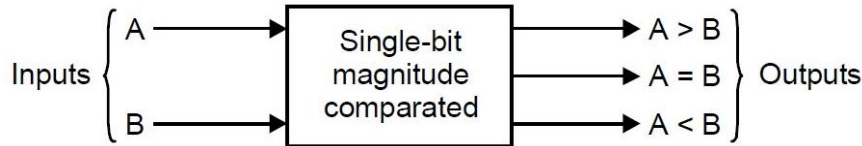


Fig. 3.27 Block diagram of a single bit comparator

To implement the magnitude comparator the properties of Ex-NOR gate and AND gate is used. Fig. 3.28(a) shows an EX-NOR gate with two inputs A and B. If $A = B$ then the output of Ex-NOR gate is equal to 1 otherwise 0.

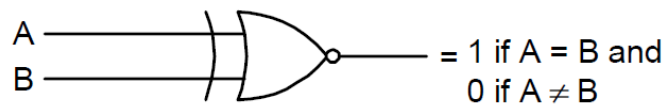


Fig. 3.28(a)

Fig. 3.28 (b) and (c) shows AND gates, one with A and B' as inputs and another with A' and B as their inputs.

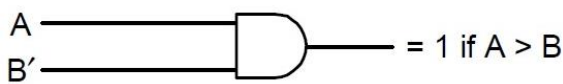


Fig. 3.28 (b)

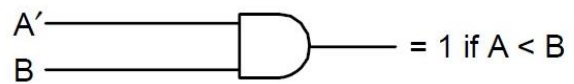


Fig. 3.28 (c)

The AND gate output of 3.28(b) is 1 if $A > B$ (i.e., $A = 1$ and $B = 0$) and 0 if $A < B$ (i.e., $A = 0$ and $B = 1$). Similarly, the AND gate output of 3.28(c) is 1 if $A < B$ (i.e., $A = 0$ and $B = 1$) and 0 if $A > B$ (i.e., $A = 1$ and $B = 0$). If the EX-NOR gate and two AND gates are combined as shown in Fig. 3.29, the circuit with function as single bit magnitude comparator. For EX-NOR implementation We have used EX-OR followed by an inverter.

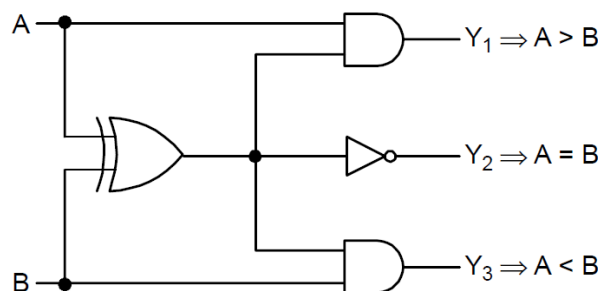


Fig. 3.29 Single bit magnitude comparator

Truth table of a single bit magnitude comparator.

Inputs		Output		
A	B	Y_1	Y_2	Y_3
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

It clearly shows, Y_1 is high when $A > B$.

Y_2 is high when $A = B$.

Y_3 is high when $A < B$.

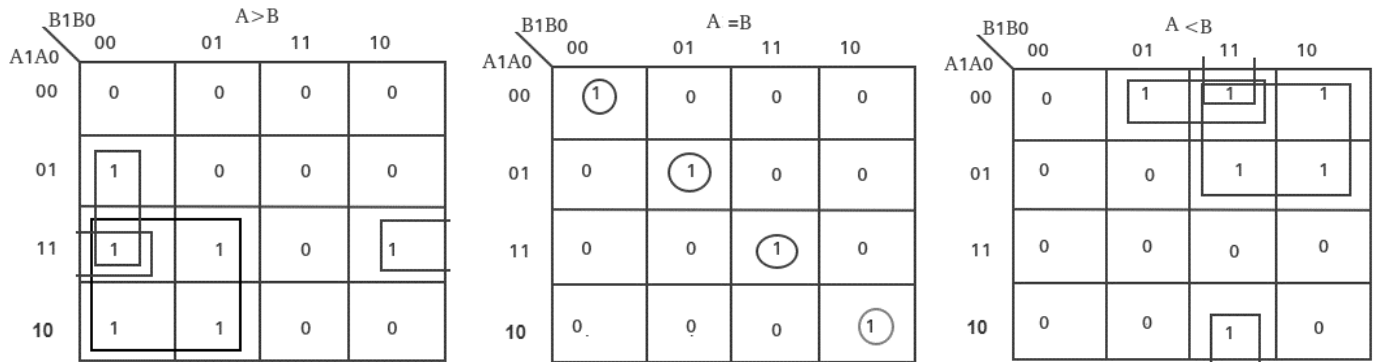
2-bit Magnitude Comparator

A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

The truth table for a 2-bit comparator is given below:

INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

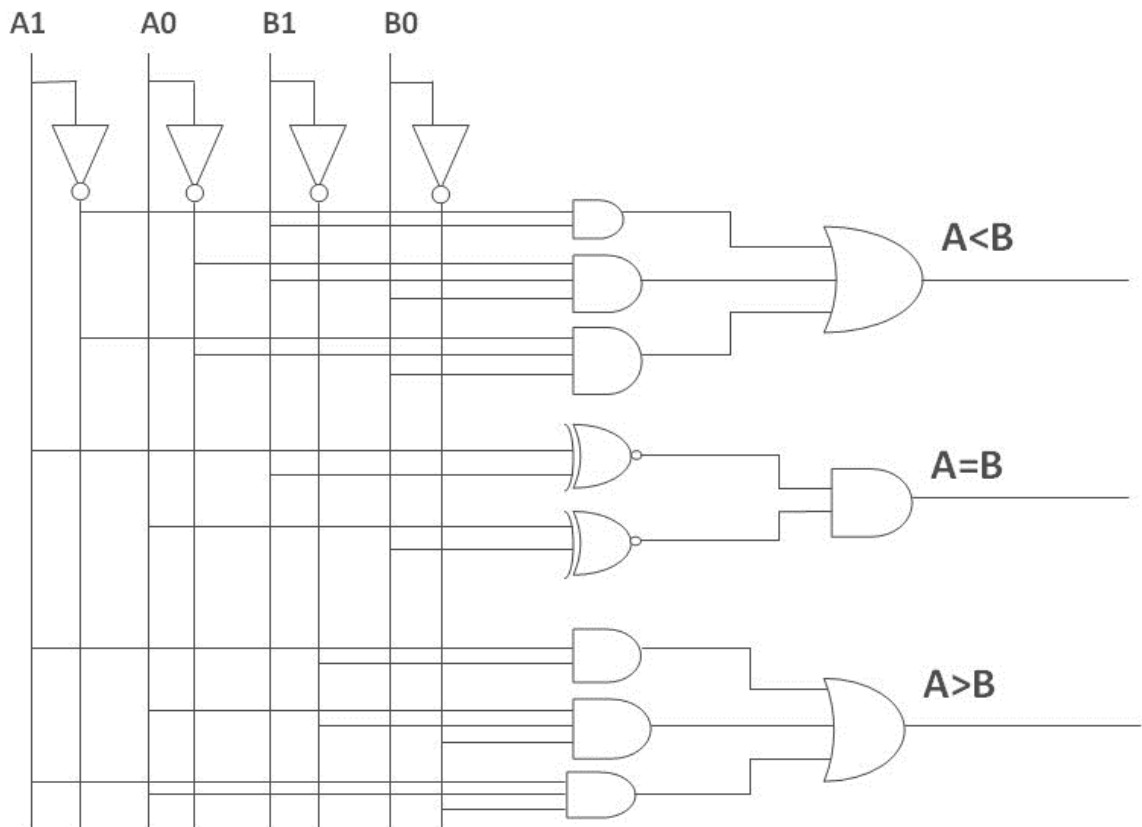
From the above truth table K-map for each output can be drawn as follows:



From the above K-maps logical expressions for each output can be expressed as follows:

$$\begin{aligned}
 A > B &: A1B1' + A0B1'B0' + A1A0B0' \\
 A = B &: A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0' \\
 &: A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0') \\
 &: (A0B0 + A0'B0') (A1B1 + A1'B1') \\
 &: (A0 \text{ Ex-Nor } B0) (A1 \text{ Ex-Nor } B1) \\
 A < B &: A1'B1 + A0'B1B0 + A1'A0'B0
 \end{aligned}$$

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



3.9 IC LOGIC FAMILIES

Definition

A logic family is a collection of different integrated circuit chips that have similar input, output, and internal circuit characteristics, but they perform different logic gate functions such as AND, OR, NOT, etc.

Characteristics of logic families

The main characteristics of Logic families include:

1. Speed
2. Fan-in
3. Fan-out
4. Noise Immunity
5. Power Dissipation

Speed: Speed of a logic circuit is determined by the time between the application of input and change in the output of the circuit.

Fan-in: It determines the number of inputs the logic gate can handle.

Fan-out: Determines the number of circuits that a gate can drive.

Noise Immunity: Maximum noise that a circuit can withstand without affecting the output.

Power Dissipation: When a circuit switches from one state to the other, power dissipates.

Types of Logic Families

Resistor-Transistor Logic (RTL) – it uses resistors and transistors to implement logic gates.

Diode- Transistor Logic (DTL) – it uses diodes and transistors to implement logic gates.

Transistor- Transistor Logic (TTL) – it uses transistors to implement logic gates.

Complementary metal oxide semiconductor Logic (CMOS) – is uses both PMOS and NMOS FETs to implement logic gates.

3.9.1 NAND and NOR gates using RTL Logic

NAND Gate:

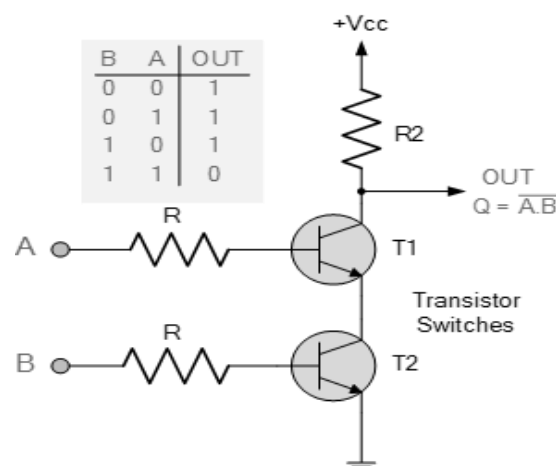
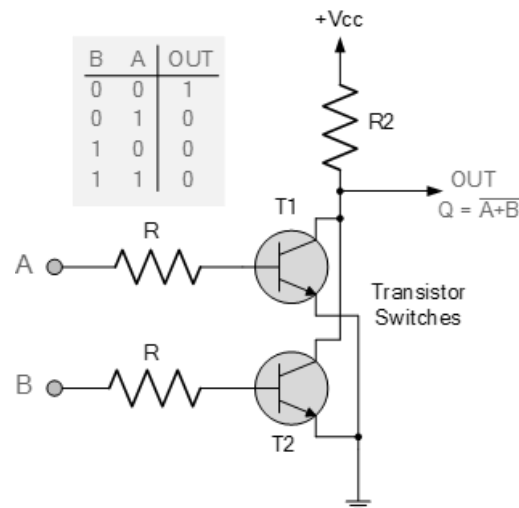


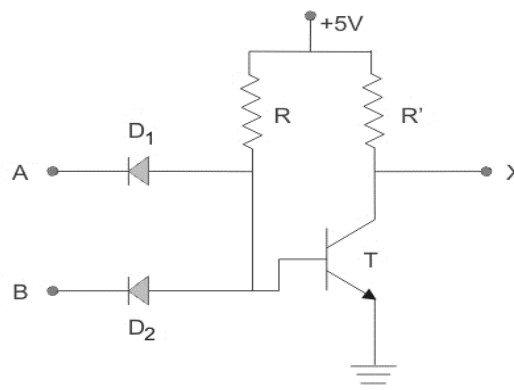
Fig. 3.30: RTL NAND Gate

Operation:

1. If any one of the input or both inputs are at low, either of the transistors T1 or T2 are in OFF state and there is no connection between output and ground. So, the output is at high state i.e., at logic 1.
2. If both inputs are at high state, then both the transistors are in ON state there by connecting the output to ground, hence the output is low i.e., at logic 0.

NOR Gate:**Fig. 3.31: RTL NOR Gate****Operation:**

1. If any one of the input or both inputs are at high state, the transistors T1 or T2 or both the transistors are in ON state there by connecting the output to ground, hence the output is at low state i.e., at logic 0.
2. When both inputs are at low state, both transistors are in OFF state and there is no connection between output and ground, hence the output is connected to VCC i.e., at high state (Logic 1).

3.9.2 NAND and NOR gates using DTL Logic**NAND Gate:****Fig. 3.32: DTL NAND Gate**

Operation:

1. If any one of the inputs is at LOW, the corresponding diode is in forward biased and input voltage to the transistor is LOW, hence the transistor is in OFF state and the output is at HIGH i.e., at logic 1.
2. If both inputs are at HIGH, both the diodes are in reverse bias and the input voltage to the base of the transistor is HIGH there by making the transistor into ON state, hence the output is at LOW state i.e., at logic 0.

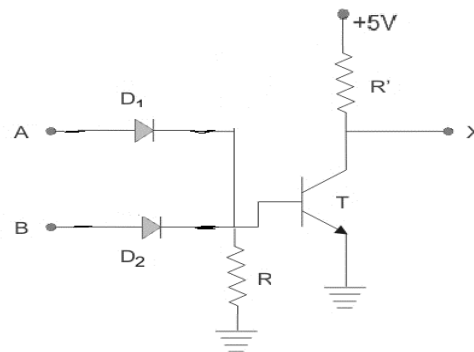
NOR Gate:

Fig. 3.33: DTL NOR Gate

Operation:

1. If any one of the inputs is at HIGH state, one of the diodes is in forward bias, input voltage to the base of the transistor is HIGH, hence the transistor is in ON state there by making the output is at LOW state i.e., at logic 0.
2. If both the inputs are at LOW, both diodes are in reverse bias, the input voltage to the base of the transistor is LOW, hence the transistor is at OFF state there by making the output at HIGH state i.e., at logic 1.

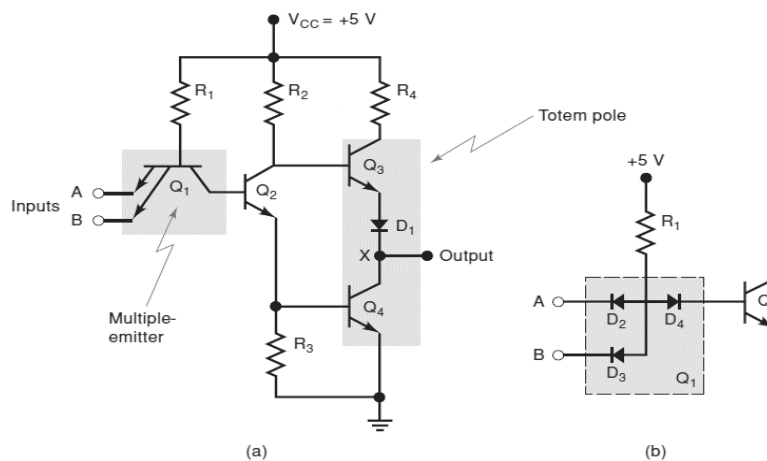
3.9.3 NAND and NOR gates using TTL Logic**NAND Gate:**

Fig. 3.34: (a) Basic TTL NAND gate; (b) diode equivalent for Q1.

Operation:

1. When either of the inputs or both inputs are at LOW, the transistor Q1 is in ON state. The total VCC is connected to ground through ON transistor Q1 and resistor R1. The transistor Q2 is in OFF state thereby making transistor Q3 in ON state and Q4 in OFF state. The output is connected to VCC through the ON transistor Q3 and resistor R4. Hence the output is at HIGH state i.e., at logic 1.
2. When both inputs are at HIGH state, the transistor Q1 is in OFF state. The applied voltage VCC appeared at the base of the Q2, making it in ON state. The transistor Q4 is in ON state and transistor Q3 is in OFF state. Hence the output is connected to ground by making the output is at low state i.e., at logic 0.

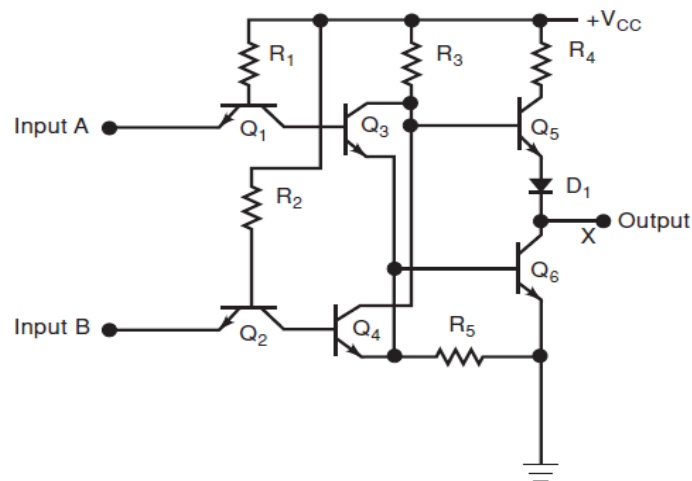
NOR Gate:

Fig. 3.35: TTL NOR gate.

Operation:

1. When either of the input or both inputs are at HIGH state, both the transistors Q1 or Q2 is in OFF state. The transistors Q3 or Q4 is in ON state. The transistors Q5 is in OFF state and Q6 is in ON state thereby connecting the output to ground, hence the output is at low state i.e., at logic 0.
2. When both inputs are at LOW state, the transistor Q1 and Q2 is in ON state. The transistors Q3 and Q4 are in OFF state. The transistor Q5 is in ON state thereby connecting the VCC to output, hence the output is at HIGH state i.e., at logic 1.

3.9.4 NAND and NOR gates using CMOS logic

NAND Gate:

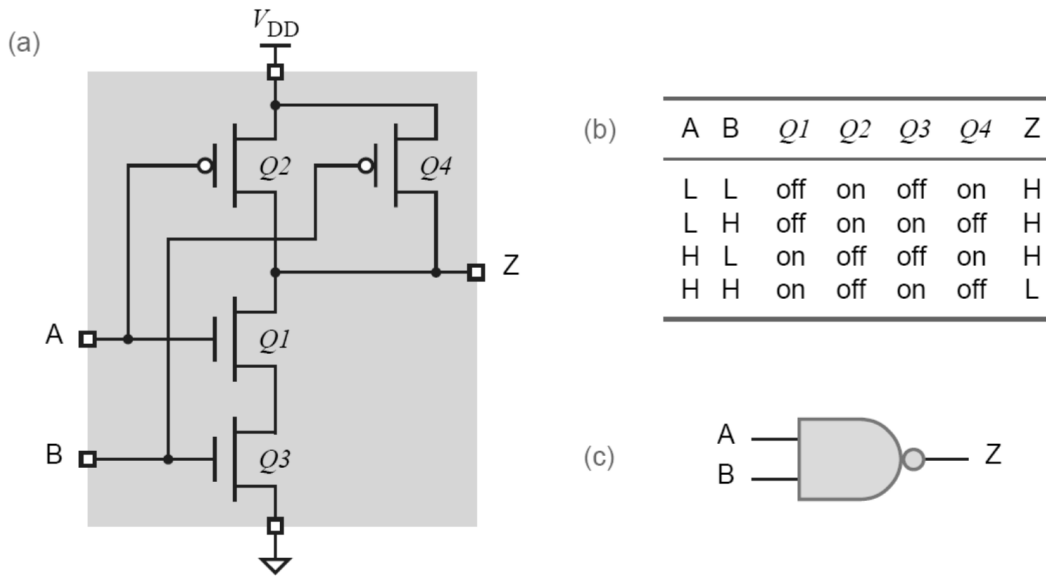


Fig. 3.36: (a) CMOS NAND gate, (b) Functional table, (c) logic symbol

1. Fig.3.36 shows a 2-input CMOS NAND gate, where if either input is LOW, the output Z has a low-impedance connection to V_{DD} through the corresponding “on” p-channel transistor, and the path to ground is blocked by the corresponding “off” n-channel transistor. Hence the output is at HIGH state i.e., at logic 1.
2. If both inputs are HIGH, the path to V_{DD} is blocked through the “off” p-channel transistors, and Z has a low-impedance connection to ground through the “on” n-channel transistors. Hence the output is at LOW state i.e., at logic 0.

NOR Gate:

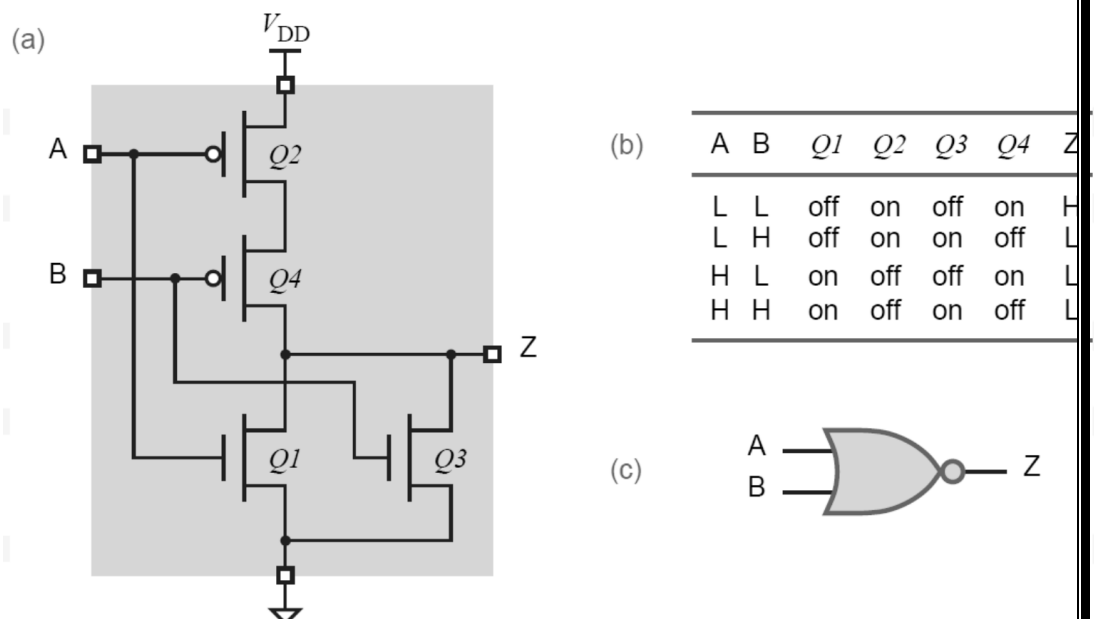
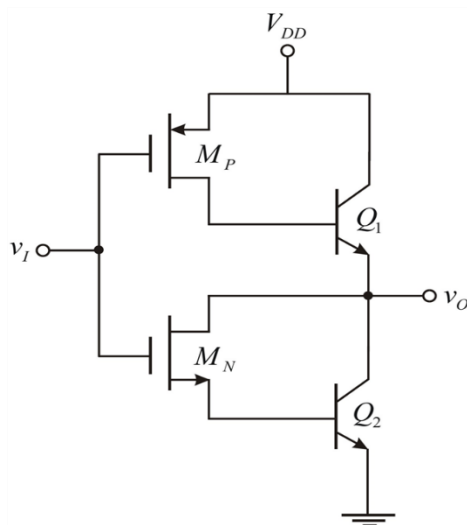


Fig. 3.37: (a) CMOS NOR gate, (b) Functional table, (c) logic symbol

1. If both inputs are LOW, the output Z has a low-impedance connection to VDD through the “on” p-channel transistors, and the path to ground is blocked by the “off” n-channel transistors. Hence the output is at HIGH state i.e., at logic 1.
2. If either input is HIGH, the path to VDD and output is blocked through the “off” p-channel transistor and Z has a low-impedance connection to ground through the “on” n-channel transistor, hence the output is at low state i.e., at logic 0.

3.9.5 Bi-CMOS Inverter



Functional Table

Input Voltage	Transistor states				Output
	M _P	M _N	Q1	Q2	
0V	ON	OFF	ON	OFF	V _{DD} (5V)
5V	OFF	ON	OFF	ON	GND (0V)

Fig. 3.38: Bi-CMOS Inverter

1. BICMOS logic circuits are made by combining the CMOS and bipolar IC technologies. These ICs combine the advantages of BJT and CMOS transistors in them. We know that the speed of BJT is very high compared to CMOS. However, power dissipation in CMOS is extremely low compared to BJT.
2. In fig. 3.38, we see that transistors M_P and M_N from the CMOS inverter.
3. When input voltage V₁ = 0V, the PMOS will conduct and the NMOS will remain OFF. This drives a current through the base of the bipolar junction transistor Q1 and turns it ON. Hence the output V₀ is connected to the V_{DD} through the ON transistor Q1 by making the output is at high state i.e., at logic 1.
4. When input voltage V₁ = 5V, the NMOS will conduct and the PMOS will remain OFF. This drives a current through the base of the bipolar junction transistor Q2 and turns it ON. Hence the output V₀ is connected to the GND through the ON transistor Q2 by making the output is at low state i.e., at logic 0.

UNIT – 4**SEQUENTIAL DIGITAL CIRCUITS****4.1 INTRODUCTION**

In our previous unit, we learned about combinational circuit and their working. The combinational circuits have set of outputs, which depends only on the present combination of inputs. Below is the block diagram of the sequential logic circuit.

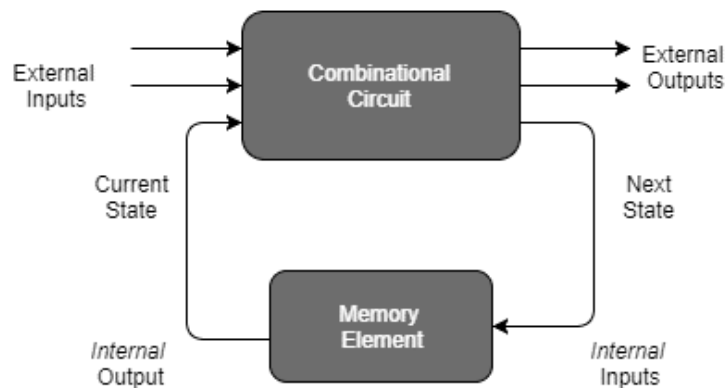


Fig. 4.1 Block diagram of sequential logic circuit

The sequential circuit is a special type of circuit that has a series of inputs and outputs. The outputs of the sequential circuits depend on both the combination of present inputs and previous outputs. The previous output is treated as the present state. So, the sequential circuit contains the combinational circuit and memory storage elements.

Difference between the combinational circuits and sequential circuits are given below:

S.No	Combinational Circuits	Sequential Circuits
1.	The outputs of the combinational circuit depend only on the present inputs.	The outputs of the sequential circuits depend on both present inputs and present state(previous output).
2.	The feedback path is not present in the combinational circuit.	The feedback path is present in the sequential circuits.
3.	In combinational circuits, memory elements are not required.	In the sequential circuit, memory elements play an important role and require.
4.	The clock signal is not required for combinational circuits.	The clock signal is required for sequential circuits.
5.	The combinational circuit is simple to design.	It is not simple to design a sequential circuit.

Types of Sequential Circuits

Asynchronous sequential circuits:

If some or all the outputs of a sequential circuit do not change (affect) with respect to active transition of clock signal, then that sequential circuit is called as Asynchronous sequential circuit. That means, all the outputs of asynchronous sequential circuits do not change (affect) at the same time. Therefore, most of the outputs of asynchronous sequential circuits are not in synchronous with either positive edges or negative edges of clock signal.

Synchronous sequential circuits:

If all the outputs of a sequential circuit change (affect) with respect to active transition of clock signal, then that sequential circuit is called as Synchronous sequential circuit. That means, all the outputs of synchronous sequential circuits change (affect) at the same time. Therefore, the outputs of synchronous sequential circuits are in synchronous with either positive edges or negative edges of clock signal.

Clock Signal and Triggering

Clock signal:

A clock signal is a periodic signal in which ON time and OFF time need not be the same. When ON time and OFF time of the clock signal are the same, a square wave is used to represent the clock signal. Below is a diagram which represents the clock signal

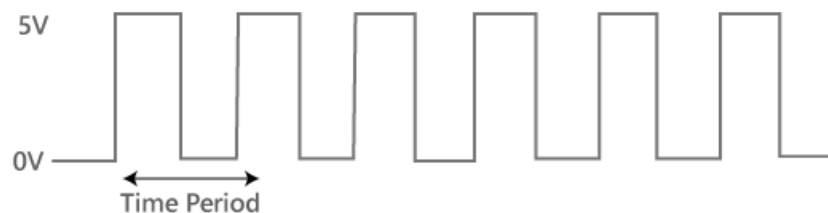


Fig. 4.2 Clock Signal

Types of Triggering

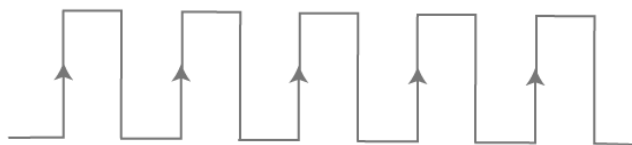
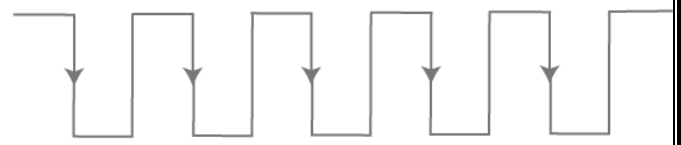
These are two types of triggering in sequential circuits:

Level triggering:

The logic High and logic Low are the two levels in the clock signal. In level triggering, when the clock pulse is at a particular level, then only the circuit is activated. There are the following types of level triggering

Positive level triggering**Negative level triggering****Fig. 4.3 Positive and negative level clock signal****Edge triggering:**

In clock signal of edge triggering, two types of transitions occur, i.e., transition either from Logic Low to Logic High (positive edge) or Logic High to Logic Low (negative edge). when the clock pulse is at a particular edge, then only the circuit is activated. Based on the transitions of the clock signal, there are the following types of edge triggering.

Positive edge triggering**Negative edge triggering****Fig. 4.4 Positive and negative edge clock signal****4.2 FLIP FLOPS**

Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip flop is said to be edge sensitive or edge triggered circuit. A flip flop has two outputs Q and Q' which are complement to each other. When the value of Q = 1, it is known as the set state of the flip flop in this state it stores a logic 1. When the value of Q = 0, it is known as the reset state of the flip flop in this state it stores a logic 0.

4.2.1 SR Flip flop

RS flip-flop is the simplest possible memory element. It can be constructed from two NAND gates or two NOR gates. Let us understand the operation of the RS flip – flop using NOR gates as shown below using the truth table for 'A NOR B' gate. The inputs R and S are referred as Reset and Set respectively. The outputs Q and Q' are complemented to each other and are referred to as normal and complement outputs respectively. The binary state of the flip flop is taken to be the normal value of the output. When Q=1 and Q'=0, it is in the set state (or 1-state). When Q=0 and Q'=1, it is in the reset/clear state (or 0-state).

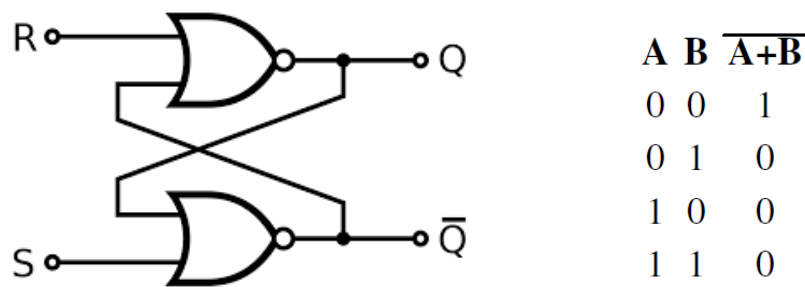
Circuit diagram:

Fig. 4.5 SR flip flop using NOR gates and NOR gate truth table.

Operation:

1. $S=1$ and $R=0$: The output of the bottom NOR gate is equal to zero, $Q'=0$. Hence both inputs to the top NOR gate are equal to 0, thus, $Q=1$. Hence, the input combination $S=1$ and $R=0$ leads to the flip-flop being set to $Q=1$.
2. $S=0$ and $R=1$: The output of the top NOR gate is equal to zero, $Q=0$. Hence both inputs to the bottom NOR gate are equal to 0, thus, $Q'=1$. Hence, the input combination $S=0$ and $R=1$ leads to the flip-flop being reset to $Q=0$.
3. $S=0$ and $R=0$: Assume the flip-flop was previously in set ($Q=1$ and $Q'=0$) condition. Now changing S to 0 results the output of bottom NOR gate to 0 ($Q'=0$). The output of top NOR gate to 1 ($Q=1$). This means the output do not change. Therefore, with inputs $S = R = 0$, the flip flop holds its last state.
4. $S=1$ and $R=1$: This condition violates the fact that both outputs are complements of each other since each of them tries to go to 0, which is not a stable configuration. It is impossible to predict which output will go to 1 and which will stay at 0. In normal operation this condition must be avoided by making sure that 1's are not applied to both inputs simultaneously, thus making it one of the main disadvantages of SR flip flop.

Truth table:

R	S	Q	Q'	State
0	0	Q	Q'	Hold state (No Change)
0	1	1	0	Set
1	0	0	1	Reset
1	1	X	X	Not allowed

Clocked SR flip flop:

A simple way to get a clocked SR flip-flop is to AND the inputs signals with clock and then apply them to S and R inputs of flip-flop as shown in fig. 4.6. For the simplicity SET and RESET inputs of un-clocked SR latch are labelled S1 and R1 respectively. Whereas external inputs are labelled S and R.

When clock (abbreviated as CLK) is LOW, outputs of gates G1 and G2 are forced to 0, which is applied to flip-flop inputs. Thus, when $CLK = 0$, $S1 = 0$ and $R1 = 0$. Since both the inputs are LOW, flip-flop remain in its previous state. Alternatively, if $Q_n = 0$, $Q_{n+1} = 0$ and if $Q_n = 1$ we get $Q_{n+1} = 1$. This is shown in first two rows of truth table given in Fig. 4.7.

When clock is HIGH, gates G1 and G2 are transparent and signals S and R can reach to flip-flop inputs S1 and R1. The next state will now be determined by the values of S and R. Thus, when $CLK = 1$ causing $S1 = S$ and $R1 = R$ and the circuit behaves exactly same as the normal flip-flop discussed in subsection 4.2.1, as shown by rest of the rows of truth table.

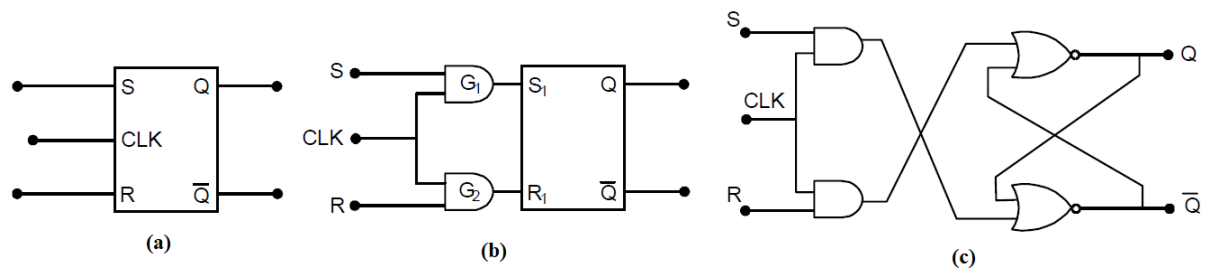


Fig. 4.6 (a)SR flip flop logic symbol, (b) SR flip flop using un-clocked SR latch, (c) SR flip flop logic diagram.

CLK	Q_n	S_n	R_n	Q_{n+1}	Comments
0	0	x	x	0	Flip-Flop disabled no change in state and last state maintained
0	1	x	x	1	
1	0	0	0	0	Last State $Q_{n+1}=Q_n$
1	0	0	1	0	Reset state
1	0	1	0	1	Set state
1	0	1	1	?	Indeterminate
1	1	0	0	1	Last state $Q_{n+1}=Q_n$
1	1	0	1	0	Reset state
1	1	1	0	1	Set state
1	1	1	1	?	Indeterminate

Fig. 4.7 SR flip flop truth table.

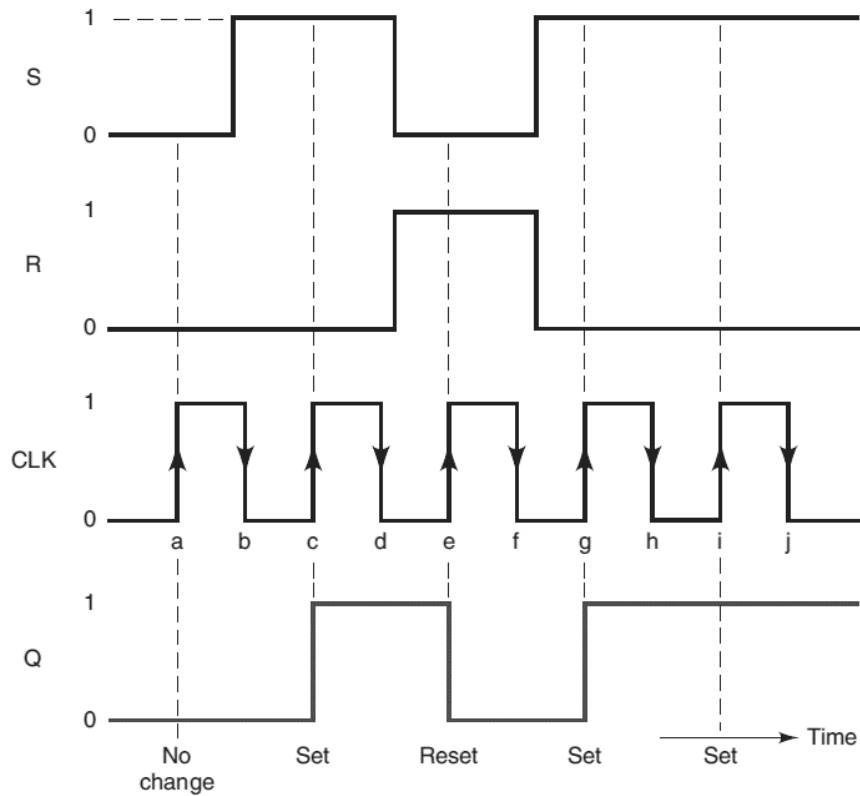


Fig. 4.8 SR flip flop timing diagram.

4.2.2 D – flip flop

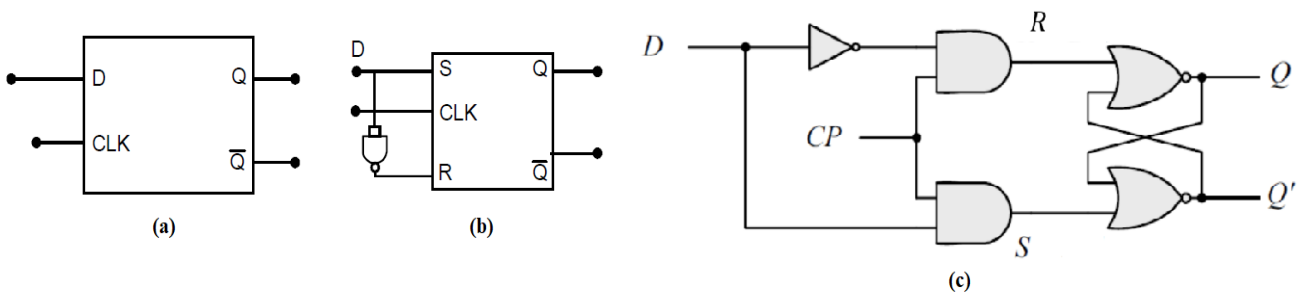


Fig. 4.9 (a) D-flip flop logic symbol, (b) D-flip flop using SR FF, (c) Logic diagram

A clocked D flip-flop is obtained by using an external inverter at the input of clocked SR flip-flop. The clocked D flip-flop is shown below in Fig. 4.9.

An RS flip-flop is rarely used in actual sequential logic because of its undefined outputs for inputs $R = S = 1$. It can be modified to form a more useful circuit called D flip-flop, where D stands for data. The D flip-flop has only a single data input D as shown in the circuit diagram. That data input is connected to the S input of an RS flip-flop, while the inverse of D is connected to the R input. To allow the flip-flop to be in a holding state, a D-flip flop has a second input called clock, CLK. The clock input is AND-ed with the D-input.

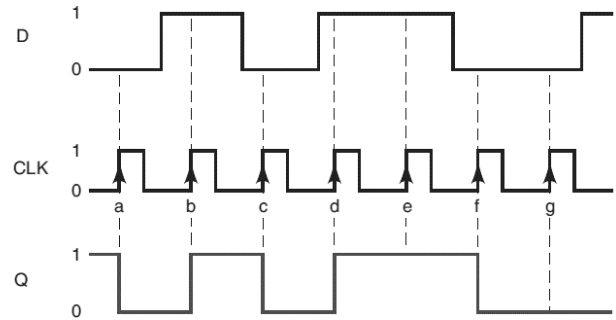
When CLK=0, irrespective of D-input, the R = S = 0 and the state is held.

When CLK= 1, the S input of the RS flip-flop equals the D input and R is the inverse of D. Hence, output Q follows D, when EN= 1.

When CLK returns to 0, the most recent input D is 'remembered'.

CLK	Q _n	D _n	Q _{n+1}	Comments
0	0	x	0	Flip-Flop Disabled last state Maintained
0	1	x	1	
1	0	0	0	Rest State
1	0	1	1	Set State
1	1	0	0	Reset State
1	1	1	1	Set State

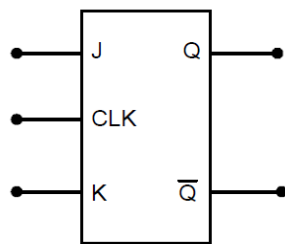
(a)



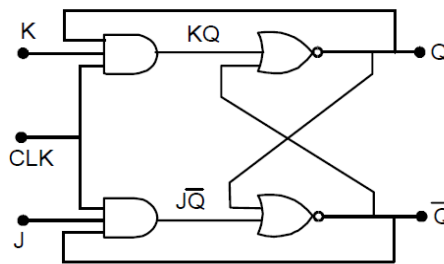
(b)

Fig. 4.10 (a) D-flip flop truth table, (b) D-flip flop timing diagram.

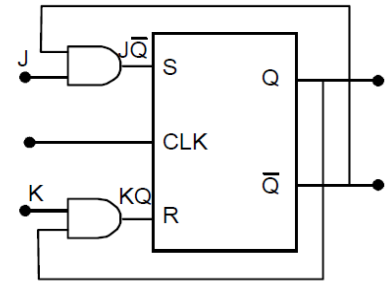
4.2.3 JK – Flip flop



(a)



(b)



(c)

Fig. 4.11 (a) JK-flip flop logic symbol, (b) JK-flip flop logic diagram, (c) JK- flip flop using SR flip flop.

Problem of SR flip-flop to lead to indeterminate state when S = R = 1, is eliminated in JK flip-flops. A simple realization of JK flip-flop by modifying the SR type is shown in Fig. 4.11 (c). In JK the indeterminate state of SR flip-flop is now modified and is defined as TOGGLED STATE (i.e., its own complement state) when both the inputs are HIGH. Table shown in Fig. 4.12(a), summarizes the operation of JK flip-flop for all types of input possibilities.

Operation:

1. When J = K = 0 and Q = 1, Q' = 0.

The outputs of top and bottom AND gates are zero, which are the inputs to an SR flip flop as shown in Fig. 4.11(c). SR flip flop with both inputs S = R = 0 is known as no change condition so the outputs remain in same state. Hence, in a JK flip flop J = K = 0 is known as no change condition.

- When $J = 1, K = 0$ and $Q = 0, Q' = 1$.

The outputs of top AND gate is one and bottom AND gate is zero, which are the inputs to an SR flip flop as shown in Fig. 4.11(c). SR flip flop with both inputs $S = 1, R = 0$ is known as set condition so the output Q changes to 1 and $Q' = 0$. Hence, in a JK flip flop $J = 1, K = 0$ is known as set condition.

- When $J = 0, K = 1$ and $Q = 1, Q' = 0$.

The outputs of top AND gate is zero and bottom AND gate is one, which are the inputs to an SR flip flop as shown in Fig. 4.11(c). SR flip flop with both inputs $S = 0, R = 1$ is known as reset condition so the output Q changes to 0 and $Q' = 1$. Hence, in a JK flip flop $J = 0, K = 1$ is known as reset condition.

- When $J = K = 1$ and $Q = 1, Q' = 0$.

The outputs of top AND gate is zero and bottom AND gate is one, which are the inputs to an SR flip flop as shown in Fig. 4.11(c). SR flip flop with both inputs $S = 0, R = 1$ is known as reset condition so the output Q changes to 1 and $Q' = 0$. Similarly, if $Q = 0$ and $Q' = 1$ and maintaining $J = K = 1$, the inputs to the SR flip flop changes to $S = 1$ and $R = 0$, which in turn changes the output $Q = 1$ and $Q' = 0$. Hence, in a JK flip flop $J = 1, K = 1$ is known as toggle condition.

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1(Toggle, \bar{Q}_n)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0(Toggle, \bar{Q}_n)

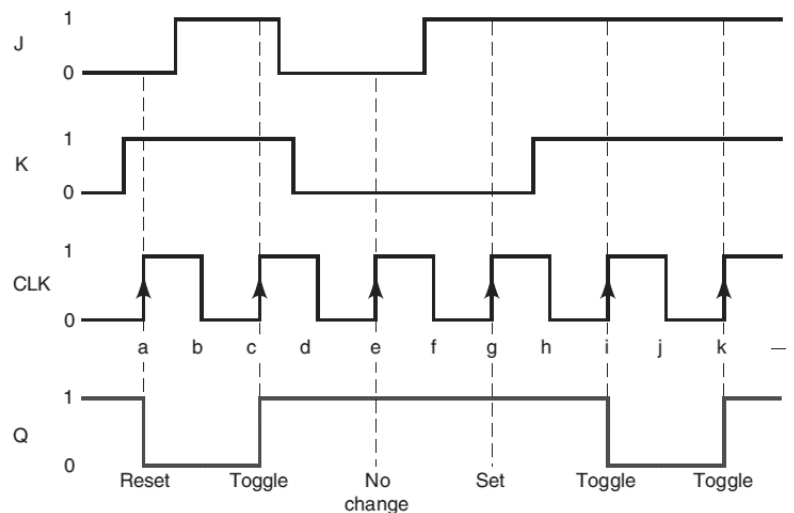


Fig. 4.12 (a) JK-flip flop truth table, (b) JK-flip flop timing diagram.

4.2.4 T – Flip flop:

In JK flip flop, due to the toggling capability, when both inputs HIGH, on each arrival of pulse, it forms the basic element for counters. For this purpose, JK flip-flop is further modified to provide a T flip-flop as shown in Fig. 4.13. T flip-flop is a single input version of JK flip-flop, in which the inputs J and K are connected together, as shown, and is provided as a single input labelled as T.

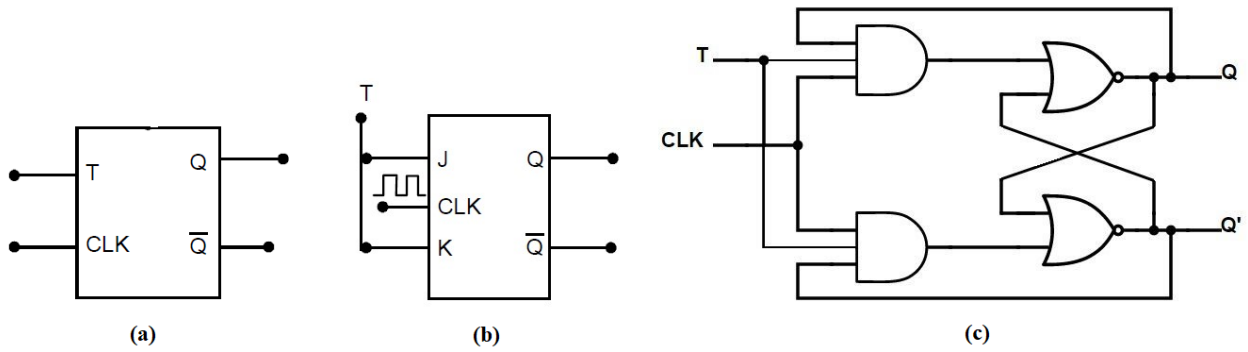


Fig. 4.13 (a) T-flip flop logic symbol, (b) T-flip flop using JK flip flop, (c) T- flip flop circuit.

Operation:

When the clock is absent, the flip-flop is disable as usual and previously latched output is maintained at output. When the clock is present and $T = 0$, even though flip-flop is enabled the output does not switch its state. It happens so because for $T = 0$ we get $J = K = 0$ and thus next state is same as present state. Thus, if either $CLK = 0$ or $T = 0$, state does not change next state is always same as present state.

When $T = 1$ during $CLK = 1$, it causes $J = K = 1$ and as earlier discussed it will toggle the output state. Thus, when input T is HIGH, flip-flop toggles its output on the arrival of clock, and for this reason input T is called as the Toggle Input.

CLK	Present States		Applied Inputs			Next state Out-put		Resulting state
	Q_n	\bar{Q}_n	T_n	J_n	K_n	Q_{n+1}		
0	0	1	x	x	x	0	Q_n	Flip-Flop Disabled for $CLK=0$, no change next state=present
0	1	0	x	x	x	1		
1	0	1	0	0	0	0	Q_n	Maintain Out-put state no change in out-put next state=present state
1	1	0	0	0	0	1		
1	0	1	1	1	1	1	\bar{Q}_n	Toggled state or complemented state next state =present state
1	1	0	1	1	1	0		

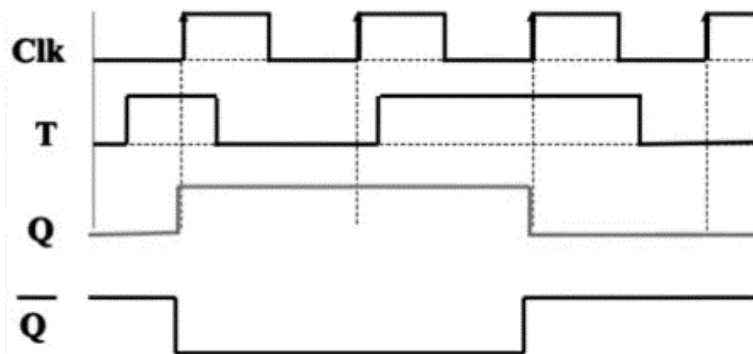


Fig. 4.14 T-flip flop truth table and timing diagram.

4.2.5 Master slave JK flip flop:

In "JK Flip Flop", when both the inputs and CLK set to 1 for a long time, then Q output toggle until the CLK is 1. Thus, the uncertain or unreliable output produces. This problem is referred to as a **race-round condition** in JK flip-flop and avoided by ensuring that the CLK set to 1 only for a very short time.

Construction:

The master-slave flip flop is constructed by combining two JK flip flops. These flip flops are connected in a series configuration. In these two flip flops, the 1st flip flop work as "master", called the master flip flop, and the 2nd work as a "slave", called slave flip flop. The master-slave flip flop is designed in such a way that the output of the "master" flip flop is passed to both the inputs of the "slave" flip flop. The output of the "slave" flip flop is passed to inputs of the master flip flop.

In "master-slave flip flop", apart from these two flip flops, an inverter or NOT gate is also used. For passing the inverted clock pulse to the "slave" flip flop, the inverter is connected to the clock's pulse. In simple words, when CP set to false for "master", then CP is set to true for "slave", and when CP set to true for "master", then CP is set to false for "slave".

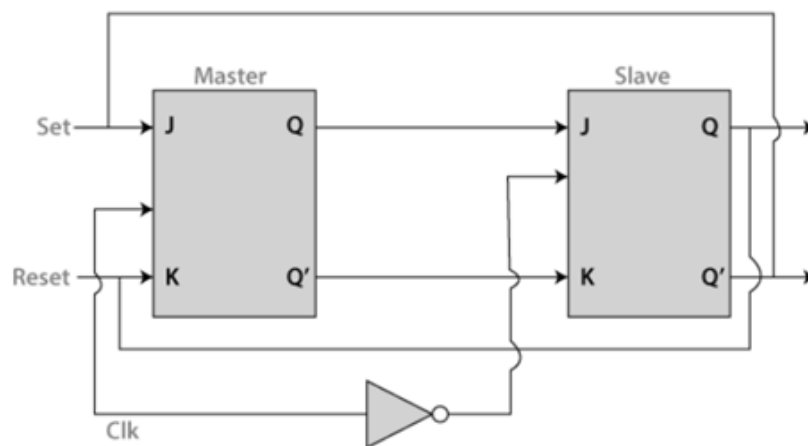


Fig. 4.15 Master slave JK flip flop.

Working:

1. When the clock pulse is true, the slave flip flop will be in the isolated state, and the system's state may be affected by the J and K inputs. The "slave" remains isolated until the CP is 1. When the CP set to 0, the master flip-flop passes the information to the slave flip flop to obtain the output.

2. The master flip flop responds first from the slave because the master flip flop is the positive level trigger, and the slave flip flop is the negative level trigger.
3. The output $Q'=1$ of the master flip flop is passed to the slave flip flop as an input K when the input J set to 0 and K set to 1. The clock forces the slave flip flop to work as reset, and then the slave copies the master flip flop.
4. When $J=1$, and $K=0$, the output $Q=1$ is passed to the J input of the slave. The clock's negative transition sets the slave and copies the master.
5. The master flip flop toggles on the clock's positive transition when the inputs J and K set to 1. At that time, the slave flip flop toggles on the clock's negative transition.
6. The flip flop will be disabled, and Q remains unchanged when both the inputs of the JK flip flop set to 0.

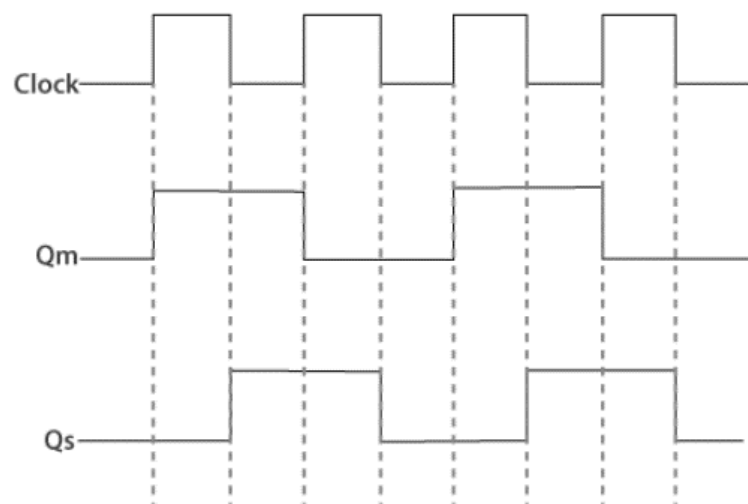


Fig. 4.16 Master slave JK flip flop timing diagram.

4.3 FLIP FLOP EXCITATION TABLES

The characteristic table is useful during the analysis of sequential circuits when the value of flip-flop inputs is known and we want to find the value of the flip-flop output Q after the rising edge of the clock signal. As with any other truth table, we can use the map method to derive the characteristic equation for each flip-flop.

During the design process we usually know the transition from present state to the next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason, we will need a table that lists the required inputs for a given change of state. Such a list is called the excitation table. There are four possible transitions from present state to the next state. The required input conditions are derived from the information available in the characteristic table. The symbol X in the table represents a “don’t care” condition, that is, it does not matter whether the input is 1 or 0.

SR flip flop excitation table:

S	R	Present state Q_n	Next state Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Truth table of SR flip flop

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

Excitation table of SR flip flop

} Invalid states

JK flip flop excitation table:

J	K	Present state Q_n	Next state Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth table of JK flip flop

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

Excitation table of JK flip flop

D-flip flop excitation table:

D	Present state Q_n	Next state Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Truth table of D flip flop

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

Excitation table of D flip flop

T-flip flop excitation table:

T	Present state Q_n	Next state Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Truth table of T flip flop

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

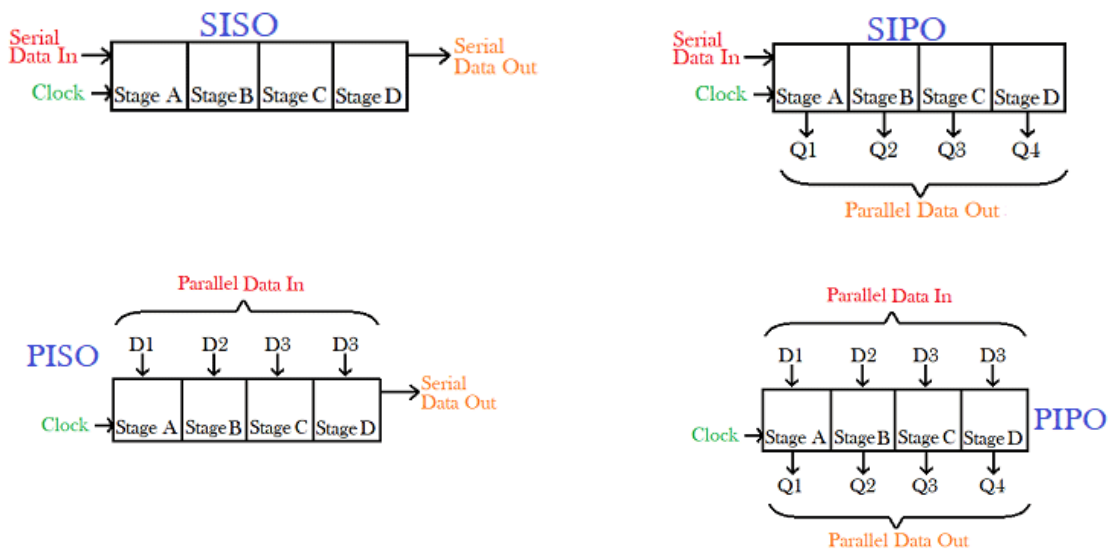
Excitation table of T flip flop

4.4 REGISTERS

We know that one flip-flop can store one-bit of information. In order to store multiple bits of information, we require multiple flip-flops. The group of flip-flops, which are used to hold (store) the binary data is known as register.

If the register is capable of shifting bits either towards right hand side or towards left hand side is known as shift register. An 'N' bit shift register contains 'N' flip-flops. Following are the four types of shift registers based on applying inputs and accessing of outputs.

1. Serial In – Serial Out shift register (SISO)
2. Serial In – Parallel Out shift register (SIPO)
3. Parallel In – Serial Out shift register (PISO)
4. Parallel In – Parallel Out shift register (PIPO)



4.4.1 Serial – in Serial – out Shift register (SISO)

The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register.

1. Right shift register:

This block diagram consists of four D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as serial input. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as serial output.

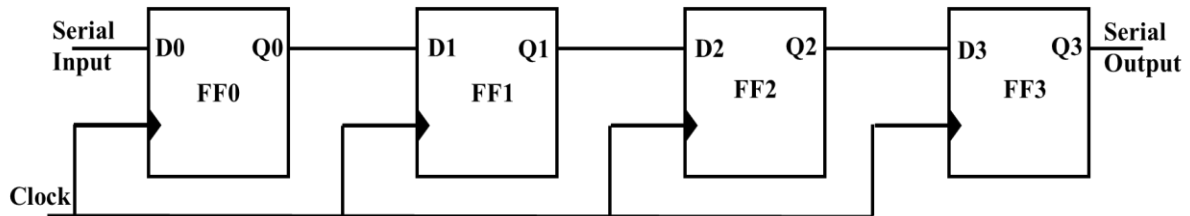


Fig. 4.17 4-bit Right Shift Register

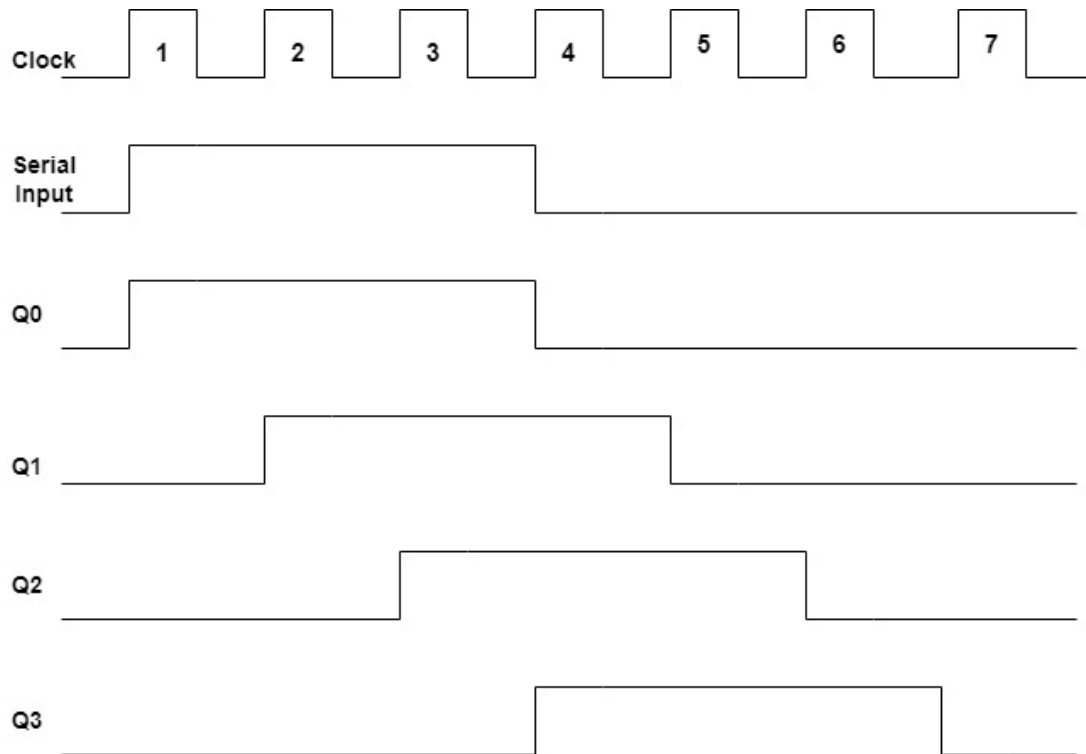
Let us see the working of 4-bit SISO right shift register by sending the binary information “0111” from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is $Q_0Q_1Q_2Q_3=0000$. We can understand the working of 4-bit SISO shift register from the following table.

No of positive edge clock	Serial input	Q0	Q1	Q2	Q3 (Serial Output)
0	-	0	0	0	0
1	1(LSB)	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	0 (MSB)	0	1	1	1(LSB)
5	-	-	0	1	1
6	-	-	-	0	1
7	-	-	-	-	0 (MSB)

The initial status of the D flip-flops in the absence of clock signal is $Q_0Q_1Q_2Q_3=0000$. Here, the serial output is coming from Q_3 . So, the LSB 1 is received at 4th positive edge of clock and the MSB 0 is received at 7th positive edge of clock.

Therefore, the 4-bit SISO shift register requires seven clock pulses in order to produce the valid output. Similarly, the N-bit SISO shift register requires $2N-1$ clock pulses in order to shift ‘N’ bit information.



2. Left shift register:

This block diagram consists of four D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of right most D flip-flop. Hence, this input is also called as serial input. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of left most D flip-flop. Hence, this output is also called as serial output.

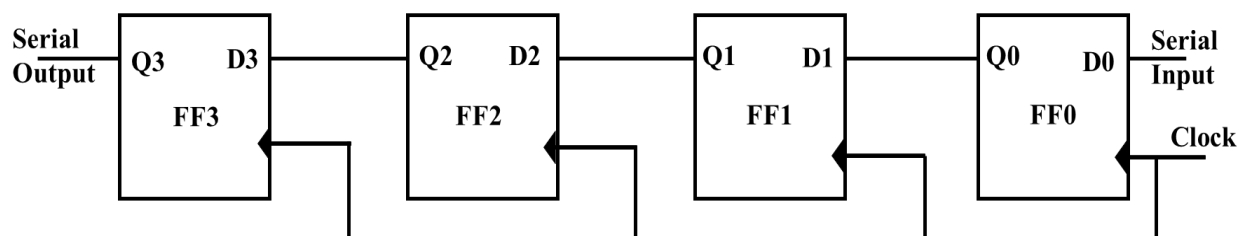


Fig. 4.18 4-bit Left Shift Register

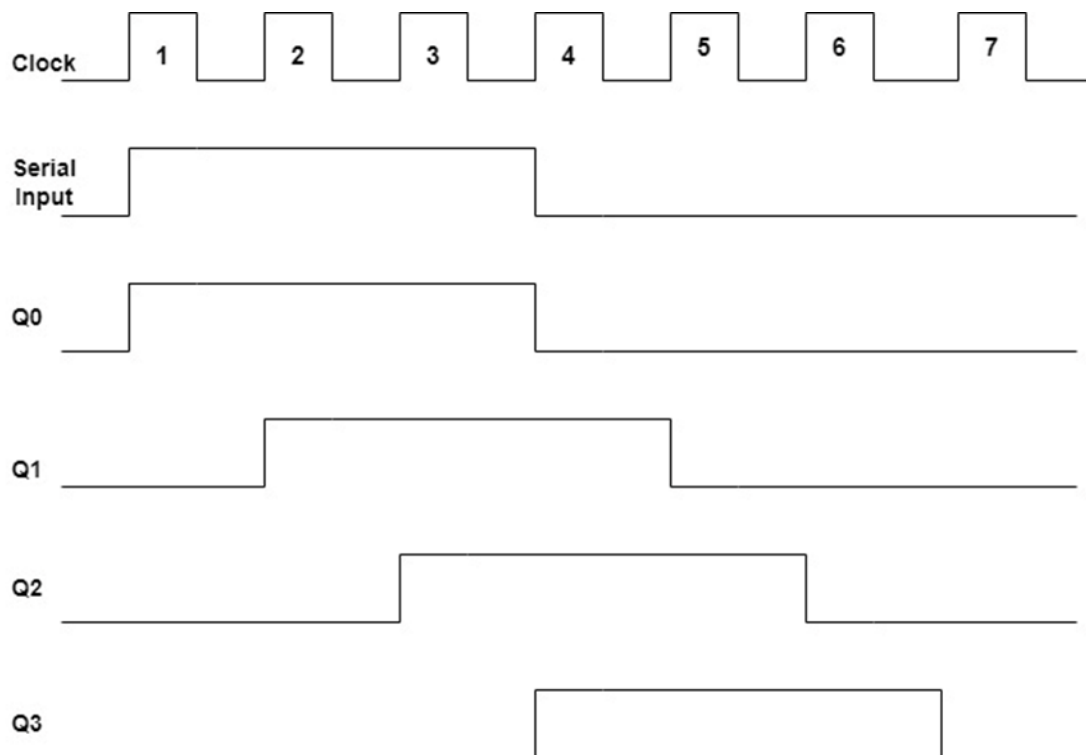
Let us see the working of 4-bit SISO left shift register by sending the binary information “0111” from LSB to MSB serially at the input.

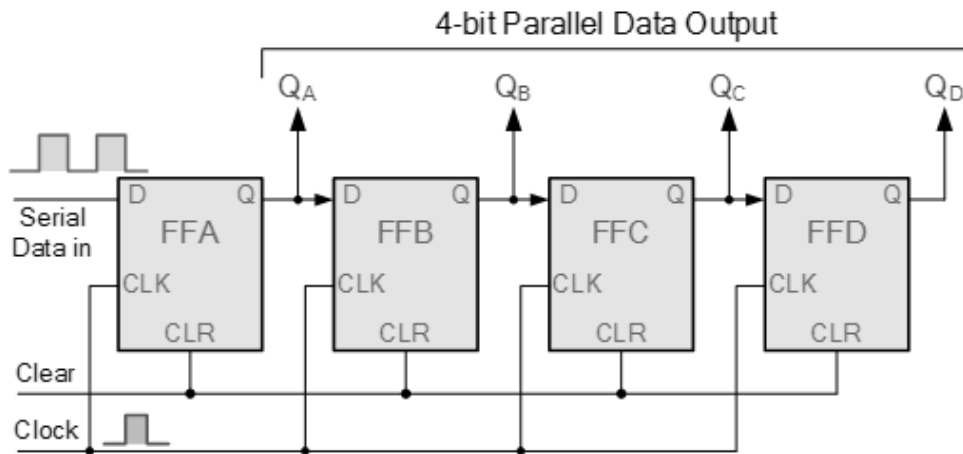
Assume, initial status of the D flip-flops from rightmost to leftmost is $Q_3Q_2Q_1Q_0=0000$. We can understand the working of 4-bit SISO left shift register from the following table.

No of positive edge clock	Serial input	Q0	Q1	Q2	Q3 (Serial Output)
0	-	0	0	0	0
1	1(LSB)	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	0 (MSB)	0	1	1	1(LSB)
5	-	-	0	1	1
6	-	-	-	0	1
7	-	-	-	-	0 (MSB)

The initial status of the D flip-flops in the absence of clock signal is $Q_3Q_2Q_1Q_0=0000$. Here, the serial output is coming from Q_3 . So, the LSB 1 is received at 4th positive edge of clock and the MSB 0 is received at 7th positive edge of clock.

Therefore, the 4-bit SISO left shift register requires seven clock pulses in order to produce the valid output. Similarly, the N-bit SISO shift register requires $2N-1$ clock pulses in order to shift 'N' bit information.



4.4.2 Serial – in Parallel – out Shift register (SIPO)**Fig. 4.19 4-bit Serial-in Parallel-out Register**

No of positive edge clock	Serial input	Q _a	Q _b	Q _c	Q _d
0	-	0	0	0	0
1	0(LSB)	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	0 (MSB)	0	1	1	0(LSB)

The shift register, which allows serial input and produces parallel output is known as Serial-In Parallel-Out SIPO shift register. The block diagram of 4-bit SIPO shift register is shown in the above Fig. 4.19.

This circuit consists of four D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as serial input. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get parallel outputs from this shift register.

Example

Let us see the working of 4-bit SIPO shift register by sending the binary information “0110” from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is $Q_a Q_b Q_c Q_d = 0000$. Here, Q_a & Q_d are MSB & LSB respectively. We can understand the working of 4-bit SIPO shift register from the following table.

The initial status of the D flip-flops in the absence of clock signal is $Q_a Q_b Q_c Q_d = 0000$. The binary information “0110” is obtained in parallel at the outputs of D flip-flops for fourth positive edge of clock. So, the 4-bit SIPO shift register requires four clock pulses in order to produce the valid output. Similarly, the N-bit SIPO shift register requires N clock pulses in order to shift ‘N’ bit information.

4.4.3 Parallel – in Serial – out Shift register (PISO)

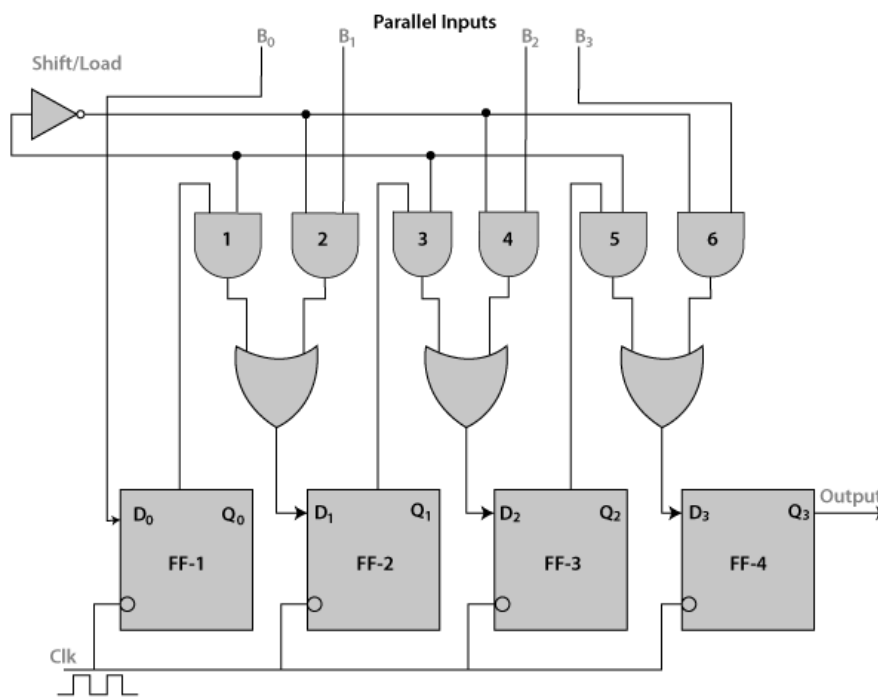
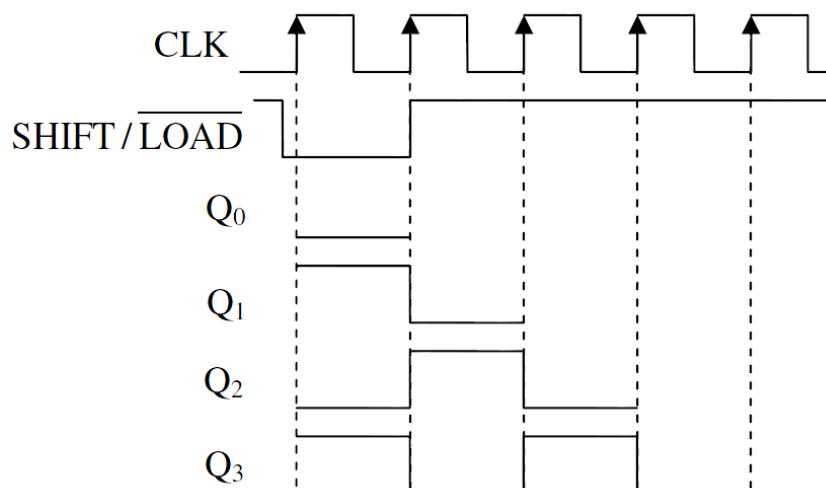


Fig. 4.20 4-bit Parallel-in Serial-out Register



1. Data bits are entered in parallel fashion.
2. The circuit shown below is a four-bit parallel input serial output register.
3. Output of previous Flip Flop is connected to the input of the next one via a combinational circuit.
4. The binary input word B₀, B₁, B₂, B₃ is applied though the same combinational circuit.
5. There are two modes in which this circuit can work namely - shift mode or load mode.

Load mode

When the shift/load bar line is low (0), the AND gate 2, 4 and 6 become active they will pass B₁, B₂, B₃ bits to the corresponding flip-flops. On the low going edge of clock, the binary input B₀, B₁, B₂, B₃ will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode

When the shift/load bar line is low (1), the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore, the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

4.4.4 Parallel – in Parallel – out Shift register (PIPO)

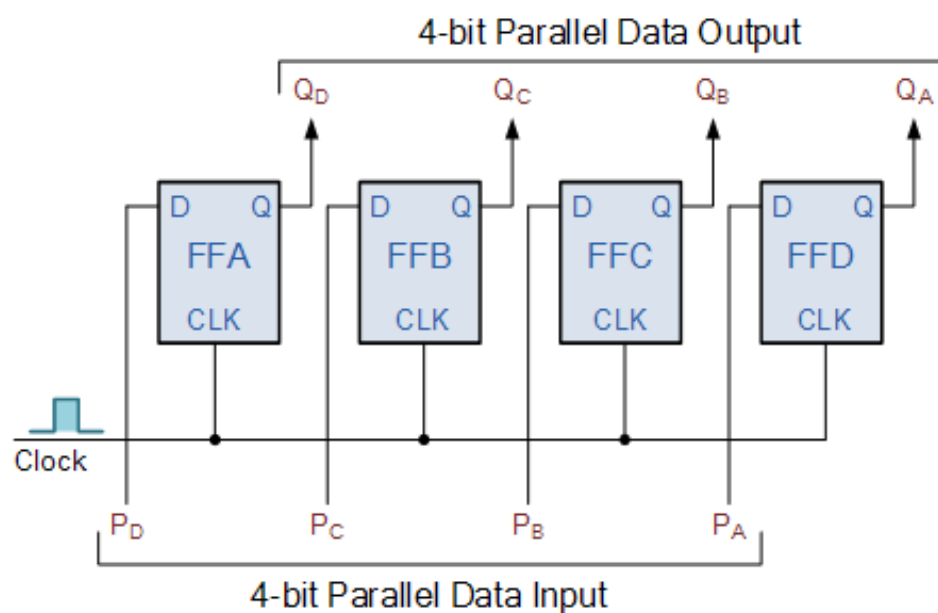


Fig. 4.21 4-bit Parallel-in Parallel-out Register

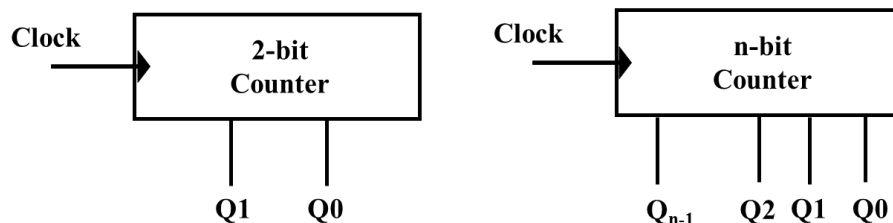
This type of shift register also acts as a temporary storage device or as a time delay device similar to the SISO configuration above. The data is presented in a parallel format to the parallel input pins PA to PD and then transferred together directly to their respective output pins QA to QD by the same clock pulse. Then one clock pulse loads and unloads the register. This arrangement for parallel loading and unloading is shown below.

The PIPO shift register is the simplest of the four configurations as it has only three connections, the parallel input (PI) which determines what enters the flip-flop, the parallel output (PO) and the sequencing clock signal (Clk).

Similar to the Serial-in to Serial-out shift register, this type of register also acts as a temporary storage device or as a time delay device, with the amount of time delay being varied by the frequency of the clock pulses. Also, in this type of register there are no interconnections between the individual flip-flops since no serial shifting of the data is required.

4.5 COUNTERS

A counter is a sequential logic circuit, which is used to count the number of clock pulses arrived at its clock input. The counter has only one input (clock) and multiple outputs, the number of outputs depends on the number of flip flops used to build the counter. For example, a 2-bit counter produces two outputs and two flip flops are used to implement the counter. Similarly, a n-bit counter produces n-outputs and it require n-number of flip flops to design.



The maximum number of clock pulses that a counter can count is called as modules. For a n-bit counter the modulus is 2^n , which means it can count the clock pulses from 0 to 2^n-1 for an up counter.

For example, a 2-bit counter has the modulus of $2^2 = 4$, which is also called as mod-4 counter or 2-bit counter.

A 3-bit counter also known as mod-8 counter, whose modulus is 8 and it counts from 0 to 7 clock pulses.

Based on the type of flip flops used and the way the clock pulse is applied; the counters can be classified into two types.

1. Asynchronous counters
2. Synchronous counters

The asynchronous counters are also known as ripple counters, in the implementation of asynchronous counters only T-flip flops or the equivalent of T-flip flop is used (connecting both inputs of a JK flip flop to logic 1 turns the JK flip flop in to TFF, connecting the complementing output of a D-flip flop to its D input turns the DFF into TFF). Only the first flip flop receives the actual clock input and the next stage flip flop clock input is driven by previous stage output.

The synchronous counters can be implemented with any flip flop (RS, JK, T or D). In this counter design all the flip flops receive the same clock pulse or all the flip flops are clocked simultaneously.

4.5.1 Mod-4 or 2-bit Ripple Up Counter

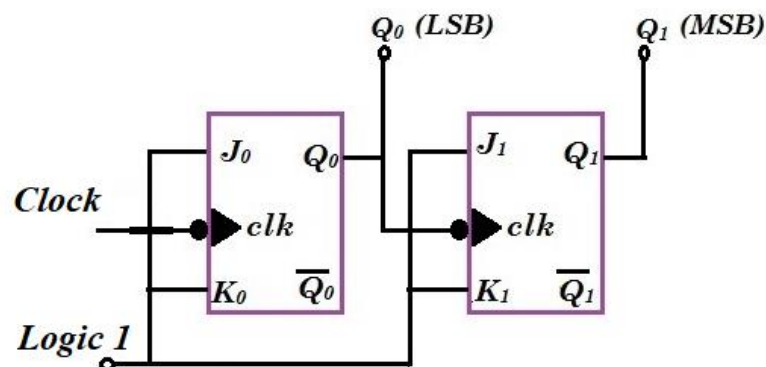


Fig. 4.22 Mod-4 or 2-bit Ripple Counter

Construction:

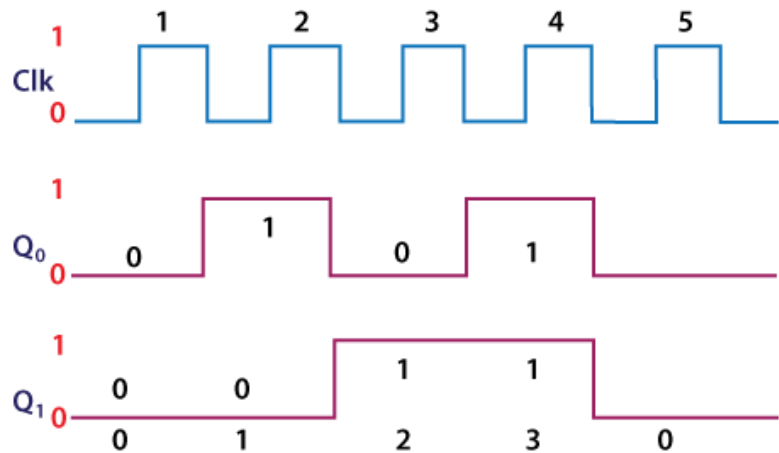
1. The above fig. 4.22 shows the circuit diagram of Mod-4 or 2-bit asynchronous or ripple counter.
2. It has two T-flip flops where the JK flip flops is converted into T flip flop by connecting both J and K inputs to logic 1.
3. The first flip flop receives the clock signal, where the second flip flop clock signal is driven by the normal output of the first flip flop.
4. It has two outputs one from each flip flop, the first flip flop output forms the LSB and the second flip flop output forms the MSB.

Working:

1. Since the inputs of a JK flip flop are high, it toggles its output for every negative edge of the clock pulse.
2. Here the output of the first flip flop toggles for every negative edge of the clock signal and the second flip flop toggles its output when the Q_0 changes its state from 0 to 1 or when Q_0 is in negative edge.
3. Initially assume all the flip flops are reset i.e $Q_0Q_1 = 00$, after every clock pulse it changes from 00, 01, 10, 11 and returns to initial state 00.

Truth table and timing diagram:

Clock	Q1	Q0
0	--	--
1	0	0
2	0	1
3	1	0
4	1	1
5	0	0



4.5.2 Mod-8 or 3-bit Ripple Up Counter

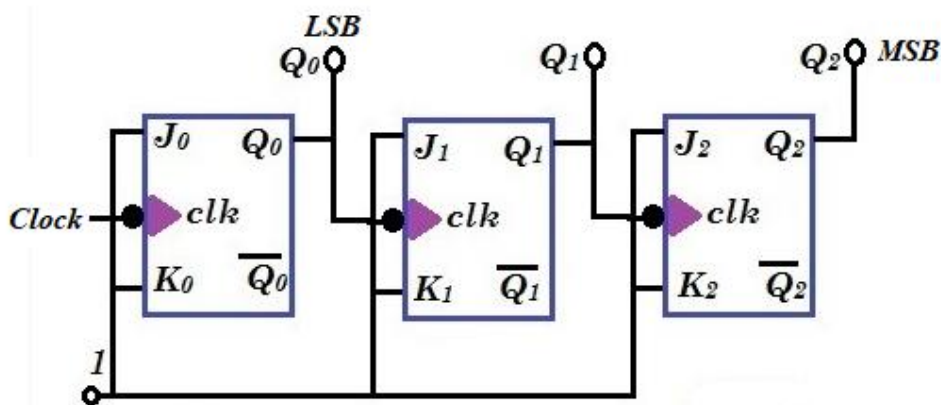


Fig. 4.23 Mod-8 or 3-bit Ripple Counter

Construction:

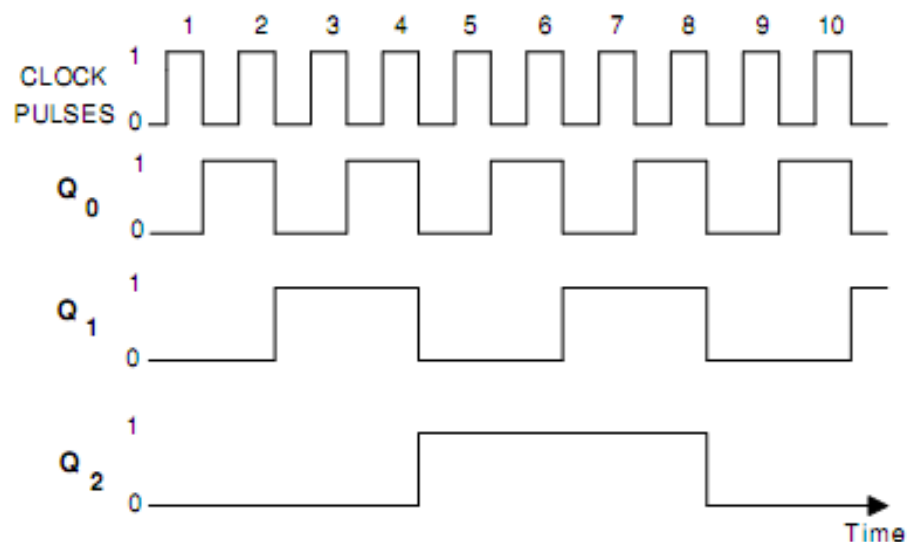
1. The above fig. 4.23 shows the circuit diagram of Mod-8 or 3-bit asynchronous or ripple counter.
2. It has three T-flip flops where the JK flip flops is converted into T flip flop by connecting both J and K inputs to logic 1.
3. The first flip flop receives the clock signal, where the second and third flip flops clock input is driven by the normal output of previous flip flops.
4. It has three outputs one from each flip flop, the first flip flop output forms the LSB and the third flip flop output forms the MSB.

Working:

1. In the 3-bit ripple counter, three flip-flops are used in the circuit. As here 'n' value is three, the counter can count up to $2^3 = 8$ values. i.e. 000,001,010,011,100,101,110,111.
2. Here the output of the first flip flop toggles for every negative edge of the clock signal and the second flip flop toggles its output when the Q_0 changes its state from 0 to 1 or when Q_0 is in negative edge, similarly, output Q_2 toggles with negative edge in Q_1 .

Truth table and timing diagram:

Clock	Q2	Q1	Q0
0	--	--	--
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
9	0	0	0



4.5.3 Mod-16 or 4-bit Ripple Up Counter

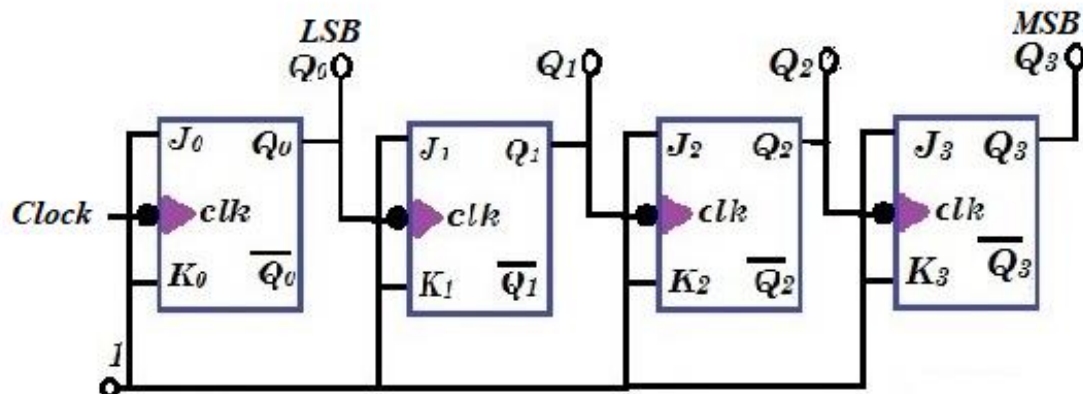


Fig. 4.24 Mod-16 or 4-bit Ripple Counter

Construction:

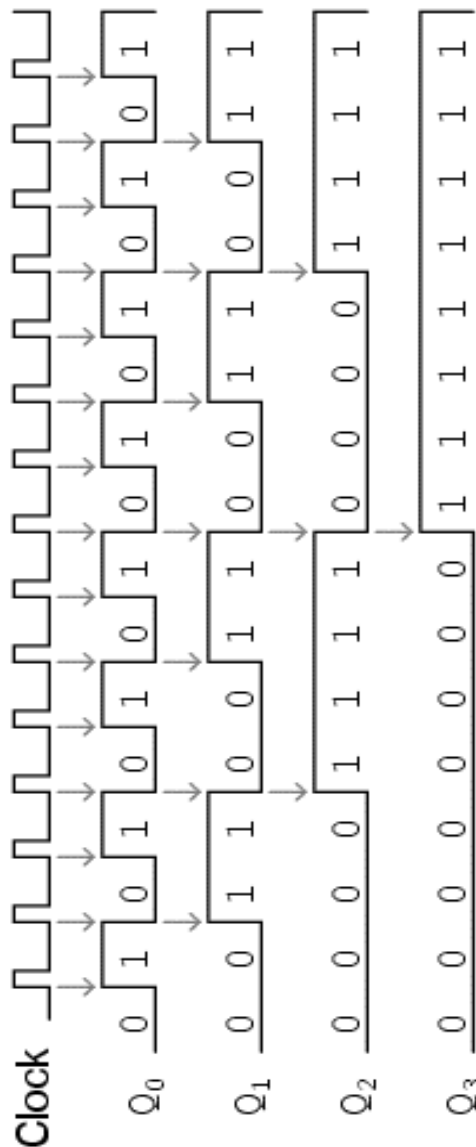
1. The above fig. 4.24 shows the circuit diagram of Mod-16 or 4-bit asynchronous or ripple counter.
2. It has three T-flip flops where the JK flip flops is converted into T flip flop by connecting both J and K inputs to logic 1.
3. The first flip flop receives the clock signal, where the second, third and fourth flip flops clock input is driven by the normal output of previous flip flops.
4. It has four outputs one from each flip flop, the first flip flop output forms the LSB and the fourth flip flop output forms the MSB.

Working:

1. In the 4-bit ripple counter, four flip-flops are used in the circuit. As here 'n' value is four, the counter can count up to $2^4 = 16$ values. i.e. 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.
2. Here the output of the first flip flop toggles for every negative edge of the clock signal and the second flip flop toggles its output when the Q_1 changes its state from 0 to 1 or when Q_0 is in negative edge, similarly, output Q_2 toggles with negative in Q_1 and output Q_3 toggles with negative edge in Q_2 .

Truth table and timing diagram:

Clock	Q3	Q2	Q1	Q0	Clock	Q3	Q2	Q1	Q0
0	--	--	--	--	9	1	0	0	0
1	0	0	0	0	10	1	0	0	1
2	0	0	0	1	11	1	0	1	0
3	0	0	1	0	12	1	0	1	1
4	0	0	1	1	13	1	1	0	0
5	0	1	0	0	14	1	1	0	1
6	0	1	0	1	15	1	1	1	0
7	0	1	1	0	16	1	1	1	1
8	0	1	1	1	17	0	0	0	0



4.5.4 Mod-10 or Decade Counter

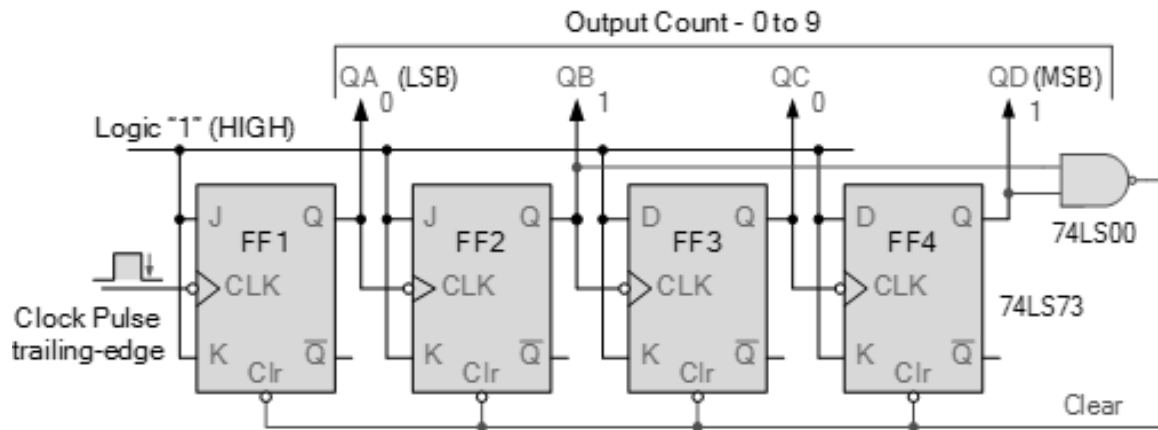
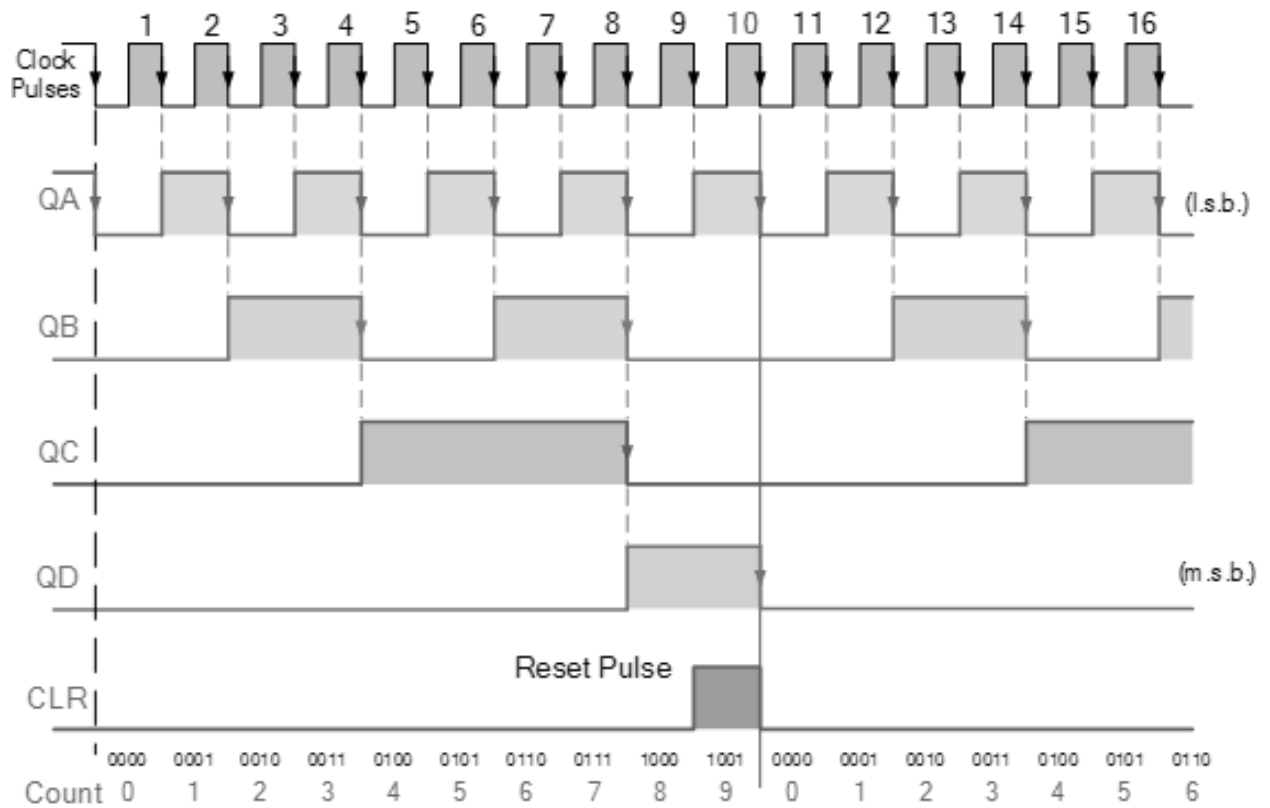


Fig. 4.25 Mod-10 or Decade Counter

1. An Asynchronous counter can have 0 to 2^n-1 possible counting states e.g. MOD-16 for a 4-bit counter, (0-15) making it ideal for use in Frequency Division applications. But it is also possible to use the basic asynchronous counter configuration to construct special counters with counting states less than their maximum output number. For example, modulo or MOD counters.
2. This is achieved by forcing the counter to reset itself to zero at a pre-determined value producing a type of asynchronous counter that has truncated sequences. Then an n-bit counter that counts up to its maximum modulus (2^n) is called a full sequence counter and a n-bit counter whose modulus is less than the maximum possible is called a truncated counter.
3. If we take the modulo-16 asynchronous counter and modified it with additional logic gates it can be made to give a decade (divide-by-10) counter output for use in standard decimal counting and arithmetic circuits. Such counters are generally referred to as Decade Counters.
4. A decade counter requires resetting to zero when the output count reaches the decimal value of 10, ie. when $Q_D Q_C Q_B Q_A = 1010$ and to do this we need to feed this condition back to the reset input.
5. When the counter enters into 1010 (10^{th}) state, the output of NAND gate is zero and it is connected to the active low clear input of each flip flop, which forces the flip flops to clear its outputs and the counter enters into 0000 initial state.

Clock Count	Output bit Pattern				Decimal Value
	QD	QC	QB	QA	
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9
11	Counter Resets its Outputs back to Zero				



4.5.5 Asynchronous Down Counter

For the 3-bit counter, we require 3 flip flops and we can generate $2^3 = 8$ state and count (111 ... 000).

In this implementation, the clock pulse is given to only the first FF. Thereafter, the output of the first FF is feed as a clock to second FF and the output of the second FF is feed as the clock for the third FF. But the complemented output is taken from each FF (i.e. same as Up counter but output states are complemented). Here Q'_A is LSB and Q'_C is MSB.

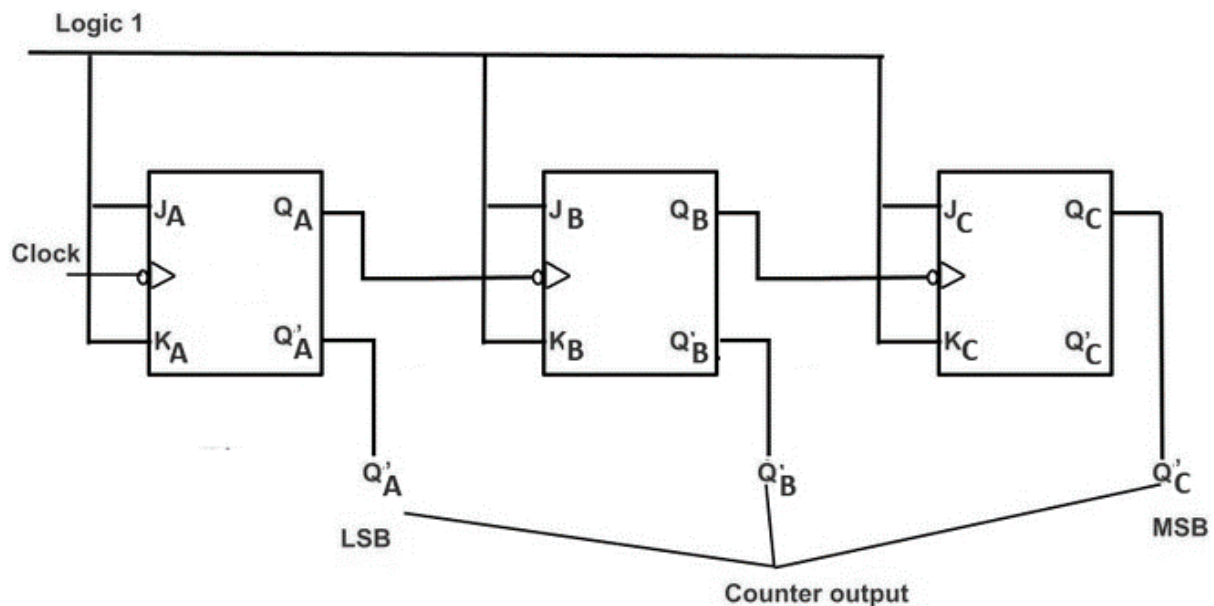


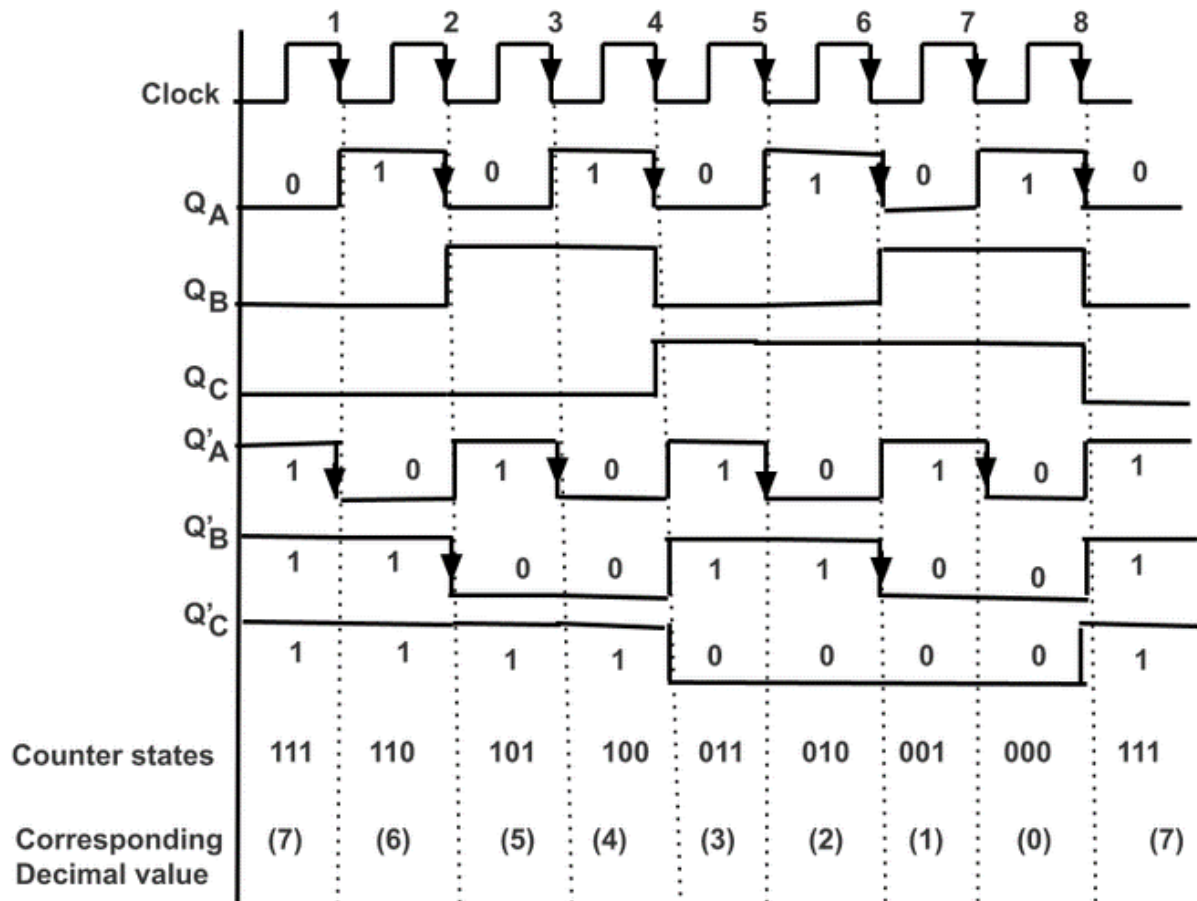
Fig. 4.26 Asynchronous 3-bit Down Counter

1. The -ve edge clock pulse is provided to 1st flip flop. Therefore, the output state of the first flip flop (i.e. Q_A) will be toggled at every falling edge of the clock pulse.
2. As the Q_A is feed as a clock to the second FF, therefore the output state (i.e. Q_B) will be toggled at every falling edge of Q_A .
3. In the same manner, the Q_B acts as clock for the third FF, therefore the output state (Q_C) of the third FF will be toggled for every falling edge of Q_B .
4. As we know, this is the working of UP counter, but here the output is taken as in complemented form (i.e. $Q'_C Q'_B Q'_A$), therefore we get the complemented outputs (i.e. down counting 111 to 000)
5. After the 8th falling edge of the external clock pulse, the counter is reset to 000.

Truth Table:

Clock	Q _C	Q _B	Q _A	Q' _C	Q' _B	Q' _A
Initially	0	0	0	1	1	1
1st	0	0	1	1	1	0
2nd	0	1	0	1	0	1
3rd	0	1	1	1	0	0
4th	1	0	0	0	1	1
5th	1	0	1	0	1	0
6th	1	1	0	0	0	1
7th	1	1	1	0	0	0

Timing Diagram:



4.5.6 Synchronous 4-bit Counter

If all the flip-flops receive the same clock signal, then that counter is called as Synchronous counter. Hence, the outputs of all flip-flops change at the same time. An ‘N’ bit Synchronous binary up counter consists of ‘N’ T flip-flops. It counts from 0 to $2^N - 1$. The block diagram of 4-bit Synchronous binary up counter is shown in the following figure.

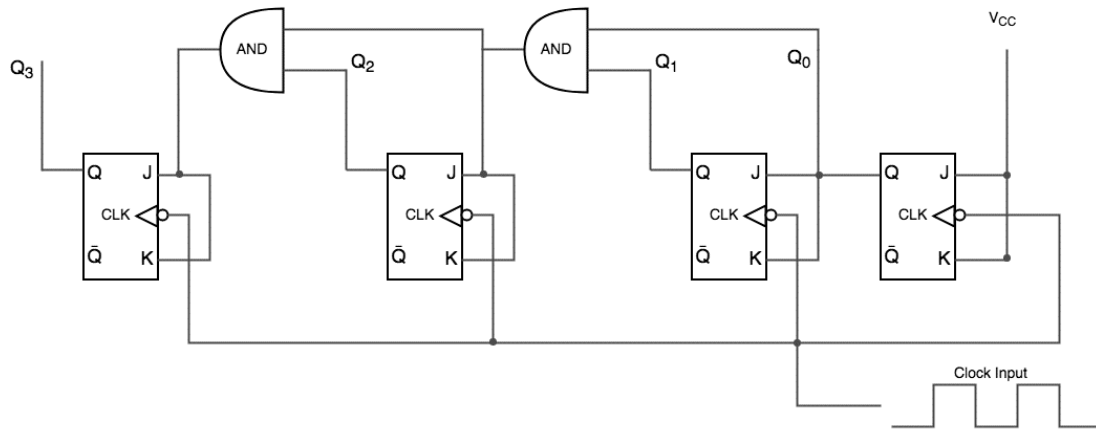
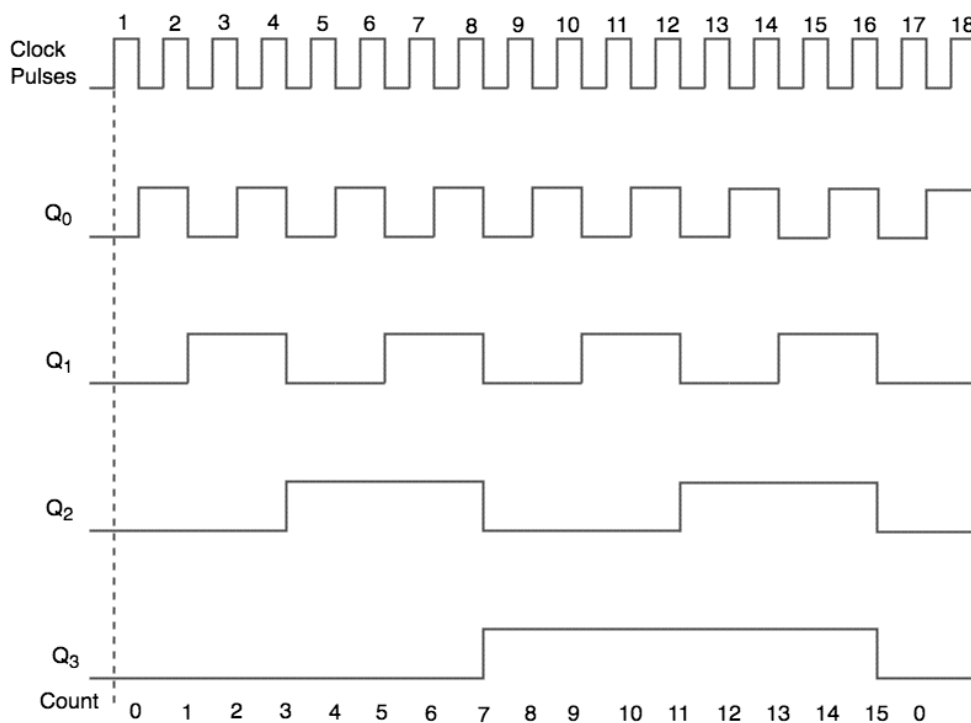


Fig. 4.27 Synchronous 4-bit Up Counter

The 4-bit Synchronous binary up counter contains four T flip-flops & two 2-input AND gates. All these flip-flops are negative edge triggered and the outputs of flip-flops change synchronously. From circuit diagram we see that Q₀ bit gives response to each falling edge of clock while Q₁ is dependent on Q₀, Q₂ is dependent on Q₁ and Q₀, Q₃ is dependent on Q₂, Q₁ and Q₀.



4.5.7 Ring Counter

Ring counter is a typical application of Shift register. Ring counter is almost same as the shift register. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in case of ring counter but in case of shift register it is taken as output.

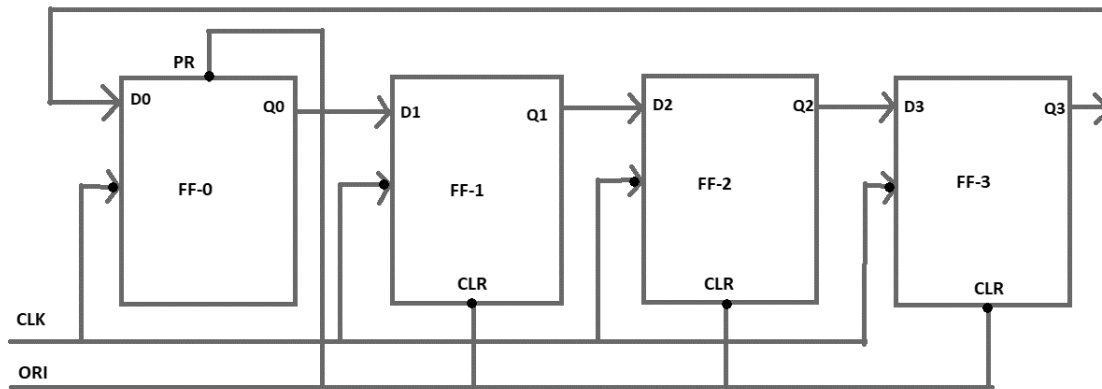


Fig. 4.28 A 4-bit Ring Counter

In this diagram, we can see that the clock pulse (CLK) is applied to all the flip-flop simultaneously. Therefore, it is a Synchronous Counter. Also, here we use Overriding input (ORI) to each flip-flop. Preset (PR) and Clear (CLR) are used as ORI. When PR is 0, then the output is 1. And when CLR is 0, then the output is 0. Both PR and CLR are active low signal that is always works in value 0. These two values are always fixed. They are independent with the value of input D and the Clock pulse (CLK).

Working: Here, ORI is connected to Preset (PR) in FF-0 and it is connected to Clear (CLR) in FF-1, FF-2, and FF-3. Thus, output $Q = 1$ is generated at FF-0 and rest of the flip-flop generate output $Q = 0$. This output $Q = 1$ at FF-0 is known as Pre-set 1 which is used to form the ring in the Ring Counter.

This Pre-setted 1 is generated by making ORI low and that time Clock (CLK) becomes don't care. After that ORI made to high and apply low clock pulse signal as the Clock (CLK) is negative edge triggered. After that, at each clock pulse the pre-setted 1 is shifted to the next flip-flop and thus form Ring. From the below table, we can say that there are 4 states in 4-bit Ring Counter.

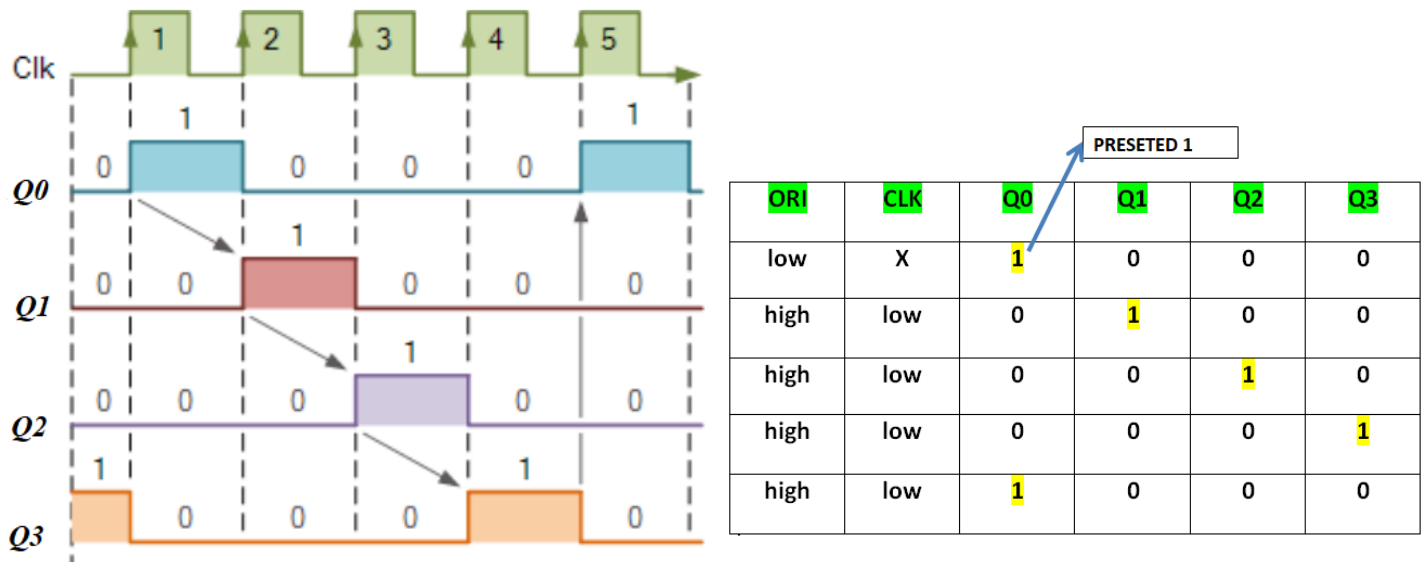


Fig. 4.29 Timing Diagram and Truth Table of 4-bit Ring Counter

UNIT – 5

MEMORY DEVICES

5.1 GENERAL MEMORY OPERATIONS

A memory unit stores binary information in groups of bits called words. Data input lines provide the information to be stored into the memory, Data output lines carry the information out from the memory. The control lines Read and write specifies the direction of transfer of data. Basically, in the memory organization, there are 2^l memory locations indexing from 0 to 2^l-1 where l is the address lines. We can describe the memory in terms of the bytes using the following formula:

$$N = 2^l \text{ Bytes}$$

l is the total address lines; N is the memory in bytes

For example, some storage can be described below in terms of bytes using the above formula:

$$\begin{aligned} 1\text{ kB} &= 2^{10} \text{ Bytes} \\ 64 \text{ kB} &= 2^6 \times 2^{10} \text{ Bytes} \\ &= 2^{16} \text{ Bytes} \\ 4 \text{ GB} &= 2^2 \times 2^{10}(\text{kB}) \times 2^{10}(\text{MB}) \times 2^{10} \text{ Bytes} \\ &= 2^{32} \text{ Bytes} \end{aligned}$$

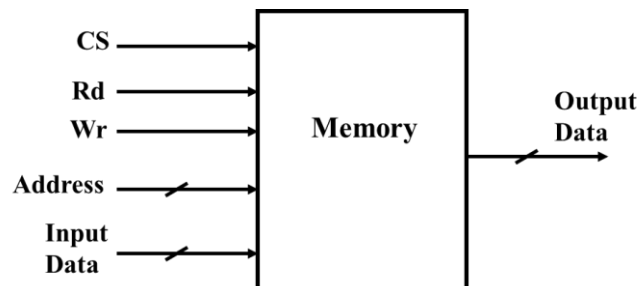


Fig. 5.1 Block diagram of Memory

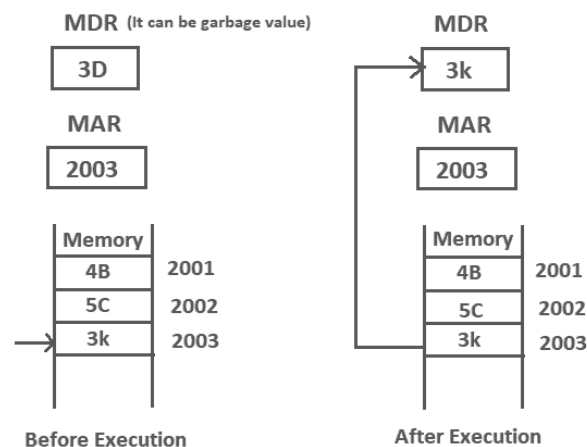
- CS : Chip Select – Used to enable /disable the memory.
- Rd, Wr : Control signals, used to indicate type of operation.
- Address : Address lines which specifies the location of memory for read or write operation.
- Input Data : Input data to memory.
- Output lines : Output data from memory.

Memory Address Register (MAR) is the address register which is used to store the address of the memory location where the operation is being performed.

Memory Data Register (MDR) is the data register which is used to store the data on which the operation is being performed.

1. Memory Read Operation:

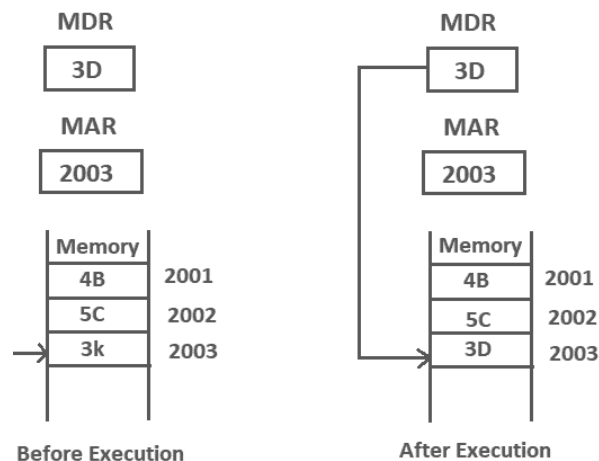
Memory read operation transfers the desired word to address lines and activates the read control line. Description of memory read operation is given below:



In the above diagram initially, MDR can contain any garbage value and MAR is containing 2003 memory address. After the execution of read instruction, the data of memory location 2003 will be read and the MDR will get updated by the value of the 2003 memory location (3D).

2. Memory Write Operation:

Memory write operation transfers the address of the desired word to the address lines, transfers the data bits to be stored in memory to the data input lines. Then it activates the write control line. Description of the write operation is given below:



In the above diagram, the MAR contains 2003 and MDR contains 3D. After the execution of write instruction 3D will be written at 2003 memory location.

5.2 TYPES OF MEMORY

Memory chips are digital logic devices used in microcomputer systems to store programs and data. It is necessary to understand how they work in order to understand the operation of a microcomputer. There are two main types of memory chips: ROM and RAM. The contents of a ROM (Read only memory) chip can only be read. The contents of a RAM (Random access memory) chip can be read and written. The advantage of a ROM is that, unlike RAM, its contents are retained when power is removed.

There are many types of ROM chips. The most common are EPROM (Erase and programmable read only memory) which can be erased by exposing them to strong UV (ultraviolet) light for several minutes and then re-written using a device called device programmer. EEPROMs (Electrically erasable read only memory) are similar to EPROMs, but they can be quickly erasable by the use of electrical signals.

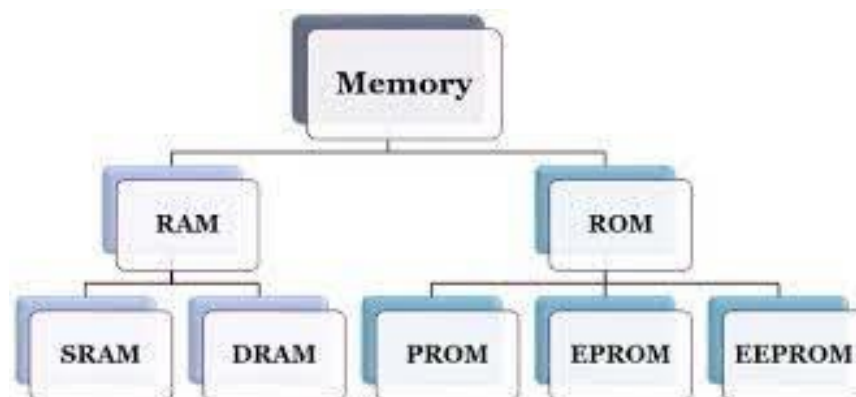


Fig. 5.2 Classification of Memory

5.2.1 Random Access Memory (RAM)

RAM (Random Access Memory) is a part of computer's Main Memory which is directly accessible by CPU. RAM is used to Read and Write data into it which is accessed by CPU randomly. RAM is volatile in nature, it means if the power goes off, the stored information is lost. RAM is used to store the data that is currently processed by the CPU. Most of the programs and data that are modifiable are stored in RAM.

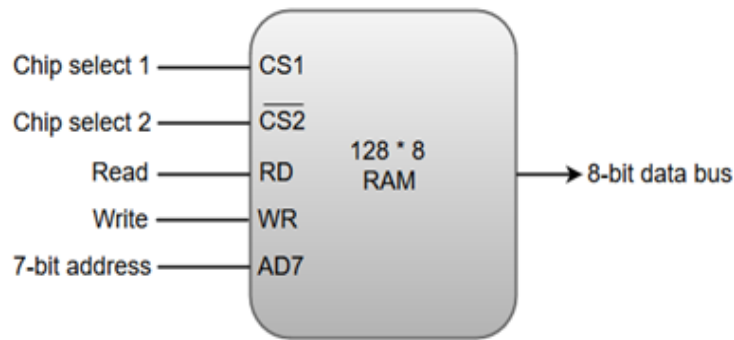


Fig. 5.3 Block diagram of RAM

Integrated RAM chips are available in two form:

1. SRAM (Static RAM)
2. DRAM (Dynamic RAM)

1. Static Random-access Memory (SRAM):

Data is stored in transistors and requires a constant power flow. Because of the continuous power, SRAM doesn't need to be refreshed to remember the data being stored. SRAM is called static as no change or action i.e. refreshing is not needed to keep the data intact. It is used in cache memories.

Advantage: Low power consumption and faster access speeds.

Disadvantage: Less memory capacities and high costs of manufacturing.

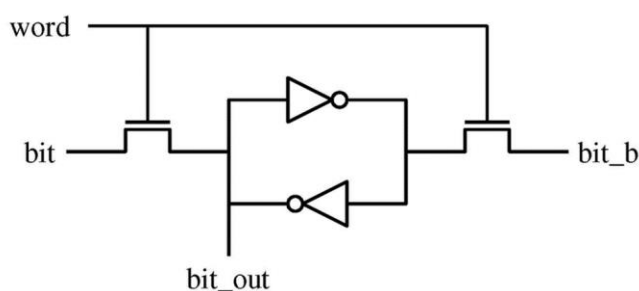
2. Dynamic Random-access Memory (DRAM):

Data is stored in capacitors. Capacitors that store data in DRAM gradually discharge energy, no energy means the data has been lost. So, a periodic refresh of power is required in order to function. DRAM is called dynamic as constant change or action i.e. refreshing is needed to keep the data intact. It is used to implement main memory.

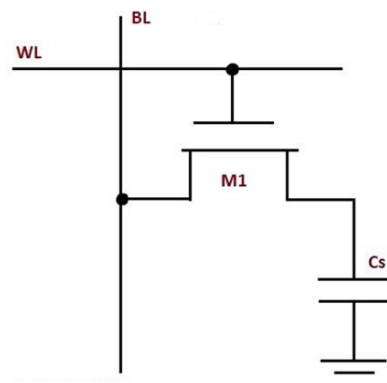
Advantage: Low costs of manufacturing and greater memory capacities.

Disadvantage: Slow access speed and high-power consumption.

SRAM Cell



DRAM Cell

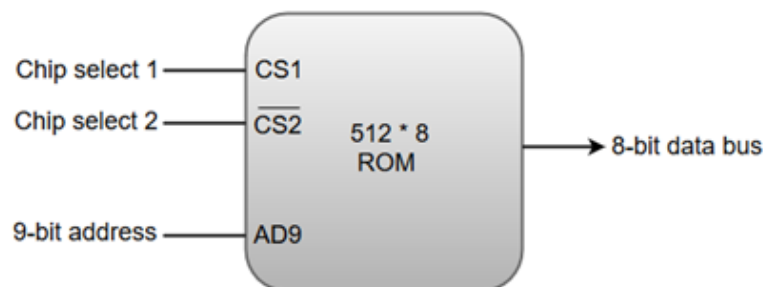


Difference between SRAM and DRAM:

SRAM	DRAM
Transistors are used to store information in SRAM.	Capacitors are used to store data in DRAM.
Capacitors are not used hence no refreshing is required.	To store information for a longer time, contents of the capacitor needs to be refreshed periodically.
SRAM is faster as compared to DRAM.	DRAM provides slow access speeds.
These are expensive.	These are cheaper.
SRAMs are low density devices.	DRAMs are high density devices.
These are used in cache memories.	These are used in main memories.

5.2.2 Read Only Memory (ROM)

ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as bootstrap. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.

**Fig. 5.4 Block diagram of ROM**

Integrated ROM chips are available in three form:

1. PROM (Programmable Read Only Memory)
2. EPROM (Erase and Programmable Read only Memory)
3. EEPROM (Electrically Erase and Programmable Read only Memory)

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

Advantages of ROM

The advantages of ROM are as follows –

1. Non-volatile in nature
2. Cannot be accidentally changed
3. Cheaper than RAMs
4. Easy to test
5. More reliable than RAMs
6. Static and do not require refreshing
7. Contents are always known and can be verified

Following are the important differences between RAM and ROM.

S. No.	Feature	RAM	ROM
1	Definition	RAM stands for Random Access Memory.	ROM stands for Read Only Memory.
2	Data Retention	RAM data is volatile. Data is present till power supply is present.	ROM data is permanent. Data remains even after power supply is not present.
3	Data Access	RAM data can be read, erased or modified.	ROM data is read only.
4	Usage	RAM is used to store data that CPU needs for current instruction processing.	ROM is used to store data that is needed to bootstrap the computer.
5	Speed	RAM speed is quite high.	ROM speed is slower than RAM.
6	CPU Access	CPU can access data stored on RAM.	Data to be copied from ROM to RAM so that CPU can access its data.
7	Capacity	RAM memory is large and high capacity.	ROM is generally small and of low capacity.
8	Usage	RAM is used as CPU Cache, Primary Memory.	ROM is used as firmware by microcontrollers.
9	Cost	RAM is costly.	ROM is cheap.

5.3 PROGRAMMABLE LOGIC ARRAY (PLA)

PLA is a programmable logic device that has both Programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The block diagram of PLA is shown in the following figure.

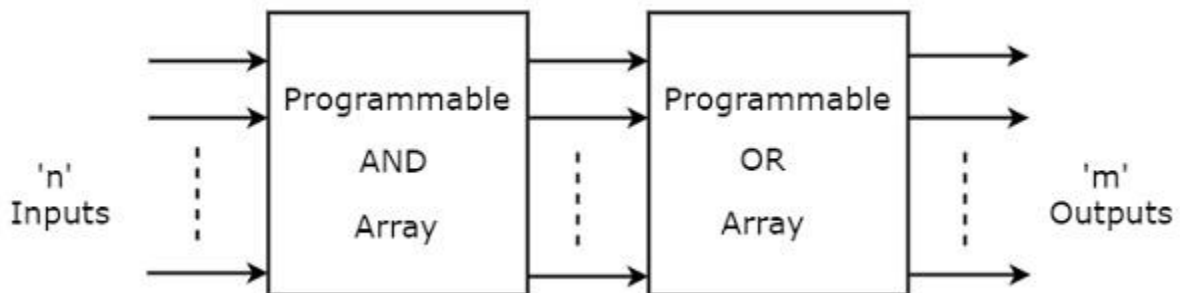


Fig. 5.5 Block diagram of PLA

Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required product terms by using these AND gates.

Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of sum of products form.

Example

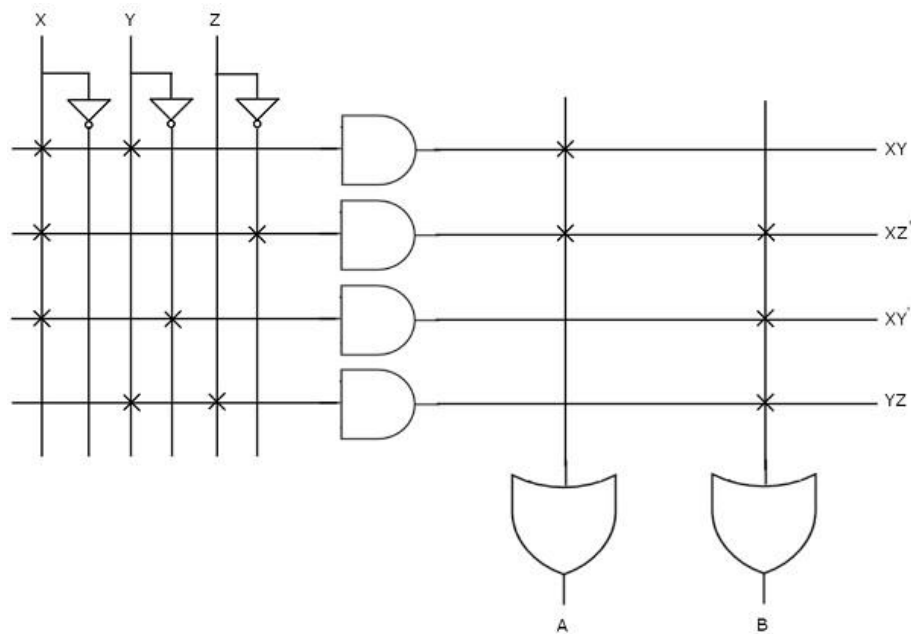
Let us implement the following Boolean functions using PLA.

$$A=XY+XZ'; B=XY'+YZ+XZ'$$

The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, XZ' is common in each function.

So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding PLA is shown in the following figure.

The programmable AND gates have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X', Y, Y', Z & Z' , are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.



All these product terms are available at the inputs of each programmable OR gate. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol 'X' is used for programmable connections.

5.4 PROGRAMMABLE ARRAY LOGIC (PAL)

PAL is a programmable logic device that has Programmable AND array & fixed OR array. The advantage of PAL is that we can generate only the required product terms of Boolean function instead of generating all the min terms by using programmable AND gates. The block diagram of PAL is shown in the following figure.

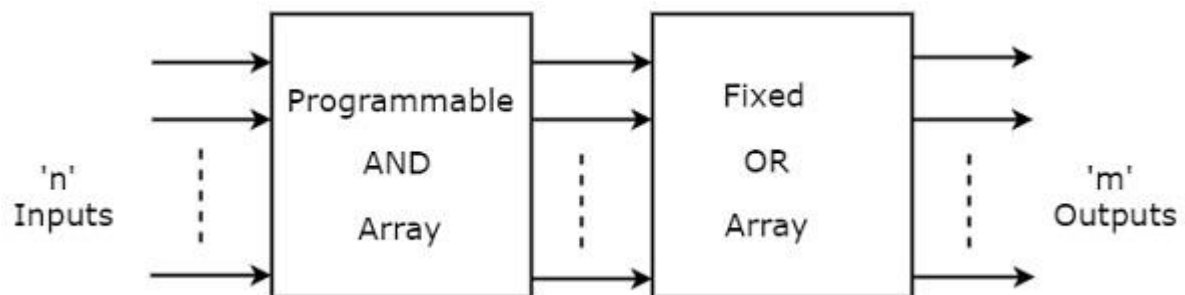


Fig. 5.6 Block diagram of PAL

Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required product terms by using these AND gates.

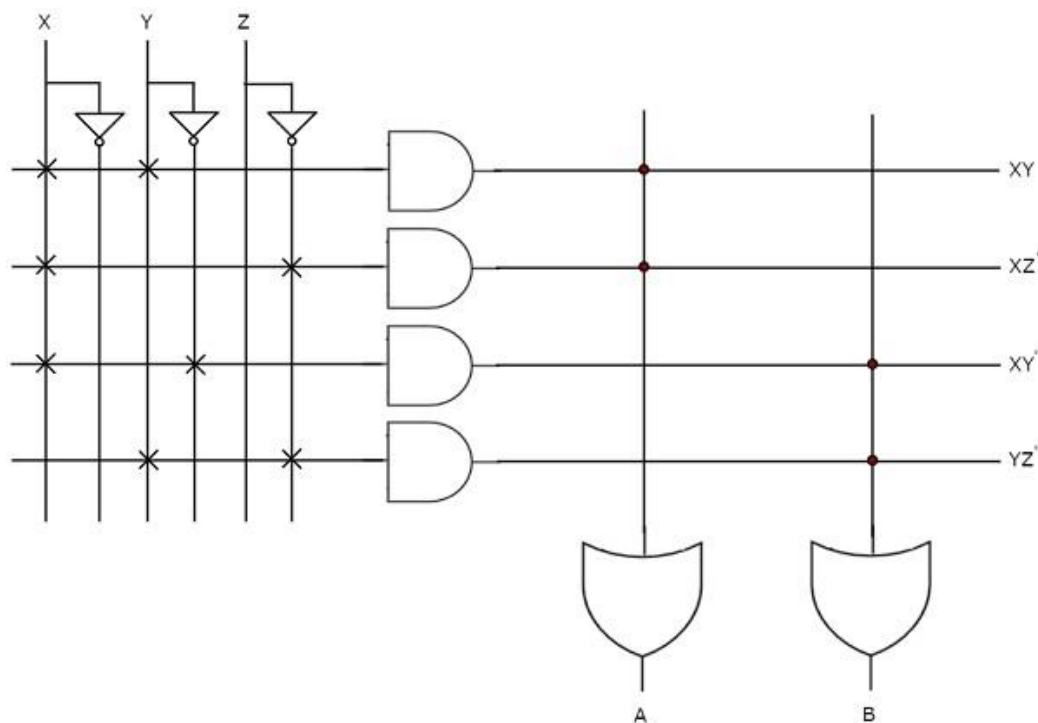
Here, the inputs of OR gates are not of programmable type. So, the number of inputs to each OR gate will be of fixed type. Hence, apply those required product terms to each OR gate as inputs. Therefore, the outputs of PAL will be in the form of sum of products form.

Example

Let us implement the following Boolean functions using PAL.

$$A=XY+XZ'; B=XY'+YZ'$$

The given two functions are in sum of products form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions. The corresponding PAL is shown in the following figure.



The programmable AND gates have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X', Y, Y', Z & Z', are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate. The symbol 'X' is used for programmable connections.

Here, the inputs of OR gates are of fixed type. So, the necessary product terms are connected to inputs of each OR gate. So that the OR gates produce the respective Boolean functions. The symbol '.' is used for fixed connections.