Insights on

DIGITAL LOGIC

By:

Dr. Dibakar Raj Pant
(Associate Prof., Dept. of Electronics and Computer Engineering, Pulchowk Campus)

Er. Raju Shrestha
(Lecturer, National College of Engineering)

Er. Reshma Maharjan
(Senior Lecturer, Kathmandu Engineering College)

Er. Ranjeeta Paudel
(Former Sr. Lecturer, Kathmandu Engineering College)

Er. Shashi Raj Pandey
(PhD student, Intelligent Networking Lab School of Computer Science and Engineering, Kyung Hee University, S. Korea)

SYSTEM INCEPTION

Insights on DIGITAL LOGIC

Published by : System Inception
Cell: 9851093684

Distributed by : Trishakti Pustak Bhandar
Thapathali, Kathmandu
Phone: 9841304570

Authors : Dr. Dibakar Raj Pant
Er. Raju Shrestha
Er. Reshma Maharjan
Er. Ranjeeta Paudel
Er. Shashi Raj Pandey

Edition : **First,** 2074 BS (2018 AD)
Second, 2075 BS (2019 AD)

ISBN : 978-9937-0-3587-3

Computer : The Creation Graphics
Bagbazar, Kathmandu

Printed at : Solution Press Pvt. Ltd.
Kathmandu

PREFACE TO THE SECOND EDITION

The textbook "***Insights on* Digital Logic"** is an outcome of our teaching experience and never ending efforts in making the subject easier. This book presents all topics in a very simple and lucid way so that the students will feel comfortable to prepare for the exams and implement their knowledge in real-world applications.

We are highly indebted to **Er. Shyam Dahal** (Sr. Lecturer, KEC, Kalimati) who paved the way towards accomplishing this fine project. We are also very thankful to all others who directly or indirectly helped us bringing the book in the most understandable format.

Before we pen down, we want to request "Always be helpful and never hurt others". Please make these feelings go viral. Let's together pray for a prosperous and peaceful Nepal.

Authors

CONTENTS

Chapter – 1

INTRODUCTION

1.1 Definitions for Digital Signals

Electronic circuits and systems can be conveniently divided into two broad categories generally referred to as analog and digital circuits. Analog circuits are designed for use with small signals while digital circuits are generally used with large signals. An op-amp with a voltage gain of 5 is an analog circuit. A remote control circuit that switches the lights in a parking area on after sunset and turns them off at sunrise is an example of a digital circuit.

Any quantity that changes with time either can be represented as an analog signal or it can be treated as a digital signal. For example, let's apply heat to a container of water and record the temperature. There are two ways to record the water temperature over a period of time. In fig. (a), the temperature is recorded continuously, and it changes smoothly from 20°C to 80°C. While being heated, the water temperature passes through every possible values between 20°C and 80°C. This is an example of an analog signal. Analog signals are continuous where all possible values are represented. If the water temperature is measured and recorded only once every minute, the temperature is recorded as in fig. (b). In this case, the recorded temperature is not continuous. Rather, it jumps from point to point, and there are only a finite number of values between 20°C and 80°C say, at an increment of 1°C like 20°C, 21°C, 22°C, and so on. There are exactly 11 values in this case. When a quantity is recorded as a series of distinct (discrete) points, it is said to be sampled. This is an example of a digital signal. Digital signals represent only a finite number of discrete values. Virtually, all naturally occurring physical phenomena are analog signals (e.g., temperature, pressure, velocity, sound).

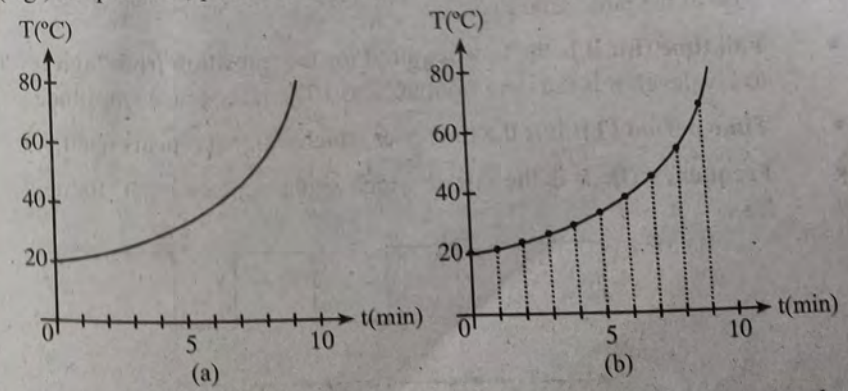


Fig.: (a) An analog (continuous) signal (b) A digital (discrete) signal.

Advantages of digital systems:

1. Digital data can be transmitted more efficiently and is more reliable than analog data.
2. Digital data can be stored in memory.
3. It is less affected from the noise.
4. The transmission is more secure with encryption.

1.2 Digital Waveforms

Digital waveforms consist of voltage levels that are changing back and forth between the HIGH and LOW level as states.

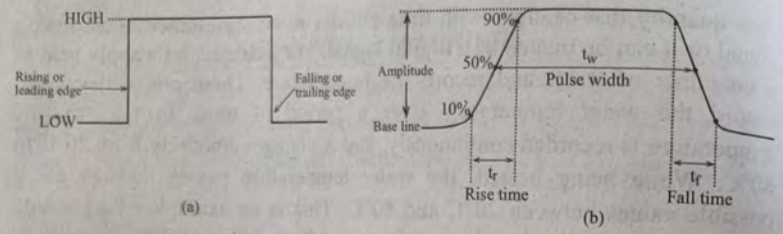
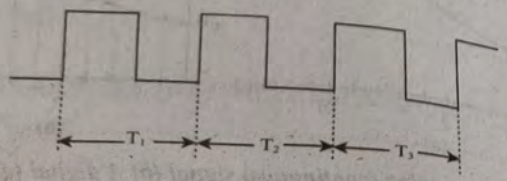


Fig.: (a) Ideal pulse (b) Non-ideal (practical) pulse

Fig. (a) shows that a single positive-going pulse is generated when the voltage (or current) goes from its normally LOW level to its HIGH level and then back to its LOW level. In practice, these transitions never occur instantaneously, although for most digital work, we can assume ideal pulse. Fig. (b) show a non-ideal pulse.

- **Rise time (t_r):** The time required for the pulse to go from its LOW level to its HIGH level is called rise time (t_r). In practice, it is common to measure rise time from 10% of the pulse amplified to 90% of the pulse amplitude.
- **Fall time (t_f):** It is the time required for the transition from high level to low level. It is the time from 90% to 10% of the pulse amplitude.
- **Time period (T):** It is the time over which a signal repeats itself.
- **Frequency (f):** It is the rate at which signal repeats itself. Its unit is Hz.



Period = $T_1 = T_2 = T_3 = \ldots\ldots\ldots T_n$

Frequency (f) = $\frac{1}{T}$

Duty cycle: Duty cycle is the ratio of the pulse width (t_w) to the period and can be expressed as percentage.

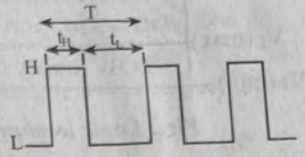


Fig.: Asymmetric signal with period T.

Duty cycle for H = $\frac{t_H}{T} \times 100\%$

Duty cycle for L = $\frac{t_L}{T} \times 100\%$

1.3 Digital Integrated Circuits and Signal Levels

A digital IC is constructed by an interconnection of resistors, transistors, and perhaps small capacitor, all of which have been formed on the surface of silicon wafer. The entire circuit resides on a tiny piece of semiconductor material called a chip.

The two digits in the binary system '1' & '0' are called bits, which is a contraction of the words "binary digit". In digital circuit, two different voltage levels are used to represent the two bits. Generally, 1 is represented by the higher voltage, which is referred to as HIGH and '0' is represented by the lower voltage, which is referred to as LOW. This is called positive logic. If in another system, '1' is represented by LOW and '0' is represented by HIGH, it is negative logic. The voltage used to represents a '1' & '0' are called logic levels. Ideally, one voltage level represents a HIGH (e.g., 5V) and another level represents a LOW (e.g., 0V).

Particularly, HIGH is the voltage between specified minimum value and maximum value, and LOW is the voltage between specified minimum and maximum value. There is no overlap between HIGH levels and LOW levels.

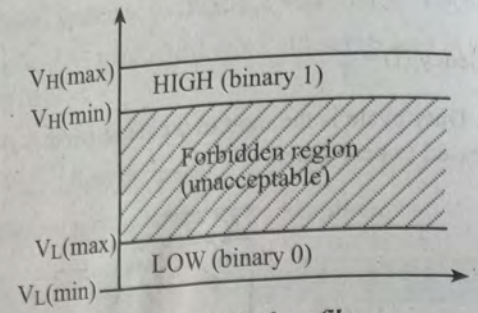


Fig.: Logic level profile

Nowadays, the majority of digital circuit families utilize a single +5V DC power supply, and the two voltage levels are used are +5V DC and 0V DC.

1.4 Number System

Computers communicate and operate in binary digits 0 and 1; on the other hand, human beings generally use the decimal system with ten digits, from 0 to 9. Other number systems are also used, such as octal with eight digits, from 0 to 7, and hexadecimal (Hex) with digits from 0 to 15. In the hexadecimal system, digits 10 through 15 are designated as A through F, respectively, to avoid confusion with the decimal numbers 10 to 15.

A positional scheme is usually used to represent a number in any of the number systems. This means that each digit will have its value according to its position in a number. The number of digits in a position is also referred to as base or radix. For example, the binary system has base 2, the decimal system has base 10, and the hexadecimal system has base 16.

1.4.1 Different Number Systems

1. **Decimal number system (base or radix 10)**

   It uses 10 digits i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. This means it uses 10 different symbols to represent the number.

   $$\overset{3\ 5\ 1\ 0}{6\ 5\ 2\ 3} = 6\times10^3 + 5\times10^2 + 2\times10^1 + 3\times10^0$$

   $$\overset{1\ 0\ 1\ -2}{6\ 5\ 2\ 3} = 6\times10^1 + 5\times10^0 + 2\times10^{-1} + 3\times10^{-2}$$

2. **Binary number system (radix 2)**

   It uses only 2 different symbols to represent number i.e., 0 and 1.

   $$\overset{6\ 3\ 4\ 3\ 2\ 1\ 0}{1\ 1\ 0\ 1\ 1\ 1\ 1} = 1\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^4 + 1\times2^2 + 1\times2^0$$

   $$\overset{3\ 2\ 0\ -1\ -2\ -3}{1\ 1\ 1\ 1\ 1\ 1} = 1\times2^3 + 1\times2^2 + 0\times2^2 + 0\times2^1 + 1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3}$$

3. **Octal number system (base 8)**

   This system uses 8 different symbols i.e., 0, 1, 2, 3, 4, 5, 6, 7

   $$\overset{3\ 2\ 1\ 0}{5\ 7\ 3\ 2} = 5\times8^3 + 7\times8^2 + 3\times\ 8^1 + 2\times8^2$$

4. **Hexadecimal number system (base 16)**

   This system uses 16 different symbols. The first 10 digits is same as in decimal number system and the latter A, B, C, D, E, and F are used for digits 10, 11, 12, 13, 14, and 15 respectively.

   $$\overset{3\ 2\ 1\ 0}{C\ 2\ 7\ 6} = 12\times16^3 + 2\times16^2 + 7\times16^1 + 6\times16^0$$

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

1.4.2 Base Conversion Methods

- **Binary to decimal conversion**

$$(1\,4\,0\,0\,1)_2 = 1\times2^4 + 1\times2^3 + 0\times2^1 + 1\times2^0$$
$$= 16 + 8 + 1$$
$$= (25)_{10}$$

- **Octal to decimal conversion**

$$(3\,4\,7)_8 = 3\times8^2 + 4\times8^1 + 7\times8^0$$
$$= 3\times64 + 4\times8 + 7$$
$$= (231)_{10}$$

- **Hexadecimal to decimal conversion**

$$(8\,6\,5\,F)_{16} = 8\times16^3 + 6\times16^2 + 5\times16^1 + F\times16^0$$
$$= 11\times16^3 + 6\times16^2 + 5\times16^1 + 15\times16^0$$
$$= (46687)_{10}$$

- **Decimal to binary conversion**

For conversion of $(13)_{10}$ to binary,

Quotient	Remainder
13÷2 = 6	1
6÷2 = 3	0
3÷2 =1	1
1÷2 =0	1
$\therefore (13)_{10} = (1101)_2$	

In case of a number with both integer and fraction part, for integer, repeated division by 2 is carried out whereas for fraction, repeated multiplication by 2 is carried out.

For example, to convert $(25.625)_{10}$ into its binary equivalents, following operations are performed.

i. **For integer part,**

Quotient	Remainder
25÷2 = 12	1
12÷2 = 6	0
6÷2 = 3	0
3÷2 = 1	1
1÷2 = 0	1
$\therefore (25)_{10} = (11001)_2$	

ii. **For fractional part,**

	Carry
0.625×2 = 1.25	1
0.25×2 = 0.5	0
0.5×2 = 1.0	1
$\therefore (0.625)_{10} = (11001.101)_2$	

$\therefore (25.625)_{10} = (11001.101)_2$

- **Decimal to octal conversion**

To convert a number in decimal to a number in octal, we have to divide the decimal number by 8 unless zero is obtained.

For conversion of $(426)_{10}$ to $(?)_8$,

Quotient	Remainder
426 ÷ 8 = 53	2
53 ÷ 8 = 6	5
6 ÷ 8 = 0	6
$\therefore (426)_{10} = (652)_8$	

For a number with both integer and fraction part, for integer, repeated division by 8 is carried out whereas for fraction, repeated multiplication by 8 is carried out.

For $(0.96)_{10} \rightarrow (?)_8$,

$0.96 \times 8 = 7.68$	7
$0.68 \times 8 = 5.44$	5
$0.44 \times 8 = 3.52$	3
$0.52 \times 8 = 4.16$	4
$\therefore (0.96)_{10} = (0.7534)_8$	

- **Decimal to hexadecimal conversion**

For a decimal number with only integer part, we have to divide the decimal number by 16 repeatedly until zero is obtained.

For conversion of $(348)_{10}$ to $(?)_{16}$,

$348 \div 16 = 21$	12 (C)
$21 \div 16 = 1$	5
$1 \div 16 = 0$	1
$\therefore (348)_{10} = (15C)_{16}$	

For a decimal number with only fraction part, repeated multiplication of the number by 16 is carried out until certain level of accuracy is achieved as shown in the following example.

For $(0.37)_{10} \rightarrow (?)_{16}$,

	Carry
$0.37 \times 16 = 5.92$	5
$0.92 \times 16 = 14.72$	14 (E)
$0.72 \times 16 = 11.52$	11 (B)
$0.32 \times 16 = 3.32$	3
$0.32 \times 16 = 5.12$	5
$0.12 \times 16 = 1.92$	1
$\therefore (0.37)_{10} = (0.5EB851)_{16}$	

- **Hexadecimal to binary**

$(2BF.29B)_{16} \rightarrow (\ )_2$

The conversion from hexadecimal to binary number is achieved by replacing each hexadecimal digit with its 4-bit binary equivalent.

2 B F 2 9 B

↓ ↓ ↓ ↓ ↓ ↓

0010 1011 1111 0010 1001 1011

$= (00101011\ 1111.00\ 10100011011)_2$

$= (1010\ 111111\ .0010100\ 11011)_2$

$(1CD.2A)_{16} \rightarrow (\ )_2$

The conversion is accomplished by grouping the binary number into groups of 4-bits each, starting from the **binary point** and proceeding to the right as well as to the left. Each group is then replaced by its hexadecimal equivalent.

1 C D 2 A

↓ ↓ ↓ ↓ ↓

0001 1100 1101 0010 1010

$= (000111\ 001101.00101010)_2$

$= (111\ 00\ 1101.\ 001\ 0101)_2$

- **Binary to hexadecimal**

$(110\ 110111.\ 1011110)_2 \rightarrow (\ )_{16}$

0001 1011 0111 1011 1100

↓ ↓ ↓ ↓ ↓

1 B 7 B C

$= (1B7.BC)_{16}$

- **Octal to binary**

$(21)_8 \rightarrow (\ )_2$

2 1

↓ ↓

010 001

$= (0100001)_2 = (10001)_2$

$(1745.246)_8 \rightarrow (\ )_2$

1 7 4 5 2 4 6

↓ ↓ ↓ ↓ ↓ ↓ ↓

001 111 100 101 010 100 110

$= (001111100101.010100110)_2$

$= (1111100101.01010011)_2$

- **Binary to octal**

  $(10110001101011.111100\ 00011)_2 \rightarrow (\ )_8$

010	110	001	101	011	1111	100	000	110
↓	↓	↓	↓	↓	↓	↓	↓	↓
2	6	1	5	3	7	4	0	6

  $= (26153.7406)_8$

- **Octal to hexadecimal**

  $(1745.246)_8 \rightarrow (\ )_{16}$

  Ans: $(3E5.530)_{16}$

  Hint: First change to binary system & then to hexadecimal system

- **Hexadecimal to octal**

  $(3E5.530)_{16} \rightarrow (\ )_8$

  Ans: $(1745.246)_8$

  Hint: First change to binary system and then to octal system

1.5 Binary-Coded Decimal (BCD)

Binary codes for decimal digits require a minimum of 4 bits. It is a straight assignment of binary equivalent to each digit. For example, 95 when converted into binary is equal to 1011111. But, when same number is represented in BCD code, each decimal digit is represented by 4-bits as: 1001 for 9 and 0101 for 5. Thus, the BCD equivalent will be 1001 0101.

There are only ten code groups in the BCD system i.e., 000, 0001, 0010, 0011, 0100, 0101, 0110, 0110, 0111, 1000, 1001.

Decimal	BCD
0	0000
1	0001
9	1001
10	0001 0000
10	0001 0000
11	0001 0001
99	10011001
100	0001 0000 0000

Merits:

a. It is easy to encode and decode decimals into BCD and vice versa.

b. It is also simple to implement a hardware algorithm for the BCD converter.

c. It is very useful in digital systems whenever decimal information is given either as inputs or displayed as outputs.

Demerits:

a. For integers greater than 9, BCD occupies more space than the corresponding binary. For example,

   Decimal → 13

   Binary → 1101

   BCD code → 0001 0011

b. The arithmetic operations using BCD code require a complex design of arithmetic and logic unit (ALU) than the straight binary number system.

1.6 ASCII Code

ASCII is the abbreviation for American Standard Code for Information Interchange. ASCII is a universally accepted alpha-numerical code used in most computers and other electronic equipments. Most computer keyboards are standardized with the ASCII code. When we enter a letter, a number or control command, the ASCII code goes into the computer. ASCII code is a 7-bit code; so, it can represent 128 characters.

'a' = 97_{10} = 1100001 , '1' = 0110001 , 'A' = 65_{10} = 1000001

1.7 EBCDIC as Alphanumeric Code

EBCDIC stands for Extended Binary Coded Decimal Interchange Code. It is an eight-bit code and primarily used in IBM made devices. The bit assignments of EBCDIC are different from the ASCII but the character symbols are the same.

1.8 Excess-3 Code

It is an important 4-bit code sometimes used with binary-coded decimal (BCD) numbers. To convert any decimal number into its excess-3 form, we have to add 3 to each decimal digit, and then convert the sum to a BCD number.

Example: Convert 12 to an excess-3 number.

⇒ **First, add 3 to each decimal digit i.e.,**

$$\begin{array}{cc} 1 & 2 \\ +3 & +3 \\ \hline 4 & 5 \end{array}$$

Second, convert the sum to BCD form.

$$\begin{array}{cc} 4 & 5 \\ \downarrow & \downarrow \\ 0100 & 0101 \end{array}$$

∴ 01000101 in the excess-3 code stands for decimal 12.

Example: Convert 29 to an excess-3 code.

⇒

$$\begin{array}{cc} 2 & 9 \\ +3 & +3 \\ \hline 5 & 12 \\ \downarrow & \downarrow \\ 0101 & 1100 \end{array}$$

After adding 9 and 3, do not carry the 1 into the next column; instead leave the result intact as 12, and then convert. Therefore, 01011100 in excess-3 code stands for decimal 29.

1.9 The Gray Code

Gray code is unweighted and is not an arithmetic code. In gray code, the bits are arranged in such a way that it changes by only one bit as it sequences from one number to next. A typical application of the gray code occurs in rotational or shaft encoder, which is used to convert some angular position of a shaft to a digital format.

Binary	Gray Code
000	000
001	001
010	011
011	010
100	110

a. Binary to gray code conversion

1. Keep MSB not gray code same as MSB of binary code.
2. Going from left to right, add adjacent pairs of binary code to get next gray code bit.
3. Discard any carries.

Example: Convert 10110 to gray code

$$\begin{array}{ccccccccc} 1 & + & 0 & + & 1 & + & 1 & + & 0 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ 1 & & 1 & & 1 & & 0 & & 1 \end{array}$$

b. Gray to binary conversion

1. Keep MSB of binary code same as of gray code.
2. Add each binary code generated to the gray code bit in the next adjacent.
3. Discard carries.

E.g., Convert 110111 gray code to binary.

⇒

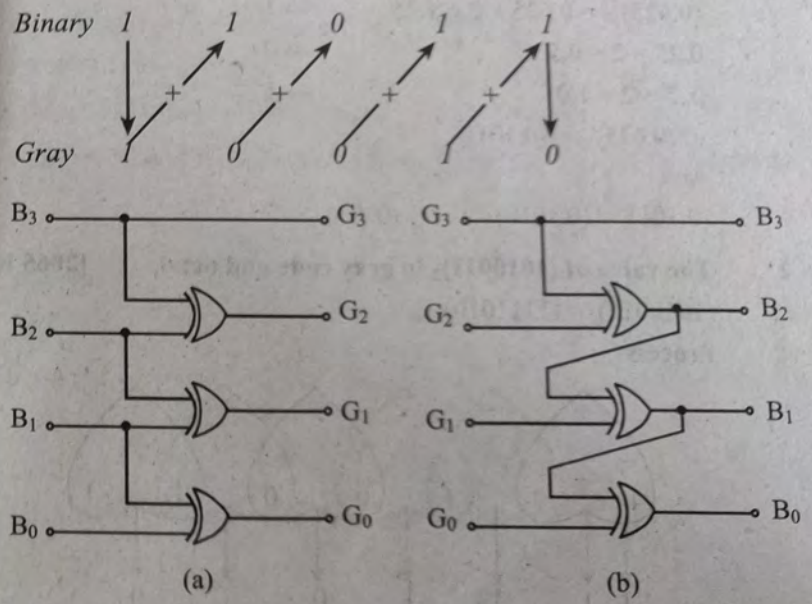


Fig.: (a) Binary to gray converter (b) Gray to binary converter

O →

Zone no.	0	1	2	3	4	5	6	7
Sensor ABC (Binary coded)	000	001	010	011	100	101	110	111

O →

Zone no.	0	1	2	3	4	5	6	7
Sensor ABC (Gray coded)	000	001	011	010	110	111	101	100

Fig.: Object moving along a track with sensor

Let an object move from one zone to another. Let the presence of the objec in one zone is sensed by sensor ABC. Sensor is in zone 0 is represented by ABC = 000, zone 1 is ABC = 001, and so on. Let's suppose the sensor move from zone 1 to zone 2, then both BC has to change to sense that movement Suppose, sensor B (may be an electro-mechanical switch) reacts lightly late than sensor C. This problem can be more prominent if the object moves from zone 3 to zone 4 when all three sensors has to change its value. Note that, if zones are gray coded such problem does not appear as between two consecutive zones only one sensor changes its value.

MORE WORKED OUT EXAMPLES:

1. Convert $(0.625)_{10}$ into binary convert the resulting binary number into hexadecimal. [2064 Falgun]

⇒ $(0.625)_{10} = 0.625 \times 2 = 1.25 \rightarrow 1$

$0.25 \times 2 = 0.5 \rightarrow 0$

$0.5 \times 2 = 1.0 \rightarrow 1$

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

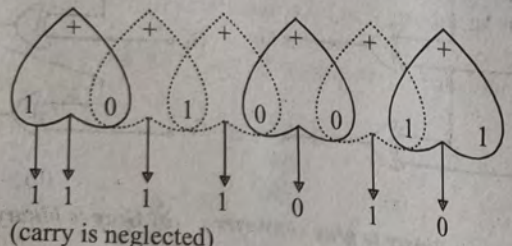
And

$(0.101)_2 = (0.1010)_2 = (0.A)_{16}$

2. The value of $(1010011)_2$ in gray code and octal. [2065 Falgun]

⇒ $(1010011)_2 = (1111010)_{Gray}$

Process



(carry is neglected)

And $(1010011)_2$

$= (001\ 010\ 011)_2 = (123)_8$

3. What is weighted code and non-weighted code? What will be the BCD, Excess-3, and Gray code for the decimal number 15? [2071 Chaitra]

⇒ **Weighted codes:** The number system which is positionally weighted or the digit position of any number represents a specific weight is known as weighted codes. Examples include decimal, binary, octal, etc.

Non- weighted codes: The number system which is not positionally weighted or the digit position of any numbers does not represent the specific weight is known as non-weighted codes. Examples include excess-3 code, gray code.

BCD of $(15)_{10} = (00010101)_{BCD}$

For excess-3 code,

$(15)_{10}$

$$\begin{array}{r} 1\ 5 \\ 3\ 1 \\ \hline 4\ 6 \\ \downarrow\ \downarrow \\ 0100\ 0110 \end{array}$$

$\therefore (15)_{10} = (01000110)_{ex\text{-}3}$

For gray code,

$(15)_{10} = (1111)_2 \rightarrow (1000)_{gray\ code}$

4. Convert 37.432 decimal number to binary. [2074 Chaitra]

⇒

	Remainder
$37 \div 2 = 18$	1
$18 \div 2 = 9$	0
$9 \div 2 = 4$	1
$4 \div 2 = 2$	0
$2 \div 2 = 1$	0
$1 \div 2 = 0$	1

	Carry
$0.432 \times 2 = 0.864$	0
$0.864 \times 2 = 1.728$	1
$0.728 \times 2 = 1.456$	1
$0.456 \times 2 = 0.912$	0
$0.912 \times 2 = 1.824$	1

$\therefore 37.432 = (100101.01101)_2$

5. **Convert decimal 39 into binary and hexadecimal.** [2074 Ashwin]

⇒ **39 into binary**

	Remainder
$39 \div 2 = 19$	1
$19 \div 2 = 9$	1
$9 \div 2 = 4$	1
$4 \div 2 = 2$	0
$2 \div 2 = 1$	0
$1 \div 2 = 0$	1

$\therefore \quad 39_{10} = 100111_2$

39 into hexadecimal

	Remainder
$39 \div 16 = 2$	7
$2 \div 16 = 0$	2

$\therefore \quad 39_{10} = 27_{16}$

Chapter – 2

DIGITAL LOGIC

2.1 Logic Gates and Logic Operators

Logic means certain declarative statement is true if certain conditions are fulfilled and false if does not meet that. The manner in which the digital circuit responds to an input is referred to as circuit's logic.

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one inputs and only one output. The relationship between the input and the output is based on certain logic. Based on logic, the gates are named as NOT gate, AND gate, OR gate, NAND gate, NOR gate, etc.

In the decimal system, which is used in our day-to-day life, the arithmetic operations such as addition, subtraction, multiplication, division, square, square root, modulus, etc. are used to solve equations. Similarly, to solve or simplify the logical expressions, used in digital circuits, we need to use logical operators. The three main logic operators may be listed as:

i. AND operator (A. B → logical multiplication)

ii. OR operator (A+B → logical addition)

iii. NOT operator ($\overline{A}$ → logical inversion)

The operation of a logic gate can be best understood with the help of a table called "truth table". The truth table consists of all the possible combinations of the inputs and the corresponding state of output of a logic gate. The relationship between the inputs and the outputs of a gate can be expressed mathematically by means of Boolean expression.

Logic gates are classified into three categories namely, the basic gates, the universal gates, and the special purpose gates.

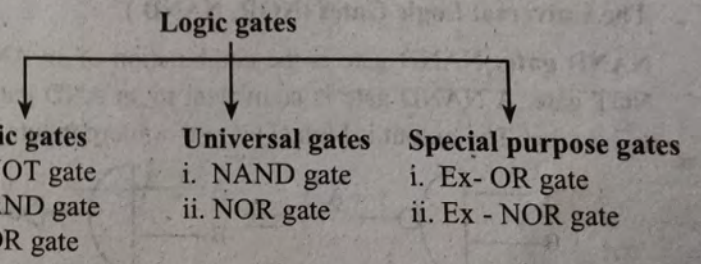


Fig.: Classification of gates.

2.2 Classification of Logic Gates

2.2.1 The Basic Gates (NOT, OR, AND)

i. **NOT gate or inverter:** The NOT gate is a logic gate having one input (A) and one output (Y). It is also known as inverter because its output is the inverted version of its input.

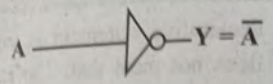


Fig.: Symbol of NOT gate.

ii. **AND gate:** AND gate performs the logical multiplication on its input. The output is high (Y = 1) if and only if all the inputs to the gate are high (1). The output is low (0), if at least one of the inputs is low (0). AND gate can have two or more inputs and only one output.

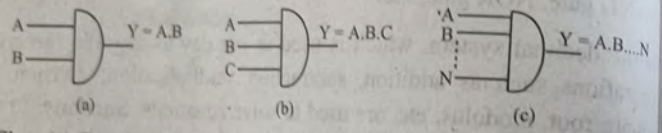


Fig.: (a) Two input AND gate (b) Three input AND gate. (c) N input AND gate.

iii. **OR gate:** OR gate perform the logical addition on its inputs. The output will be high (1) if anyone input is high. The output will be low (0) if and only if all inputs are simultaneously low (0).

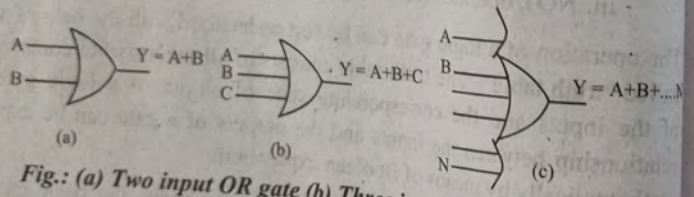


Fig.: (a) Two input OR gate (b) Three input OR gate. (c) N input OR gate.

2.2.2 The Universal Logic Gates (NOR, NAND)

i. **NAND gate:** NAND gate is the combination of an AND gate and a NOT gate. A NAND gate is equivalent to an AND gate followed by an inverter. The output is high (1) if one or more inputs are low (0).

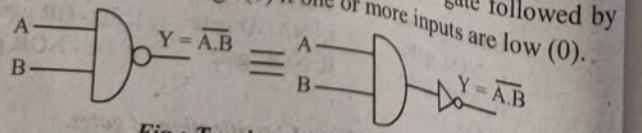


Fig.: Two input NAND gate

ii. **NOR gate:** NOR gate is the combination of an OR gate and a NOT gate. It is opposite of OR gate. NOR gate is equivalent to an OR gate followed by an inverter. The output is low (0) if one or more inputs are high (1).

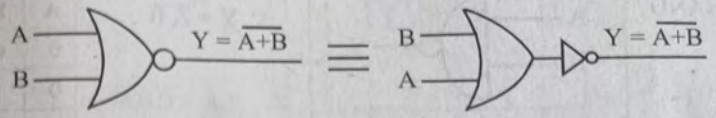


Fig.: Two input NOR gate

2.2.3 The Special Purpose Logic Gates (X-OR, X-NOR)

i. **Exclusive OR gate (X-OR):** The output is high if inputs are different and low if both inputs are same.

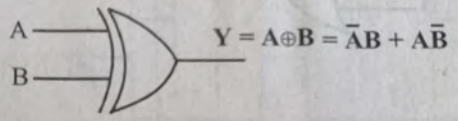


Fig.: Two input X-OR gate

ii. **Exclusive NOR gate (X-NOR):** The output is high if both inputs are same and low if inputs are different.

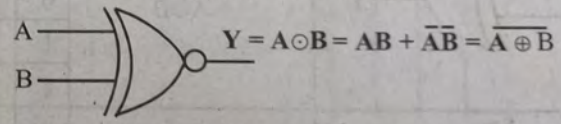
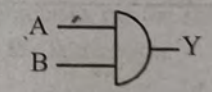
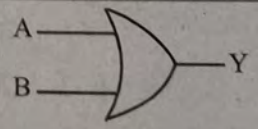
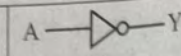
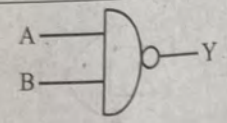
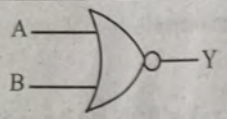
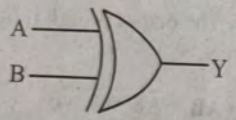
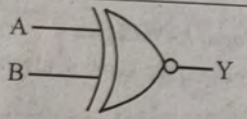


Fig.: Two input X-NOR gate

The following table illustrates the graphic symbol, algebraic function, and truth table of each gate.

Name	Graphic symbol	Algebraic function	Truth table		
AND		Y = A.B	**A**	**B**	**Y**
			0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		Y = A + B	**A**	**B**	**Y**
			0	0	0
			0	1	1
			1	0	1
			1	1	1

NOT		$Y = \overline{A}$	A	Y	
			0	1	
			1	0	
NAND		$Y = \overline{A.B}$	A	B	Y
			0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$Y = \overline{A+B}$	A	B	Y
			0	0	1
			0	1	0
			1	0	0
			1	1	0
X – OR		$Y = \overline{A}B + A\overline{B}$ $= A \oplus B$	A	B	Y
			0	0	0
			0	1	1
			1	0	1
			1	1	0
X – NOR		$Y = AB + \overline{A}\,\overline{B}$ $= \overline{A \oplus B}$ $= A \odot B$	A	B	Y
			0	0	1
			0	1	0
			1	0	0
			1	1	1

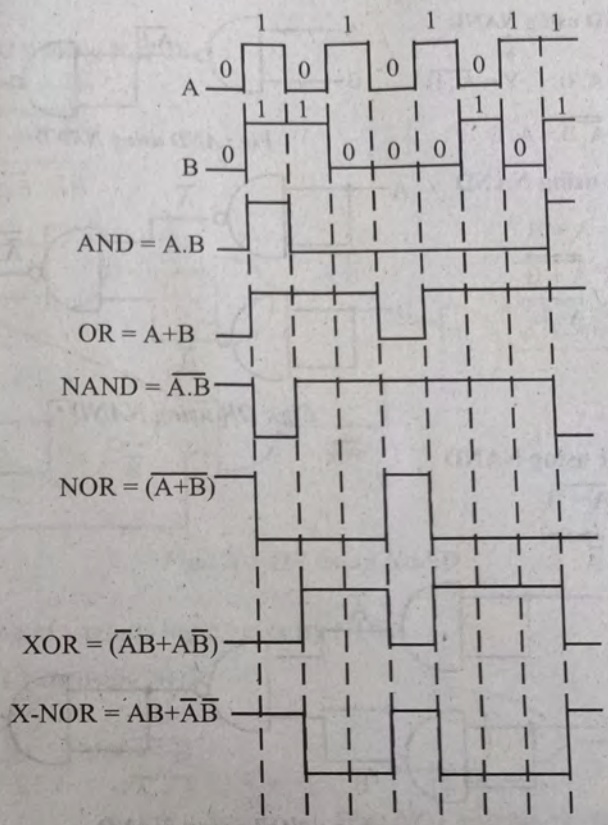


Fig.: Pulse operation

2.3 Universal Gates- NAND, NOR

NAND gates and NOR gates are called universal gates because any type of gates or logic functions can be implemented by these gates. They are small in size and easy for fabrication.

Realization of various logic gates by NAND:

i. NOT gate using NAND

$Y = \overline{A \cdot B}$

$= \overline{A \cdot A}$ $[\because A = B]$

$= \overline{A}$

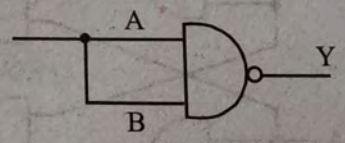


Fig.: NOT gate using NAND

ii. AND using NAND

$Y = A.B$ $\quad Y = \overline{A.B}$

$Y = \overline{\overline{A.B}} = A.B$

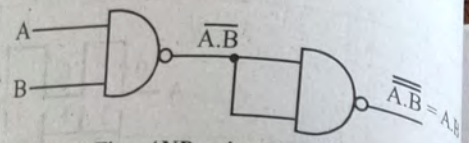


Fig.: AND using NAND

iii. OR using NAND

$Y = A + B$

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

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

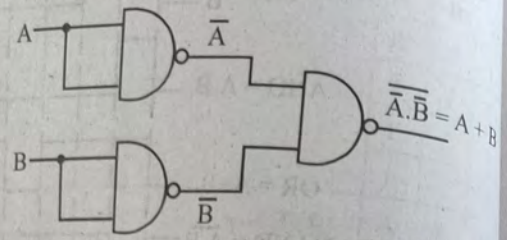


Fig.: OR using NAND

iv. NOR using NAND

$Y = \overline{A + B}$

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

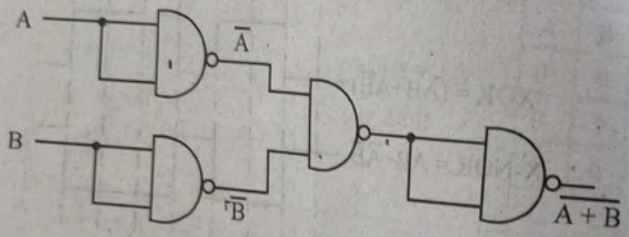


Fig.: NOR using NAND

v. X-OR using NAND

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

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

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

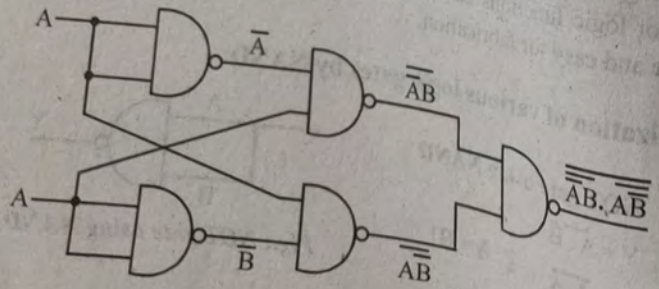


Fig.: X-OR using NAND

vi. X-NOR using NAND

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

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

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

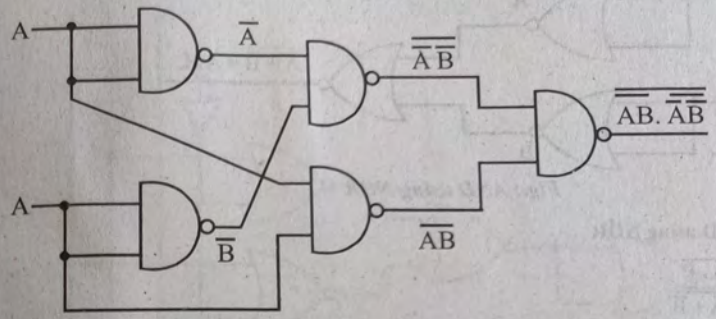


Fig.: X-NOR using NAND

Realization of various logic gates by NOR:

i. NOT gate using NOR

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

$= \overline{A + A}$ [$\because A = B$]

$= \bar{A}$

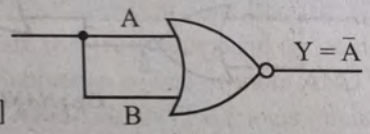


Fig.: NOT gate using NOR

ii. OR using NOR

$Y = A + B$

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

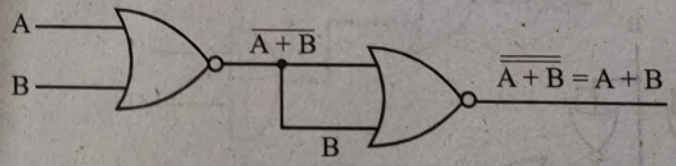


Fig.: OR using NOR

iii. **AND using NOR**

$Y = AB$

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

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

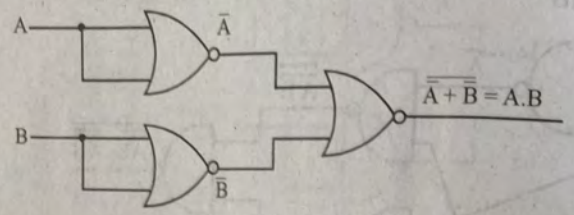


Fig.: AND using NOR

iv. **NAND using NOR**

$Y = \overline{A.B}$

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

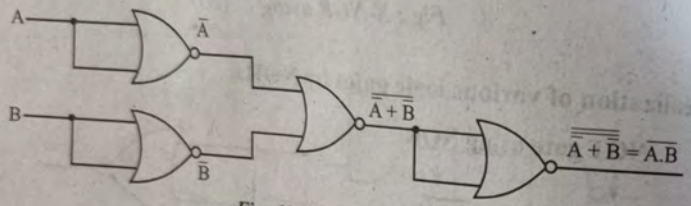


Fig.: NAND using NOR

v. **X-OR using NOR**

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

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

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

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

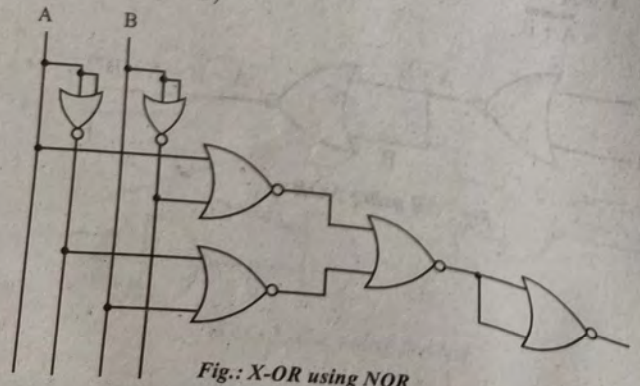


Fig.: X-OR using NOR

vi. **X-NOR using NOR**

$Y = AB + \overline{A}\,\overline{B}$

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

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

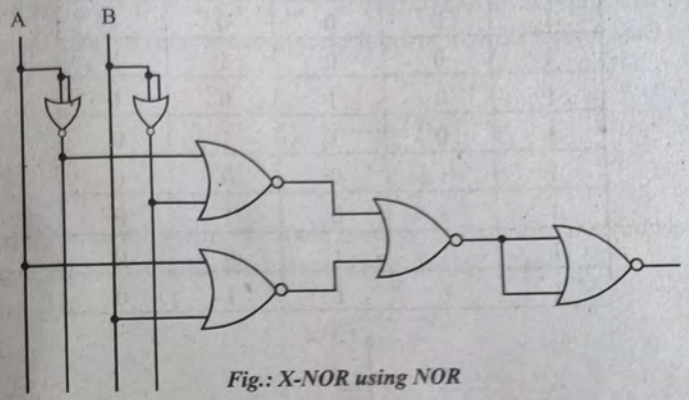


Fig.: X-NOR using NOR

2.4 AND-OR-INVERT Gates

AND-OR-INVERT (AOI) logic gates are two level compound (or complex) logic function constructed from the combination of one or more AND gates followed by a NOR gate. For a 4 input AND-OR- INVERT logic circuit, the output is low if both inputs A and B are high or both inputs C and D are high.

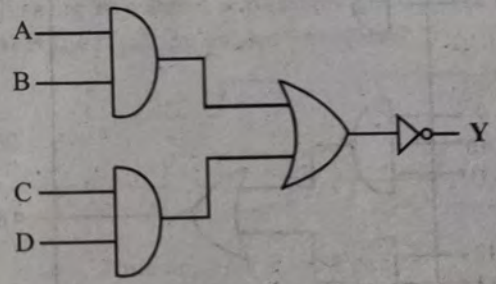


Fig.: AND-OR-INVERT gates

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0

A	B	C	D	Y
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

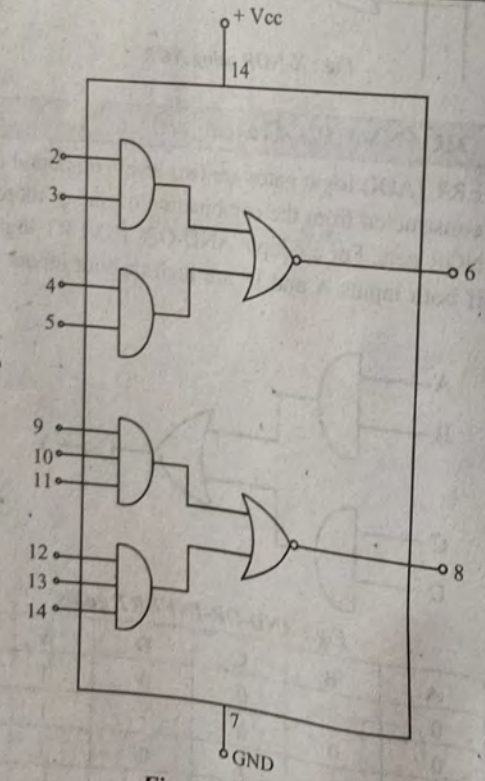
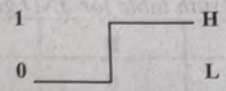


Fig.: AND-OR-INVERT IC

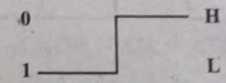
2.5 Positive and Negative Logic

The binary signals at the inputs or outputs of any gate may be one of two values, except during transitions. One signal value represents logic 1, and the other represents logic 0.

For a positive logic system, the most positive voltage level represents logic 1 state or high level (H) and lowest voltage level represents logic 0 state or low level (L).



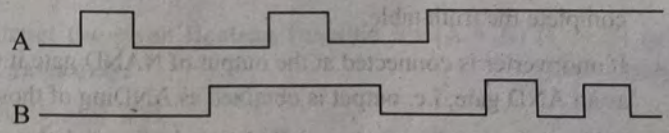
For a negative logic system, the most positive voltage level represents logic 0 state and the lowest voltage level represents logic 1 state.



Example: If the voltage levels are –1 volt and –10 volt, then in a positive logic system, –1 volt represents logic 1 and –10 volt represents logic 0, and in negative logic system, –1 volt represents logic 0 and –10 volt represents logic 1.

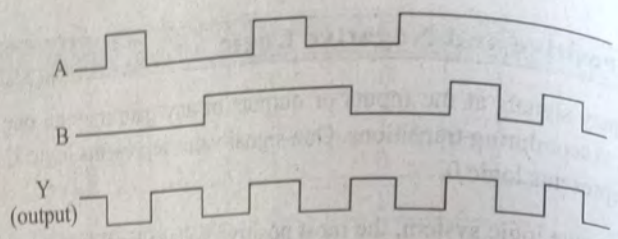
MORE WORKED OUT EXAMPLES:

1. If the following two input waveforms are applied to an X-NOR gate, what is the resulting output waveform? [2066 Bhadra]



⇒ *Truth table for X-NOR gate*

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



2. When FF_H is ANDed with CO_H, what will be the result number? [2065 Shrawa

⇒

Truth table for AND gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$FF_H = (1111\ 1111)_2$

$CO_H = (1100\ 0000)_2$

When FF_H is ANDed with CO_H, the result is 1100 0000

Answer = CO_H

3. A NAND gate has seven inputs. How many numbers of inpu combinations you need to complete the truth table of such gate? I an inverters is connected at its output then what logic function will perform? [2064 Jestha

⇒ Number of combination, $N = 2^n = 2^7 = 128$

Hence, if 7 inputs are provided, 128 combinations are required t complete the truth table.

If an inverter is connected at the output of NAND gate it will perfor as an AND gate, i.e. output is obtained as ANDing of those inputs.

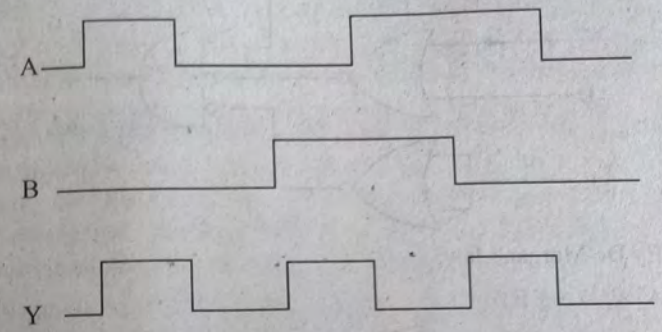
4. If the two waveforms A and B shown in figure below are applie to exclusive-OR gate, determine the resulting waveform.

⇒ **Truth table for exclusive-OR gate [2064 Jestha**

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = A \oplus B$

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



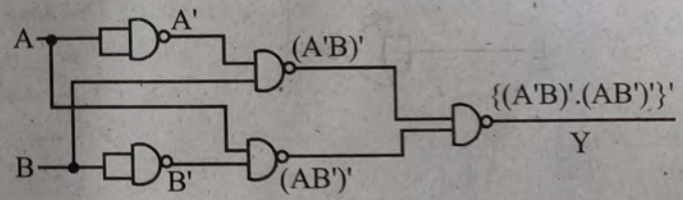
5. Implement exclusive-OR gate by using NAND gates only. [2071 Chaitra]

⇒ The logic expression of XOR = AB' + A'B.

$Y = A'B + AB'$

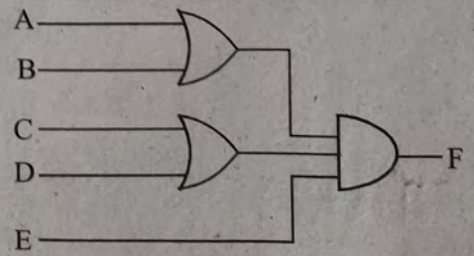
$= (A'B + AB')''$

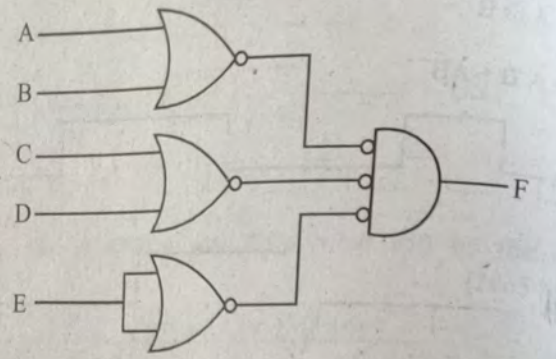
$= \{(A'B)'.(AB')'\}'$



6. Construct the given Boolean function F= (A + B) (C + D) using NOR gates only. [2069 Chaitra]

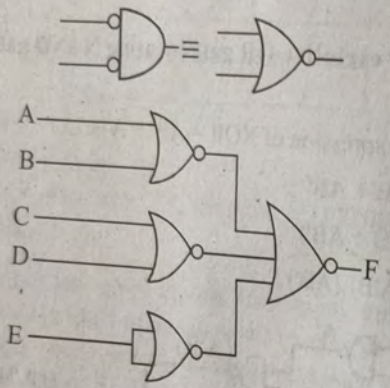
⇒ F = (A+B) (C + D)E





By De-Morgan's law,

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



Chapter – 3

COMBINATIONAL LOGIC CIRCUITS

3.1 Boolean Laws and Theorems

Boolean algebra is used to analyze and simplify the digital circuits. Because it uses only the binary numbers i.e., 0 and 1, it is also called "binary algebra" or "logical algebra". The rules of Boolean algebra are different from those of the conventional algebra in the following manner:

i. Symbols used in Boolean algebra (usually letters) do not represent numerical values.

ii. Arithmetic operations like subtraction, division are not performed. Also, there are no fractions, negative numbers, square, square root, logarithms, imaginary number, etc.

iii. The third and most important point is that Boolean algebra allows only two possible values (0 or 1) for any variable.

The Boolean laws are:

1. **Commutative law**: This law allows change in the position of the input variables in OR and AND expression.

   i. $A + B = B + A$

   ii. $A.B = B.A$

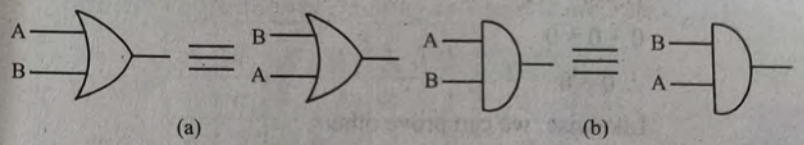


Fig.: (a) Illustration of (i) (b) Illustration of (ii)

2. **Associative law:** The law states that it make no difference in what order the variables are grouped when ORing or ANDing more than two variables.

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

   ii. $A.(B.C) = (A.B).C$

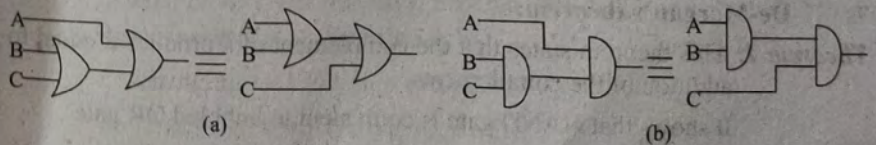


Fig.: (a) Illustration of (i) (b) Illustration of (ii)

3. **Distributive law:** This law permits factoring or multiplying out expression.

i $A(B+C) = AB + AC$

ii. $AB + AC = A(B+C)$

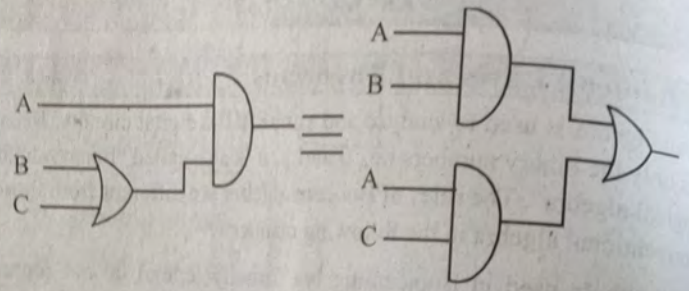


Fig.: Illustration of (i) and (ii)

4. **OR law:**

i. $A + 0 = A$

ii. $A + 1 = 1$

iii. $A + A = A$

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

For law (i), A can be 0 or 1

Let $A = 0$

$A + 0 = A$

$0 + 0 = 0$

$\therefore 0 = 0$

Likewise, we can prove others.

5. **AND law:**

i. $A.0 = 0$

ii. $A.1 = A$

6. **Inversion law:**

$A = \bar{\bar{A}}$

7. **De-Morgan's theorem:**

Theorem 1: This theorem states that the complement of a product is equal to addition of the complements.

It shows that NAND gate is equivalent to bubbled OR gate.

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

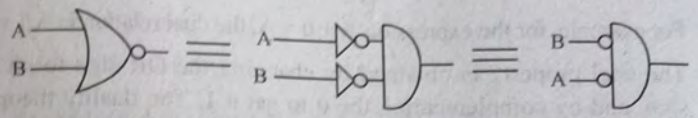


Fig.: Illustration of De-Morgan's first theorem.

Table: Verification of De-Morgan's first theorem

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

Theorem 2: This theorem states that the complement of a sum is equal to the product of complements.

It shows that NOR gate is equivalent to bubbled AND gate.

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

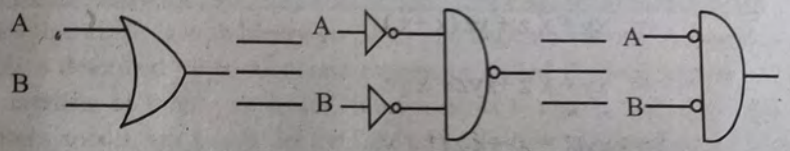


Fig.: Illustration of De-Morgan's second theorem.

Table: Verification of De-Morgan's second theorem

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

8. **Duality theorem**

The duality theorem is one of those elegant theorems proved in advanced mathematics. We will state the theorem without proof. Here is what duality theorem says. Starting with a Boolean relation, we can derive another Boolean relation by:

i. changing each OR sign to an AND sign

ii. changing each AND sign to an OR sign

iii. complementing any 0 or 1 appearing in the expression.

For example, for the expression $A + 0 = A$, the dual relation is $A.1 = A$.

The dual property is obtained by changing the OR sign to an AN sign, and by complementing the 0 to get a 1. The duality theorem useful because it sometimes produce a new Boolean relation.

For example, $A(B + C) = AB + AC$

By changing each OR and AND operation, we get the dual relation

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

EXAMPLE:

1. Simplify using Boolean algebra:

i. $\bar{x}\,\bar{y}\,z + \bar{x}\,yz + x\bar{y}$

$= \bar{x}\,z(\bar{y} + y) + x\bar{y}$

$= \bar{x}\,z + x\bar{y}$ [$\because y + \bar{y} = 1$]

ii. $xy + \bar{x}\,z + yz$

$= xy + \bar{x}\,z + yz(x + \bar{x})$

$= xy + \bar{x}\,z + xyz + \bar{x}\,yz$

$= xy(1 + z) + \bar{x}\,z(1 + y)$

$= xy + z\bar{x}$ [$\because 1 + z = 1$ and $1 + y = 1$]

iii. $\bar{x}\,\bar{y}\,\bar{z} + \bar{x}\,y\bar{z} + xy\bar{z} + x\,\bar{y}\,\bar{z}$

$= \bar{x}\,\bar{z}\,(\bar{y} + y) + x\bar{z}\,(\bar{y} + y)$

$= \bar{x}\,\bar{z} + x\bar{z}$

$= \bar{z}\,(\bar{x} + x)$

$= \bar{z}$

iv. $AB + A(B + C) + B(B + C)$

$= AB + AB + AC + B.B + BC$

$= AB + AC + B + BC$

$= AB + AC + B(1 + C)$

$= AB + AC + B$

$= B(1 + A) + AC$

$= B + AC$

2. Prove that $A(\bar{A}\,C)(\bar{A}\,B + C)(\bar{A}\,BC + \bar{C}) = 0$

$\Rightarrow$ L.H.S. $= A(\bar{A} + C)(\bar{A}\,B + C)(\bar{A}\,BC + \bar{C})$

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

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

$= (A\,\bar{A}\,BC + ACC)(\bar{A}\,BC + \bar{C})$

$= (A\bar{A}\,BC + ACC)(\bar{A}\,BC + \bar{C})$

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

$= A\,\bar{A}\,BC\,C + AC\,\bar{C}$

$= 0 + 0$

$= 0$ Hence, proved.

3.2 Boolean Expressions and Boolean Functions

Boolean algebra deals with binary variables and logic operations. A Boolean function is described by an algebraic expression called Boolean expression which consists of binary variables, the constants 0 and 1, and the logic operation symbols. Let's consider the following illustrative examples.

F (A,B,C,D)	=	$A + B\bar{C} + ADC$
Boolean function		Boolean expression

Any logic expression may be expressed in two standard forms: Sum-of-products (SOP) form, and product-of-sums (POS) form.

1. Sum-of-products (SOP)

When two or more product terms (or ANDed terms) are summed by Boolean algebra, the resulting expression is sum-of-products.

E.g., $F = AB + \bar{A}\,\bar{B}\,C$

$F = ABC + C\bar{D} + \bar{B}\,\bar{C}\,D$

SOP expression is two or more AND functions ORed together. SOP form can also contain a term with a single variable such as $A + \bar{A}\,\bar{B} + \bar{B}\,C$.

In SOP representation, 0 represents $\bar{A}$ and 1 represents A. Only 1s output is taken.

Standard SOP (or canonical SOP)

Standard SOP is one in which all the variables in the domain appear in each product term in the expression.

$AB + B\,C + A\bar{D}$ is not standard SOP.

$A\bar{B}\,CD + ABCD + AB\bar{C}\,\bar{D}$ is standard SOP.

Each individual product term in standard SOP form is called a "minterm" and is represented by 'm'. Standardization makes evaluation, simplification, implementation, realization of Boolean expression much more systematic and easier.

Minterms for three variables (A, B, C)

A	B	C	Term	Designation (m_i)
0	0	0	$\bar{A}\bar{B}\bar{C}$	m_0
0	0	1	$\bar{A}\bar{B}C$	m_1
0	1	0	$\bar{A}B\bar{C}$	m_2
0	1	1	$\bar{A}BC$	m_3
1	0	0	$A\bar{B}\bar{C}$	m_4
1	0	1	$A\bar{B}C$	m_5
1	1	0	$AB\bar{C}$	m_6
1	1	1	ABC	m_7

Consider the following example.

Input			Function output	
A	B	C	F_1	F_2
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

$F_1 = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,BC + AB\bar{C}$

$= m_0 + m_3 + m_6$

$= \Sigma m\,(0, 3, 6)$

$F_2 = \bar{A}\,BC + A\bar{B}\,C + AB\bar{C} + ABC$

$= m_3 + m_5 + m_6 + m_7$

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

Converting SOP to standard SOP:

1. Multiply each product term by $(A + \bar{A})$ where A is the missing term.
2. Expand the multiplication and repeat step 1 until standard SOP is obtained.

EXAMPLE:

i. Convert $A + A\bar{B} + \bar{B}\,C$ to standard SOP.

$\Rightarrow A + A\bar{B} + \bar{B}\,C$

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

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

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

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

ii. Convert A + B to standard SOP.

$\Rightarrow A + B$

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

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

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

2. Product–of–sums form (POS)

When two or more sum terms are multiplied, the resulting expression is product of sum. E.g., $(\bar{A} + B)(A + \bar{B} + \bar{C})(\bar{A} + B + C)$

In terms of logic function, it is the AND function of two or more ORed functions. In POS representation, 0 represents A and 1

represents $\bar{A}$ and only 0's output is taken. A POS expression can also contain a single variable terms. E.g., $A(\bar{B}+\bar{C})(B+C)$

Standard POS

Standard POS is one in which all the variables in the domain appear in each sum term in the expression.

$A(\bar{B}+\bar{C})(B+C)$ is not standard POS

$(A+B+\bar{C})(A+\bar{B}+\bar{C})$ is standard POS.

Each individual sum term in standard POS form is called "maxterm" and is represented by 'M'.

Maxterm of three variables

A	B	C	Term	Designation (M_i)
0	0	0	$A+B+C$	M_0
0	0	1	$A+B+\bar{C}$	M_1
0	1	0	$A+\bar{B}+C$	M_2
0	1	1	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$\bar{A}+B+C$	M_4
1	0	1	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$\bar{A}+\bar{B}+C$	M_6
1	1	1	$\bar{A}+\bar{B}+\bar{C}$	M_7

Consider the same example.

Input			Function output	
A	B	C	F_1	F_2
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

$F_1(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+\bar{C})$

$= M_1 . M_2 . M_4 . M_5 . M_7$

$= \pi(1, 4, 5, 7)$

$F_2 = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)$

$= M_0, M_1, M_2, M_4,$

$= \pi(0,1,2,4)$

Converting POS to standard POS:

1. Add $A.\bar{A}$ to each sum term, where A is missing term.
2. Apply rule $A+BC = (A+B)(A+C)$
3. $A.A = A$
4. Repeat step 1 to 3 until standard POS is obtained.

EXAMPLE:

i. Convert $(A+\bar{B}+C)(\bar{B}+C+D)$

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

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

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

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

Relationship between minterms and maxterms:

$m_i = \overline{M}_j$

E.g., $\bar{A}.\bar{B}.\bar{C} = \overline{A+B+C}$ [De – Morgan's law]

i.e., Each maxterm is the complement of its minterm & viceversa.

Conversion of standard SOP to POS

1. Evaluate each SOP terms and determine its binary representation.
2. Determine all the binary number not included in evaluation in step 1.
3. Write equivalent sum term for each binary number obtained in step 2.

Note: The similar process is applied for POS to SOP.

EXAMPLE:

i. Convert SOP: $\bar{A}\,\bar{B}\,C + \bar{A}\,BC + A\,\bar{B}\,C + ABC$ to POS expression.

⇒

Step 1: $\bar{A}\,\bar{B}\,C + \bar{A}\,BC + A\bar{B}\,C + ABC$

$= 001 \quad 011 \quad 101 \quad 111$

$1 \quad 3 \quad 5 \quad 7$

Step 2: Remaining binary numbers

$= 000 \quad 010 \quad 100 \quad 110$

$\mathbf{0 \quad 2 \quad 4 \quad 6}$

Step 3: POS = π (0, 2, 4, 6)

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

ii. Convert POS: $(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C)$ to SOP expression.

⇒ Step 1: $(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C)$

$= \quad 001 \quad 100 \quad 110$

$1 \quad 4 \quad 6$

Step 2: Remaining binary numbers

$= \quad 000 \quad 010 \quad 011 \quad 101 \quad 111$

$0 \quad 2 \quad 3 \quad 5 \quad 7$

Step 3: SOP = Σm (0, 2, 3, 5, 7)

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

3.3 Karnaugh Map (K-map)

K-map is a graphical chart which contains boxes and is a pictorial form of truth table. It is used to simplify the Boolean equations. The number of squares contained in the K–map depends upon the number of logical variables i.e., for n variables, there are 2^n squares arranged in specific order. It comprises a box for every line in truth table.

For

2 variables, there are $2^2 = 4$ cells in K-map

3 variables, there are $2^3 = 8$ cells in K-map

4 variables, there are $2^4 = 16$ cells in K-map

n variables, there are $2^n = 2^n$ cells in K-map.

Truth table to K–map:

A \ B	$\bar{B}$	B
$\bar{A}$	$\bar{A}\,\bar{B}$	$\bar{A}\,B$
A	$A\,\bar{B}$	AB

OR

A \ B	0	1
0	m_0	m_1
1	m_2	m_3

For two variables

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

A \ BC	0	1
00	m_0	m_1
01	m_2	m_3
11	m_6	m_7
10	m_4	m_5

For three variables

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

For four variables

Relation between a truth table and K–map:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

A \ B	0	1
0	0	0
1	0	1

Truth Table of AND gate

Simplification procedure:

1. Plotting
2. Grouping
3. Simplification

1. **Ploting**
   - We have to plot 1, if the given expression is SOP
   - We have to plot 0, if the given expression is POS.
2. **Grouping techniques**

We should group adjacent cells only; number of 1s or 0s should be in 2^n numbers.

i.e., $2^0 = 1$ (isolated term)

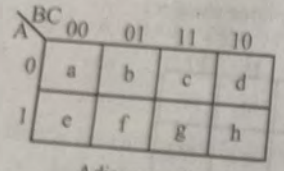
$2^1 = 2$ (pair)

$2^2 = 4$ (quad)

$2^3 = 8$ (octet)

$2^4 = 16$ (hex)

A pair eliminates one variable and its complements, a quad eliminates two variables and their complements, an octet eliminates three variables and their complements, and a hex eliminates four variables and their complements. So, for greatest simplification results, we should encircle the hex first, octets second, the quads third, and the pairs last.

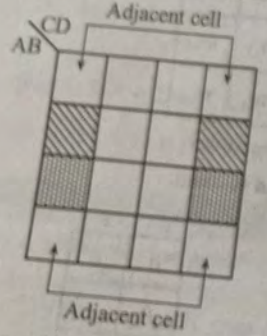


a and b are adjacent cells

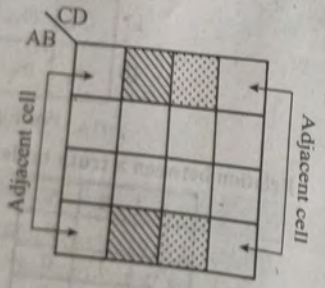
a and e are adjacent cells

a and f are not adjacent cells

a and d are adjacent cells



Leftmost and rightmost cells are adjacent cells.



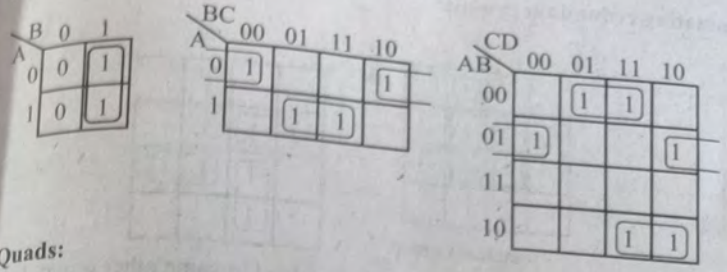
Top and corresponding bottom cells are adjacent cells.

Adjacent cell

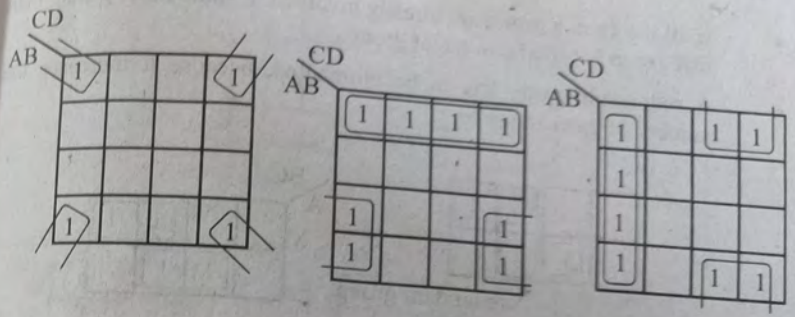
Adjacent cell

Fig.: Illustration of adjacent cells

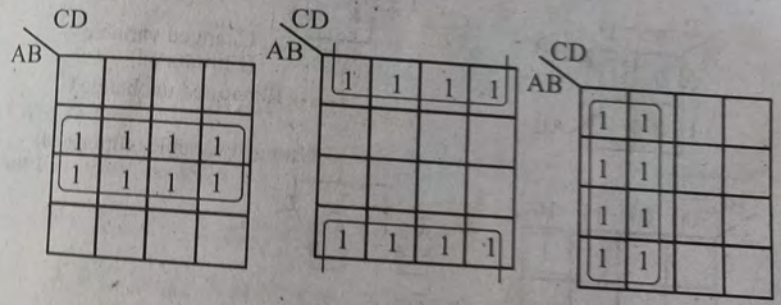
Pairs:



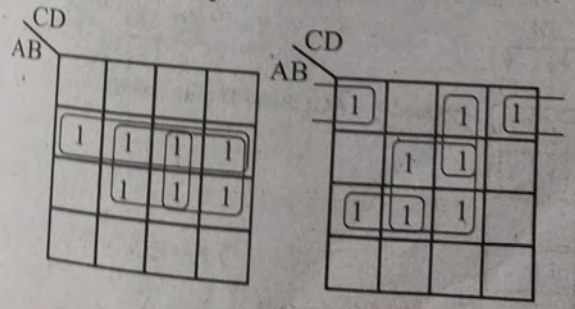
Quads:



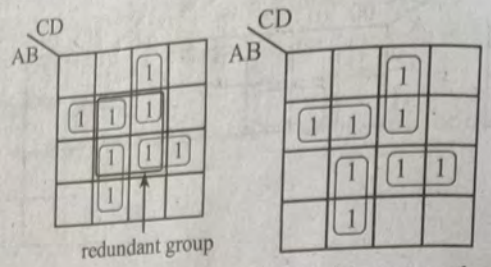
Octets:



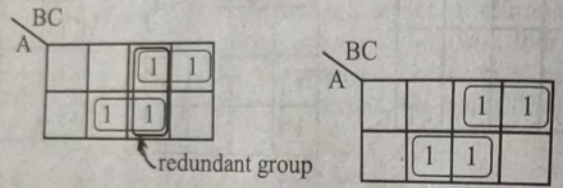
Overlapping groups:



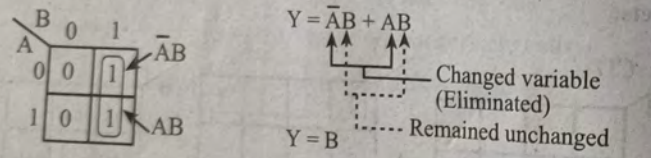
Eliminating redundant groups:

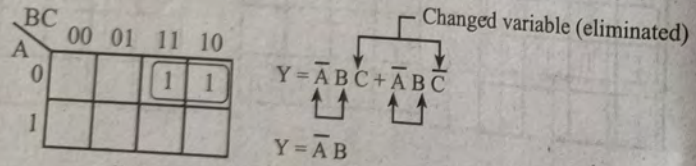


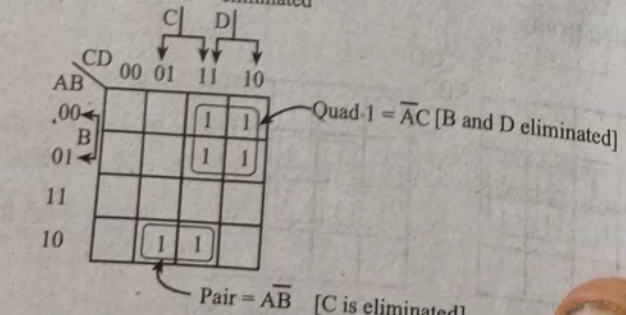
- If all the 1s in a group are already involved in some other group, the that group is called redundant group.
- A redundant group has to be eliminated, because it increases the number of gates required.



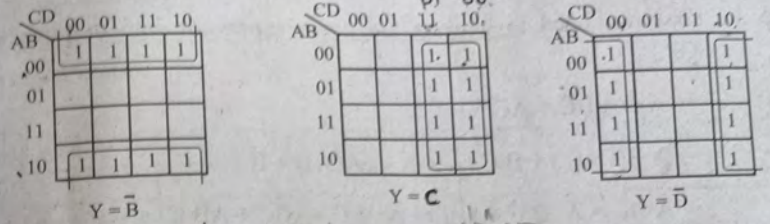
Simplification (SOP)

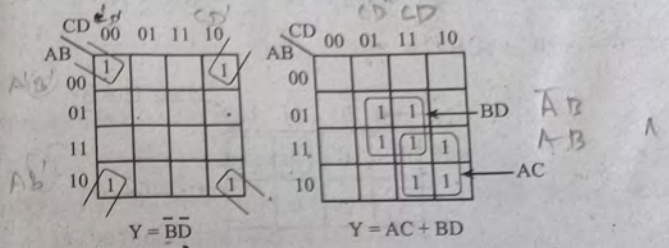


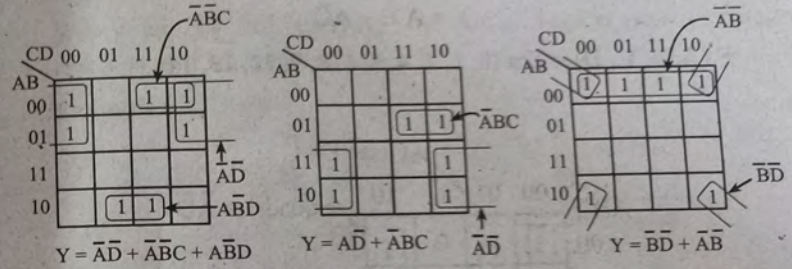




Some more examples:





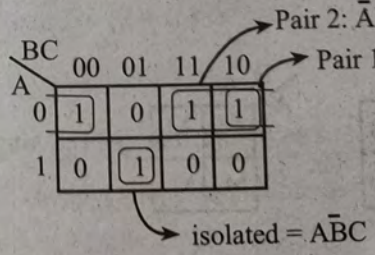


EXAMPLE:

Simplify using K–map:

1. $F = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C$

$\Rightarrow$ $\bar{A}\bar{B}\bar{C} = m_0$, $\bar{A}B\bar{C} = m_2$, $\bar{A}BC = m_3$, $A\bar{B}C = m_5$



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

2. $F = \bar{A}B + B\bar{C} + A\bar{C}$

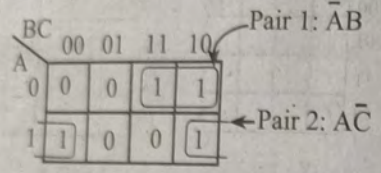
⇒ First, we need to convert the given expression into the standa SOP.

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

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

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

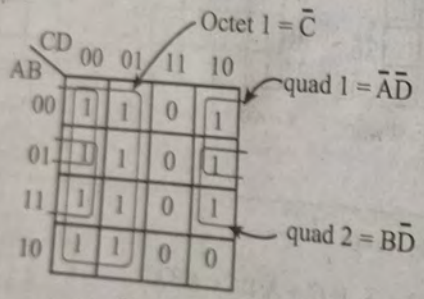
$= \bar{A}BC + \bar{A}B\bar{C} + AB\bar{C} + A\bar{B}\bar{C}$



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

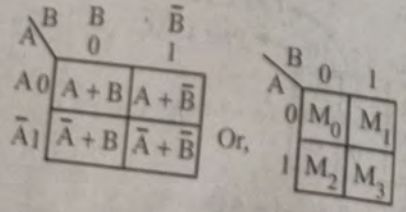
3. $F(A, B, C, D) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

⇒

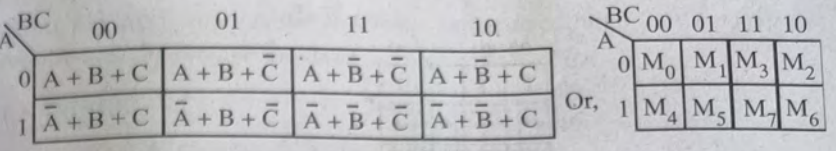


$Y = \bar{C} + \bar{A}\bar{D} + B\bar{D}$

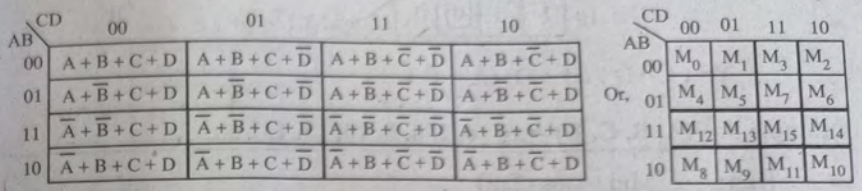
Simplification (POS):



For 2 variables



For 3 variables

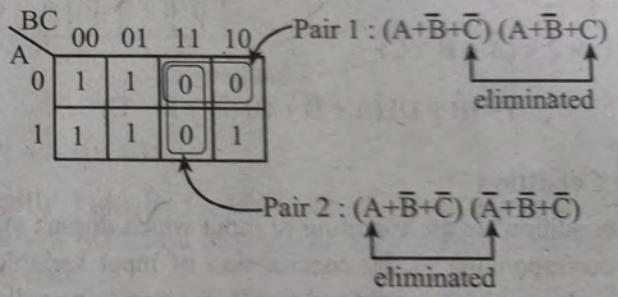


For 4 variables

Simplification procedure (POS):

1. The given POS expression consists of maxterms.
2. Corresponding to every maxterm, we enter a 0s in K-map and we enter 1s in the remaining cells of k-map.
3. We group 0s for carrying out simplification using grouping techniques of adjacent cells.

Example:



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

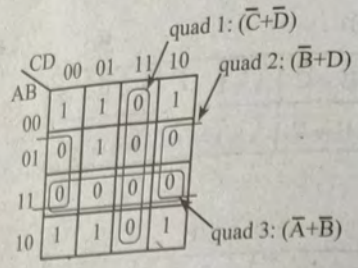
pair 2 = $(\bar{B} + \bar{C})$

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

Example 2: F(A, B, C,D) = ∑m(0, 1, 2, 5, 8, 9, 10)

⇒ The given terms are minterms , so the remaining are maxterms .

i.e., $F(A, B, C, D) = \pi_M(3, 4, 6, 7, 11, 12, 13, 14, 15)$

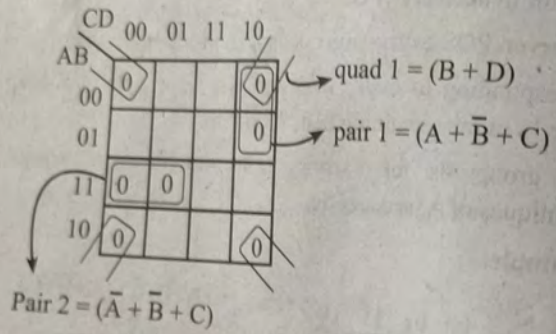


$\therefore\ Y = (\bar{C}+\bar{D})\,(\bar{B}+D)\,(\bar{A}+\bar{B})$

Example 3: F(A, B, C, D) = π_M (0, 2, 6, 8, 10, 12, 13)

⇒ Hint: Min terms (Σm)

Max terms (π_M)



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

Don't Care Condition

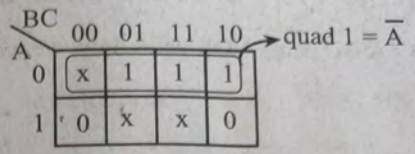
"Don't care condition" is the condition of input which doesn't affect output. We plot 1s corresponding to the combination of input variables for SOP form which produces a high output and plot 0s for remaining cells of K-map. We plot 0s corresponding to the combination of input variables for POS form which produces a low output and plot 1s for remaining cells of K-map. However, it is not always true that the cell not containing 1s (in SOP form) will contain 0s because some conditions of input variable do not occur. Also, for some function, the output corresponding to certain combinations of input variables do not occur. In such condition, we have a freedom to assume a 0 or 1 as output for each of these combinations. These conditions are known as the "don't care conditions". It is represented as 'x' (cross) mark in corresponding cells.

Note: The don't care condition (X) may be assumed to be 0 or 1 as per the requirement for simplification.

EXAMPLE:

1. **F (A, B, C)= Σm (1, 2, 3) + D(0, 5, 7)**

⇒

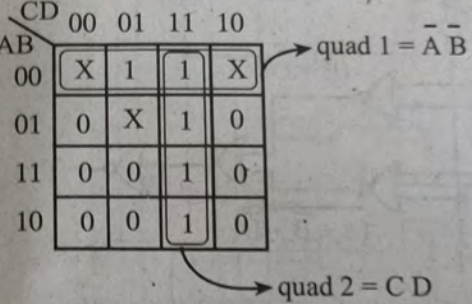


$\therefore\ Y = \bar{A}$

2. **F (A, B, C, D) = Σm (1, 3, 7, 11, 15)**

D(A, B, C, D)= d(0, 2, 5)

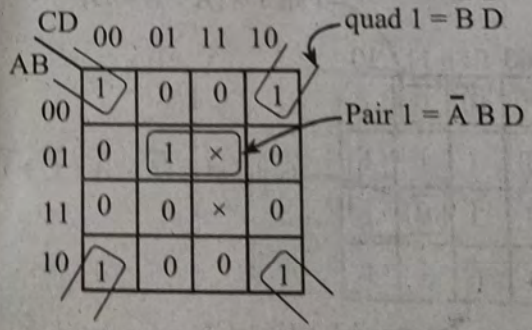
⇒



$\therefore\ Y = \bar{A}\,\bar{B} + CD$

3. **Simplify F(A, B, C, D)=Σm(0, 2, 5, 8, 10)+d(7, 15). Write its standard SOP and implement the simplified circuit using NOR gates only.**

⇒



$Y = \bar{B}\,\bar{D} + \bar{A}\,BD$

For standard SOP,

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

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

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

For implementing $Y = \bar{B}\,\bar{D} + \bar{A}\,BD$ using NOR gates only, we write

$Y = \bar{B}\,\bar{D} + \bar{A}\,BD$

$= \overline{\overline{\bar{B}\,\bar{D}}} + \overline{\overline{\bar{A}\,BD}} = \overline{B + D} + \overline{A + \bar{B} + \bar{D}}$

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

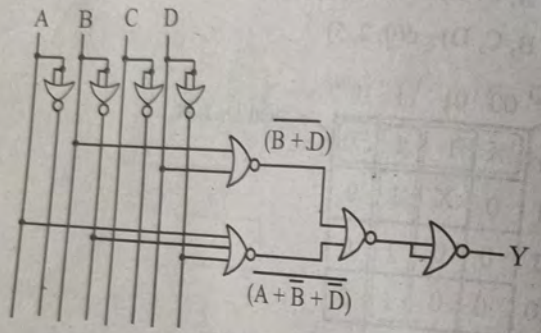
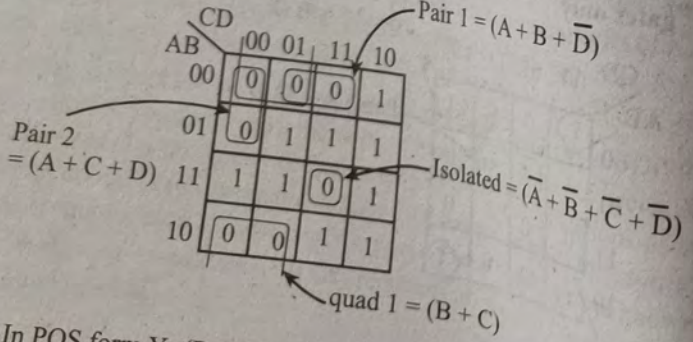


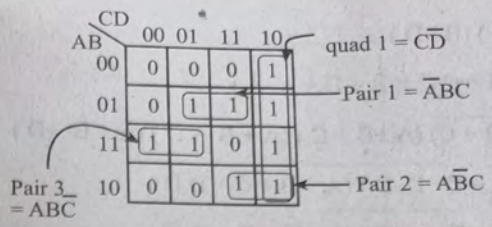
Fig.: Implementing using NOR gates only

4. **Simplify the Boolean function in both SOP and POS, and implement using basic gates only.**
$F(A, B, C, D) = \pi M(0, 1, 3, 4, 8, 9, 15)$

⇒



In POS form, $Y = (B + C)\,(A + B + \bar{D})\,(A + C + D)\,(\bar{A} + \bar{B} + \bar{C} + \bar{D})$



In SOP form, $Y = C\bar{D} + \bar{A}\,BC + A\bar{B}\,C + AB\,\bar{C}$

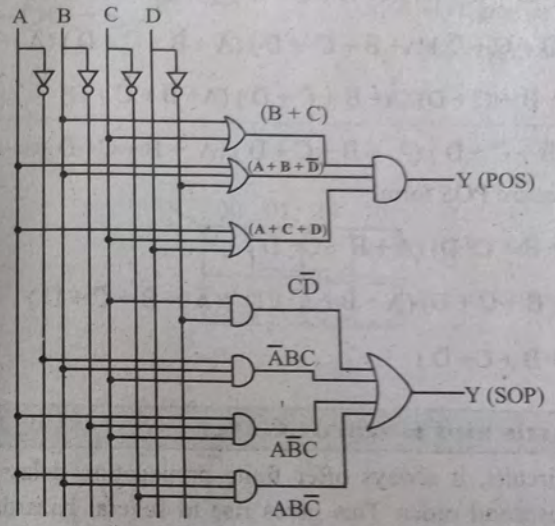
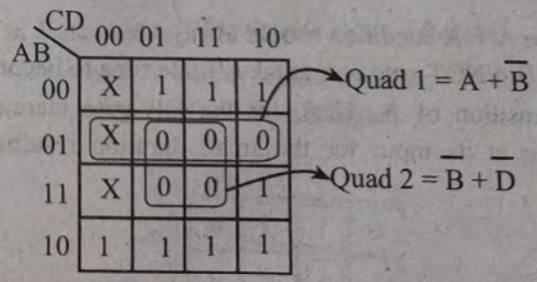


Fig.: Circuit diagram of both SOP and POS expression

5. **Simplify $\Sigma m(1, 2, 3, 8, 9, 10, 11, 14)$ and d (0, 4, 12) by using K-map and write its standard product of sum (POS) expression.**

⇒



In POS form, $Y = (A + \bar{B})(\bar{B} + \bar{D})$

For standard POS,

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

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

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

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

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

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

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

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

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

In standard POS form,

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

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

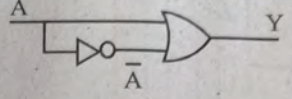
$$(\bar{A}+\bar{B}+C+\bar{D})$$

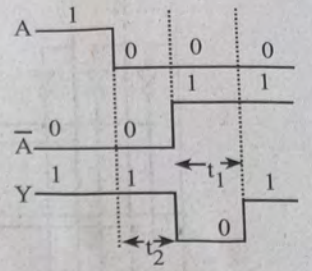
3.4 Hazards and Hazards Cover

In practical circuits, it always offer finite propagation delay though very small, in nanosecond order. This gives rise to several hazards and hazard recover are additional terms in an equation that prevents occurring of them.

1) Static-1 Hazard

This type of hazard occurs when $Y = A + \bar{A}$ type of situation appears for a logic circuit for certain other inputs and A makes transition from $1 \rightarrow 0$. An $A + \bar{A}$ condition should always generate 1 at the output i.e., static 1. But NOT gate output takes finite time to become 1 following $1 \rightarrow 0$ transition of A. Thus, for the OR gate there are two zeros appearing at its input for the small duration resulting a 0 at this output.





t_1 = NOT gate delay

t_2 = OR gate delay

EXAMPLE:

The K-map is given

A \ BC	00	01	11	10
0	0	1	1	0
1	0	0	1	1

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

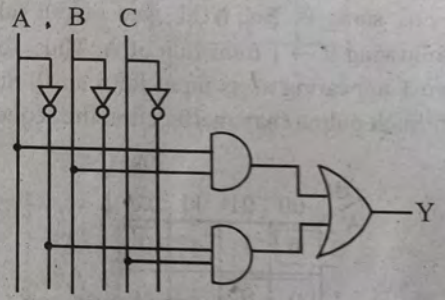


Fig.: Circuit with static-1 hazard

When B = 1, C= 1, and A makes transition from $1 \rightarrow 0$, static–1 hazard occurs. To remove it, additional BC term is added.

A \ BC	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y = \bar{A}C + AB + BC$$

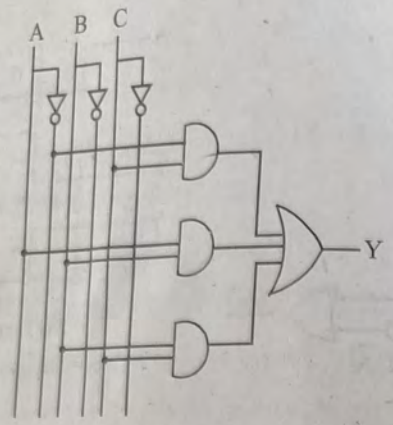
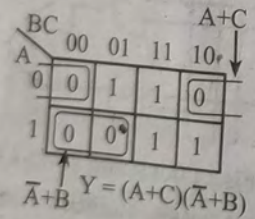


Fig.: Hazard free circuit

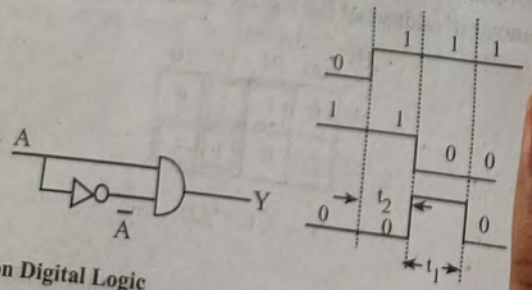
When, B = 1, C = 1, and A makes transition from 1 to 0, there is no effect upon output.

2) **Static–0 Hazard**

This type of hazard occurs when Y = A. $\bar{A}$ kind of situation occurs in a logic circuit for certain combination of other inputs and A makes transition from 0 → 1. A. $\bar{A}$ condition should always generate 0 at the output i.e., static 0. But NOT gate output takes finite time to become 0 following 0 → 1 transition of A. Thus, for final AND gate, there are two 1 appearing at its input for a small duration resulting 1 at its output. Such output may malfunction the sequential circuit.



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



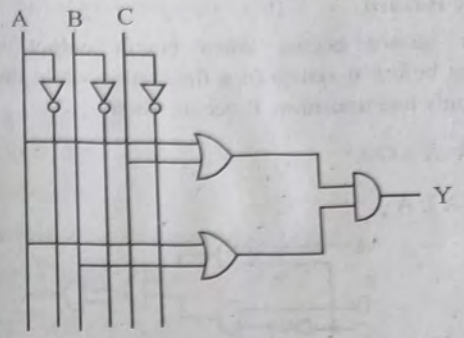
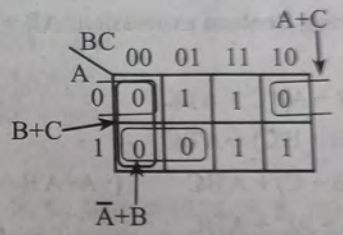


Fig.: Circuit with static–0 hazard

When B = 0, C = 0 and A makes transition from 0 →1, static–0 hazard will occur at output. To prevent this, we add additional term (B + C).



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

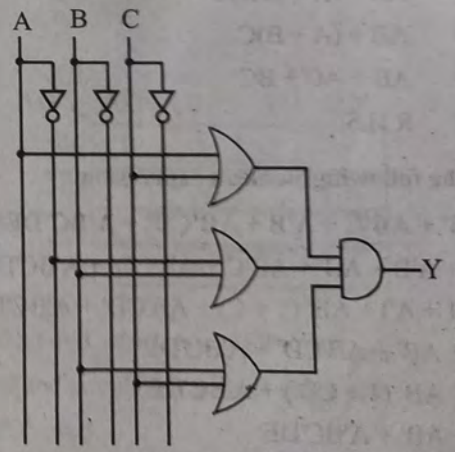


Fig.: Hazard free circuit

When B = 0, C = 0 and C tends 0 → 1, there is no change in output Y = 0

3) Dynamic Hazard

Dynamic hazard occurs when circuit output makes multiple transitions before it settles to a final value while the logic equation asks for only one transition. It occurs when

$Y = A + \bar{A}.A$ **OR**

$Y = (A + \bar{A}).A$

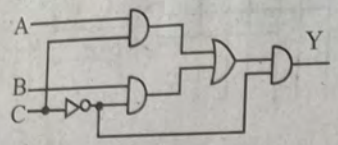


Fig.: For illustrating dynamic hazard

MORE WORKED OUT EXAMPLES:

1. Prove the following Boolean expression: AB + AB'C + A'BC = AB + AC + BC. **[2064 Jestha]**

⇒ L.H.S. = AB + AB'C + A'BC

= A(B + B'C) + A'BC

= A(B + C) + A'BC [∵A+A'B = A+B]

= AB + AC + A'BC

= AB + (A + A'B)C

= AB + (A + B)C

= AB + AC + BC

= R.H.S.

2. Simplify the following Boolean expression **[2064 Falgun]**

⇒ **a. B + AB' + AB'C + A'B + AB'CD' + A'BC'DE'**

= B + A'B + AB' + AB'C + AB'CD' + A'BC'DE'

= B(1 + A') + AB' (1 + C) + AB'CD' + A'BC'DE'

= B + AB' + AB'CD' + A'BC'DE'

= B + AB' (1 + CD') + A'BC'DE'

= B + AB' + A'BC'DE'

= B(1 + A'C'DE') + AB'

= B + AB'

= A + B

b. [{(A + B) ' + A}'+ {(A + B) ' + B}']'

= {(A + B) ' + A}". {(A + B) ' + B}"

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

= (A + B)' + AB

= A'B' + AB

= A ⊙ B

3. Prove the following Boolean expression:

$AB + A\bar{B}C + \bar{A}BC = AB + AC + BC$ **[2067 Ashadh]**

⇒ LHS = $AB + A\bar{B}C + \bar{A}BC$

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

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

= $AB + AC + \bar{A}BC$

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

= AB + C(A + B)

= AB + AC + BC

= RHS.

4. Simplify F(A, B, C, D) = π (0, 2, 5, 8, 10) + d (7, 15). Write its standard SOP and implement the simplified circuit using NOR gates only. **[2069 Chaitra]**

⇒

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	0	×	1
11	1	1	×	1
10	0	1	1	0

F = BD' + B'D + AB which is in SOP form

To implement the circuit using NOR gates only, we write

F = BD' + B'D + AB

= (BD')" + (B'D)" + (AB)"

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

= [(B' + D)' + (B + D')' + (A' + B')']"

3) **Dynamic Hazard**

Dynamic hazard occurs when circuit output makes multiple transitions before it settles to a final value while the logic equation asks for only one transition. It occurs when

$Y = A + \bar{A}.A$ **OR**

$Y = (A + \bar{A}).A$

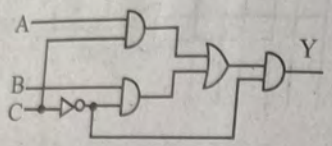


Fig.: For illustrating dynamic hazard

MORE WORKED OUT EXAMPLES:

1. **Prove the following Boolean expression: AB + AB'C + A'BC = AB + AC + BC.** **[2064 Jestha]**

⇒ L.H.S. = AB + AB'C + A'BC

= A(B + B'C) + A'BC

= A(B + C) + A'BC [∵A+A'B = A+B]

= AB + AC + A'BC

= AB + (A + A'B)C

= AB + (A + B)C

= AB + AC + BC

= R.H.S.

2. **Simplify the following Boolean expression**

⇒ **a. B + AB' + AB'C + A'B + AB'CD' + A'BC'DE'** **[2064 Falgun]**

= B + A'B + AB' + AB'C + AB'CD' + A'BC'DE'

= B(1 + A') + AB' (1 + C) + AB'CD' + A'BC'DE'

= B + AB' + AB'CD' + A'BC'DE'

= B + AB' (1 + CD') + A'BC'DE'

= B + AB' + A'BC'DE'

= B(1 + A'C'DE') + AB'

= B + AB'

= A + B

b. [{(A + B) ' + A}'+ {(A + B) ' + B}']'

= {(A + B) ' + A}". {(A + B) ' + B}"

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

= (A + B)' + AB

= A'B' + AB

= A ⊙ B

3. **Prove the following Boolean expression:**

$AB + A\bar{B}C + \bar{A}BC = AB + AC + BC$ **[2067 Ashadh]**

⇒ LHS = $AB + A\bar{B}C + \bar{A}BC$

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

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

= $AB + AC + \bar{A}BC$

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

= AB + C(A + B)

= AB + AC + BC

= RHS.

4. **Simplify F(A, B, C, D) = π (0, 2, 5, 8, 10) + d (7, 15). Write its standard SOP and implement the simplified circuit using NOR gates only.** **[2069 Chaitra]**

⇒

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	0	×	1
11	1	1	×	1
10	0	1	1	0

F = BD' + B'D + AB which is in SOP form

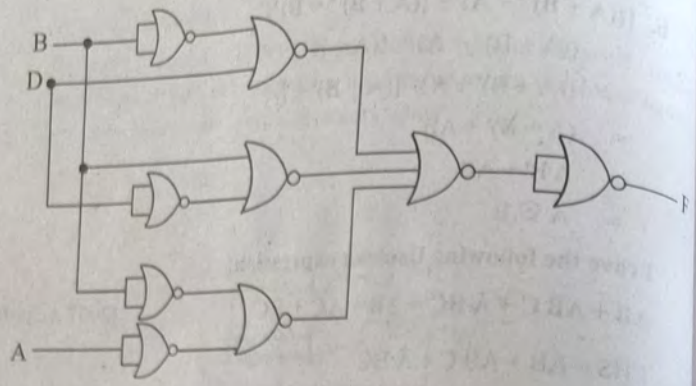
To implement the circuit using NOR gates only, we write

F = BD' + B'D + AB

= (BD')" + (B'D)" + (AB)"

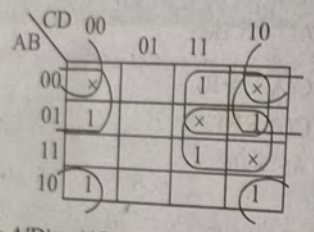
= (B' + D)' + (B + D')' + (A' + B')'

= [(B' + D)' + (B + D')' + (A' + B')']"



5. Simplify the following using K–map: $F = \Sigma_m (3, 4, 6, 8, 10, 15) + (0, 2, 7, 14)$. **[2069 Ashadh**

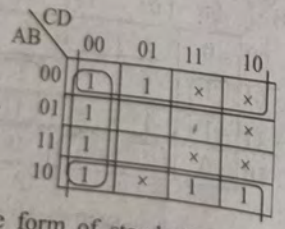
⇒



F(A, B, C, D) = A'D' + A'C + BC + B'D' in SOP form.

6. **Simplify the function using K–map: F = Σ (0, 1, 4, 8, 10, 11, 12) and D = Σ(2, 3, 6, 9, 15). Also convert the result into standard minterm.** **[2068 Chaitra]**

⇒

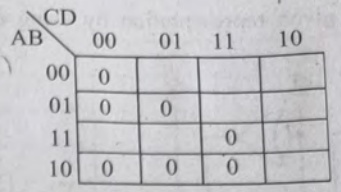


Expressing in the form of standard minterms i.e., sum of product expression, we have

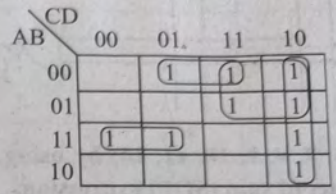
F(A, B, C, D) = C'D' + B'

7. **Simplify π(0, 4, 5, 8, 9, 11, 15) using K–map and write its standard SOP expression.** **[2068 Baishakh]**

⇒ F(A, B, C, D) = π(0, 4, 5, 8, 9, 11, 15)



For representing the expression in standard SOP form,



SOP expression is

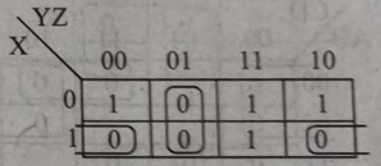
F(A, B, C, D) = ABC' + A'B'D + A'C + CD'

8. **Obtain the product of sum expression for the function F = Σ (0, 2, 3, 7) and draw a circuit using any universal gate of your choice to realize this function.** **[2066 Bhadra]**

⇒ Let's choose input terminals: A, B and C

Output terminal: Y

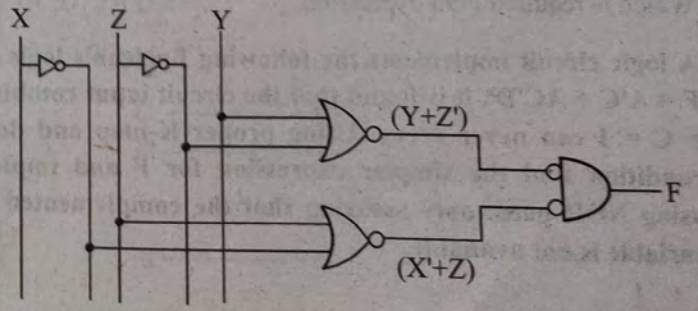
Given function: F = Σ (0, 2, 3, 7)



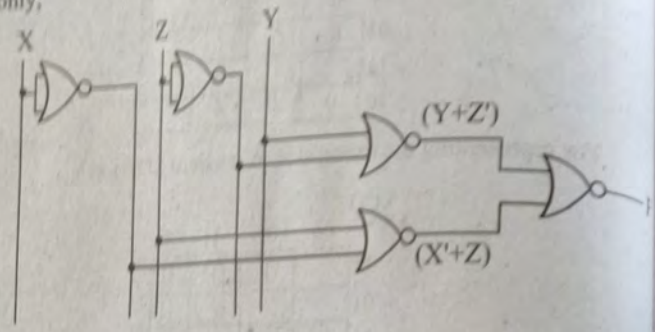
Expressing the given function in POS form, we have

F = (Y + Z').(X' + Z) [∵Using De-Morgan's theorem]

Now, graphical realization



Realizing the given representation by using universal (NOR) gate only,



9. **Simplify Σ(1, 5, 7, 8, 9, 10, 11, 14) by using K–map and write its standard product of sum (SOP) expression.** **[2064 Jestha]**

⇒

AB \ CD	00	01	11	10
00		1		
01		1	1	
11				1
10	1	1	1	1

For POS expression,

AB \ CD	00	01	11	10
00	0		0	0
01	0			0
11	0	0	0	
10				

$Y = (A + D).(A + B + C').(A' + B' + C) . (A' + B' + D')$

Which is required POS expression.

10. **A logic circuit implements the following Boolean's logic function $F = A'C + AC'D'$, it is found that the circuit input combination A = C = 1 can never occur. Using proper K-map and don't care condition find the simpler expression for F and implement it using NOR gates only assuring that the complemented from of variable is not available.**

⇒ $F = A'C + AC'D'$

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	×
1	0	1	1	×
1	1	0	0	1
1	1	0	1	0
1	1	1	0	×
1	1	1	1	×

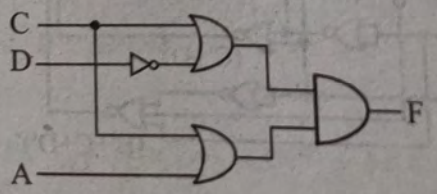
AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	×	×
10	1	0	×	×

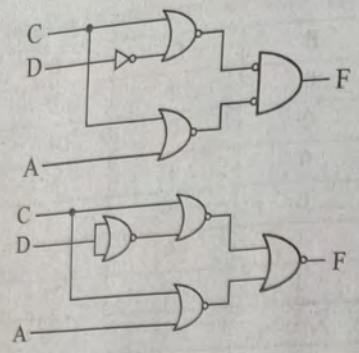
$F' = C'D + A'C'$

$F = (C'D + A'C')'$

$= (C'D)'.(A'C')'$

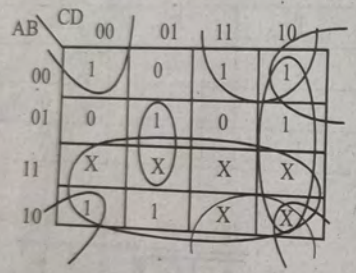
$= (C + D')(A + C)$





11. Simplify the following function using K-map and implement the result using only NOR gates: $F(A, B, C, D) = \Sigma_m (0, 2, 3, 5, 6, 8, 9) + d (10, 11, 12, 13, 14, 15)$. [2071 Chaitra]

⇒ $F(A, B, C, D) = \Sigma_m (0, 2, 3, 5, 6, 8, 9) + d (10, 11, 12, 13, 14, 15)$

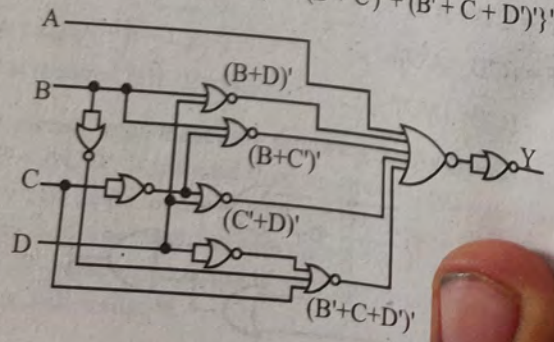


$F = A + CD' + B'D' + B'C + BC'D$

$= A + (CD')'' + (B'D')'' + (B'C)'' + (BC'D)''$

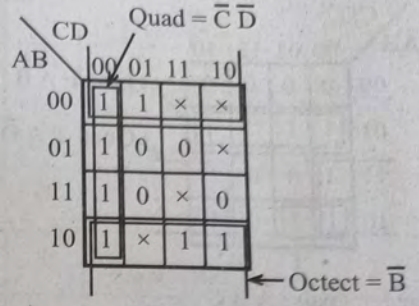
$= A + (C' + D)' + (B + D)' + (B + C)' + (B' + C + D')'$

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



12. Simplify the function using K-map $F = \Sigma(0, 1, 4, 8, 10, 11, 12)$ and $D = \Sigma(2, 3, 6, 9, 15)$. Also realize the simplified circuit using NOR gates. [2075 Ashwin]

⇒

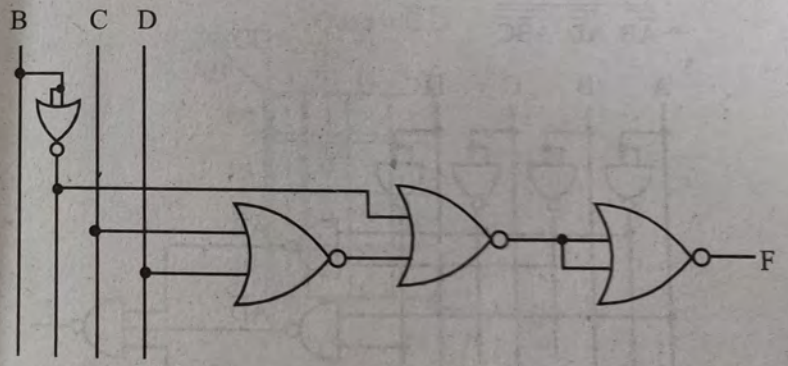


$F = \bar{B} + \bar{C}\,\bar{D}$

Implementing above function using NOR gates:

$F = \bar{B} + \bar{C}\,\bar{D}$

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



13. Minimize the expressions and implement the reduced expression by using NAND gates

$F = \bar{A}\,B\bar{C}\,\bar{D} + \bar{A}\,B\bar{C}\,D + \bar{A}\,BCD + \bar{A}\,BC\bar{D} + A\bar{B}\,\bar{C}\,\bar{D} + A\bar{B}\,\bar{C}\,D + A\bar{B}\,C\bar{D} + AB\bar{C}\,\bar{D} + ABC\bar{D}$ [2074 Chaitra]

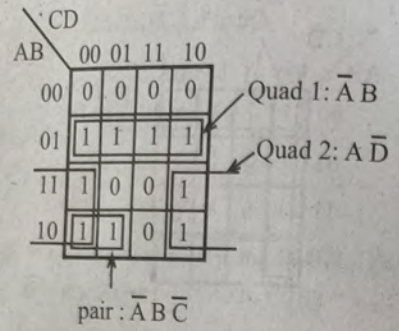
⇒

$= \bar{A}\,B\bar{C}\,\bar{D} + \bar{A}\,B\bar{C}\,D + \bar{A}\,BCD + \bar{A}\,BC\bar{D} + A\bar{B}\,\bar{C}\,\bar{D} + A\bar{B}\,\bar{C}\,D + A\bar{B}\,C\bar{D} + AB\bar{C}\,\bar{D} + ABC\bar{D}$

$= 0100 + 0101 + 0111 + 0110 + 1000 + 1001 + 1010 + 1100 + 1110$

$= m_4 + m_5 + m_7 + m_6 + m_8 + m_9 + m_{10} + m_{12} + m_{14}$

$= \sum m(4, 5, 6, 7, 8, 9, 10, 12, 14)$



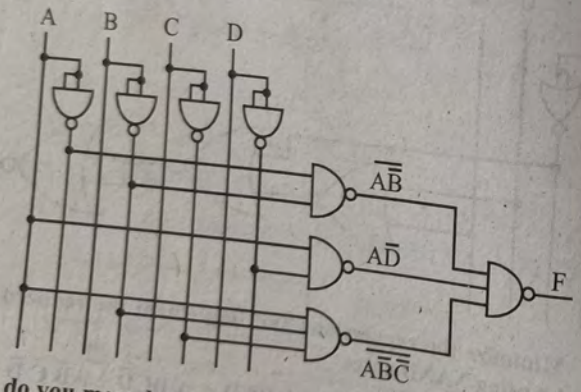
$F = \bar{A}B + A\bar{D} + A\bar{B}\bar{C}$

Implementing using NAND gates:

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

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

$= \overline{\overline{\bar{A}B}}.\ \overline{\overline{A\bar{D}}}.\ \overline{\overline{A\bar{B}\bar{C}}}$



14. What do you mean by maxterm? Explain with example. **[2074 Chaitra]**

⇒ Standard POS is one in which all the variables in the domain appear in each sum term in the expressions.

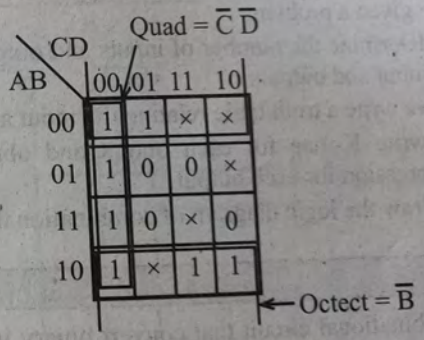
Each individual sum term in standard POS form is called maxterm and is represented by M.

Maxterm of three variables:

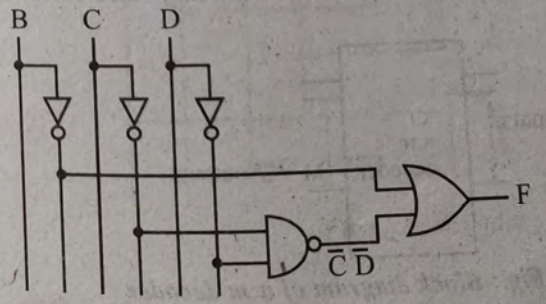
A	B	C	Term	Designation (M_j)
0	0	0	A+B+C	M_0
0	0	1	$A+B+\bar{C}$	M_1
0	1	0	$A+\bar{B}+C$	M_2
0	1	1	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$\bar{A}+B+C$	M_4
1	0	1	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$\bar{A}+\bar{B}+C$	M_6
1	1	1	$\bar{A}+\bar{B}+\bar{C}$	M_7

15. Simplify using K–map: $F = \sum(0, 1, 4, 8, 10, 11, 12)$ and $D = \sum(2, 3, 6, 9, 15)$. Also realize the simplified logic circuit.

⇒



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



Chapter – 4

DATA PROCESSING CIRCUITS

4.1 Combinational Circuit

A combinational circuit may be defined as a logic circuit the output of which depends only upon the combination of the present inputs. The output does not depend on the past value of inputs or outputs. Therefore, a combinational circuit does not need any memory. Combinational circuits are faster in operation than sequential circuits.

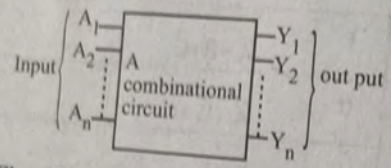


Fig.: Block diagram of a combinational circuit.

Examples of combinational circuit are half adder, full adder, half subtractor, full subtractor, encoder, decoder, multiplexer, demultiplexer, etc.

Designing procedure of a combinational circuit:

1. We will be given a problem.
2. Then, we determine the number of inputs and output and assign letter symbol to input and outputs.
3. After that we write a truth table relating the input and output.
4. Then, we write K-map for each output and obtain the simplified Boolean expression for each output.
5. Lastly, we draw the logic diagram of combinational circuit.

4.2 Decoder

A decoder is a combinational circuit that convert binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

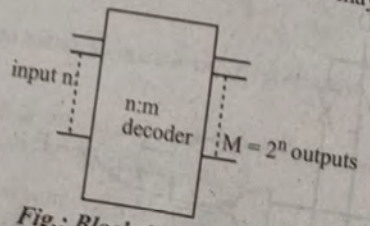


Fig.: Block diagram of n:m decoder

Application:

It is used in many types of application. For example, in computer, it is used to select input/output ports. The binary port address is decoded and the appropriate peripherals is selected for communication. Each I/O port has a unique address.

2:4 Decoder

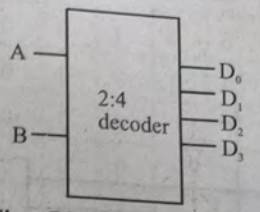


Fig.: Block diagram of a 2:4 decoder

Truth table

Input		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

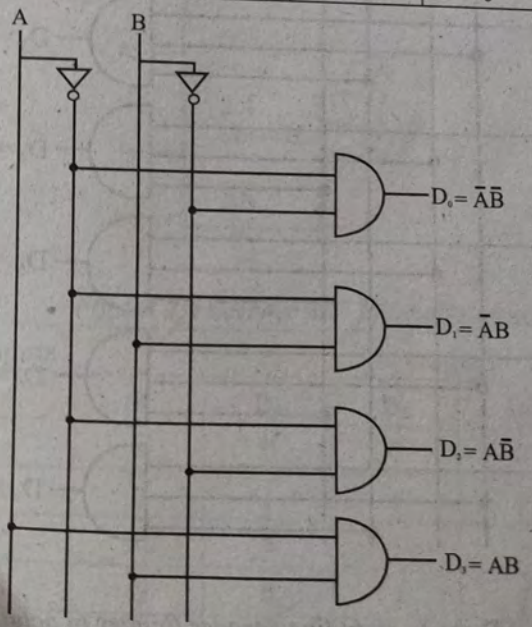


Fig.: 2: 4 decoder

3×8 Decoder (Binary to Octal Converter)

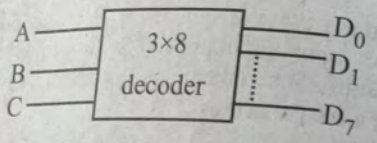


Fig.: Block diagram of a 3×8 decoder

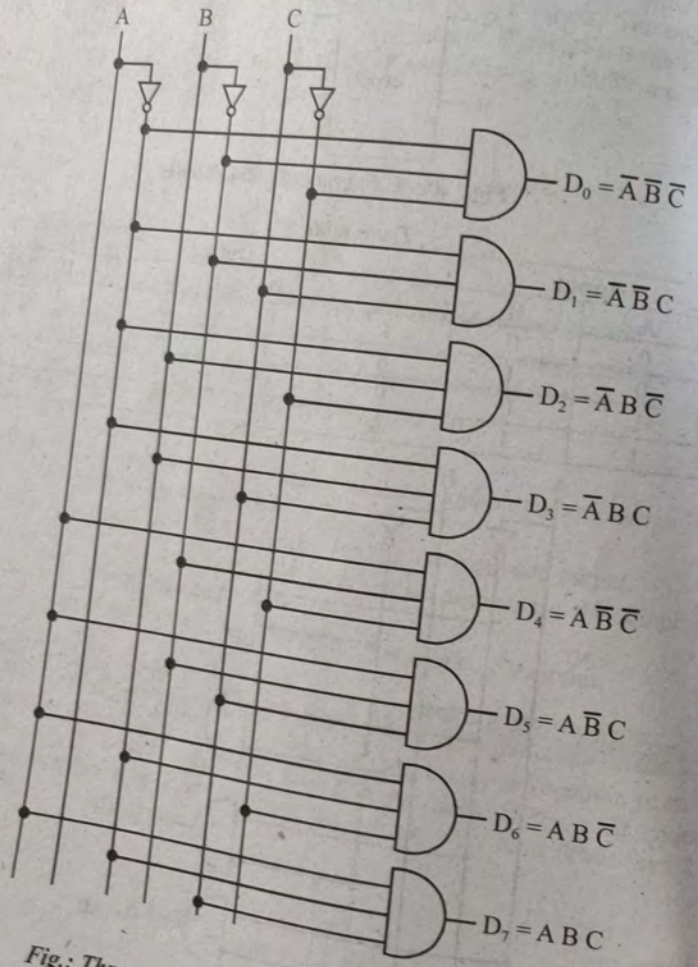


Fig.: Three-to-eight line decoder (binary to octal conve

Truth table

Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder with Enable

In different application, an additional signal input called a strobe is added so that the gates are enabled and decoding take place. It is sometime desired to decode only during certain interval of time. In such cases, decoder with enable is used.

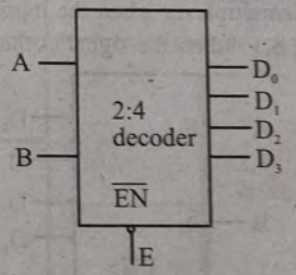


Fig.: A 2:4 decoder with an enable input

Inputs			Outputs			
E	A	B	D_0	D_1	D_3	D_3
1	×	×	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

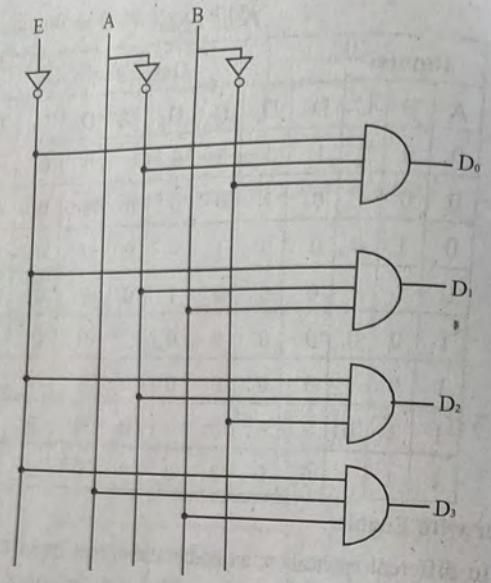


Fig.: Logic diagram of a 2:4 decoder with an enable input.

Decoder can work as a demultiplexer when the input (A, B) of decoder are made selection inputs (S_0, S_1) and enable signal as the data input.

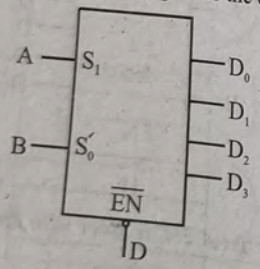


Fig.: Decoder working as a demultiplexer

Construction of a 3 ×8 decoder using 2× 4 decoder:

For 3×8 decoder, no. of inputs = 3 (E, A, B), no. of outputs = 8

For 2×4 decoder, no. of inputs = 2 (A, B), no. of outputs = 4

Truth table

E	A	B	D	
0	0	0	D_0	2 × 4 decoder 'I' when E = 0
0	0	1	D_1	
0	1	0	D_2	
0	1	1	D_3	
1	0	0	D_4	2 × 4 decoder 'II' when E = 1
1	0	1	D_5	
1	1	0	D_6	
1	1	1	D_7	

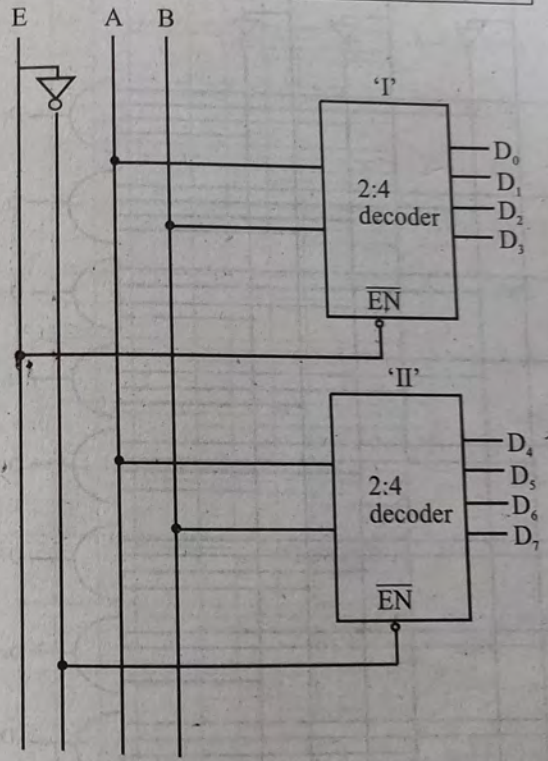


Fig.: 3×8 decoder using 2× 4 decoder

Design of BCD to Decimal Decoder

Inputs				Outputs									
A	B	C	D	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_7
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0

Inputs				Outputs									
A	B	C	D	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

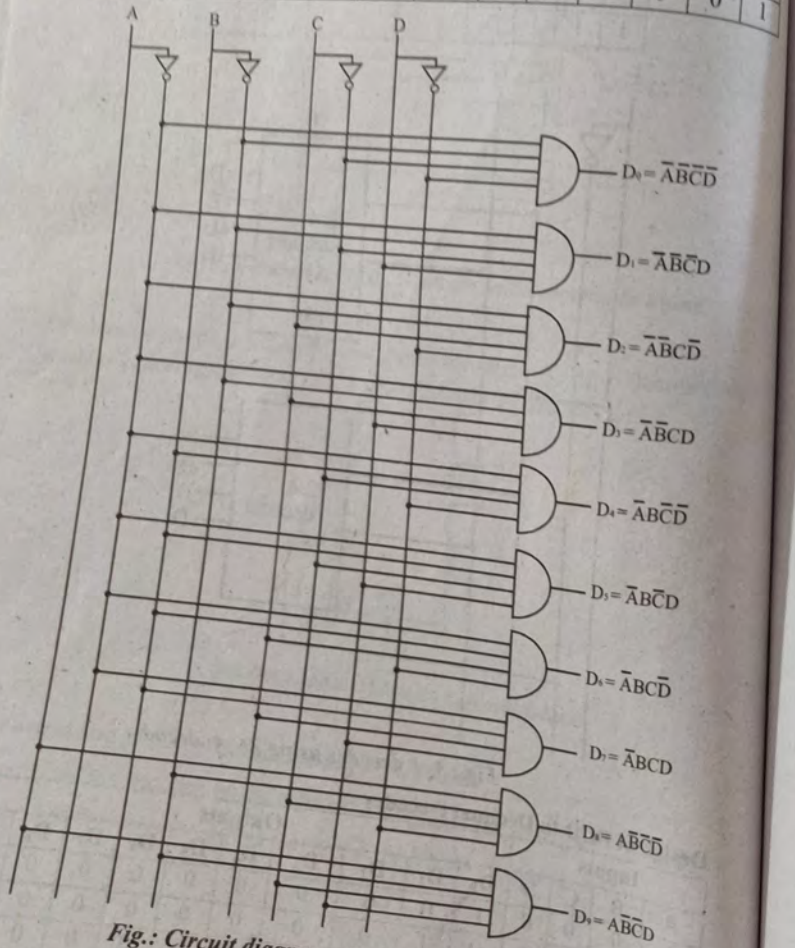


Fig.: Circuit diagram of a BCD to decimal decoder.

4:6 Line Decoder (Binary to Hexadecimal Converter)

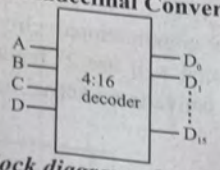


Fig.: Block diagram of a 4:6 decoder.

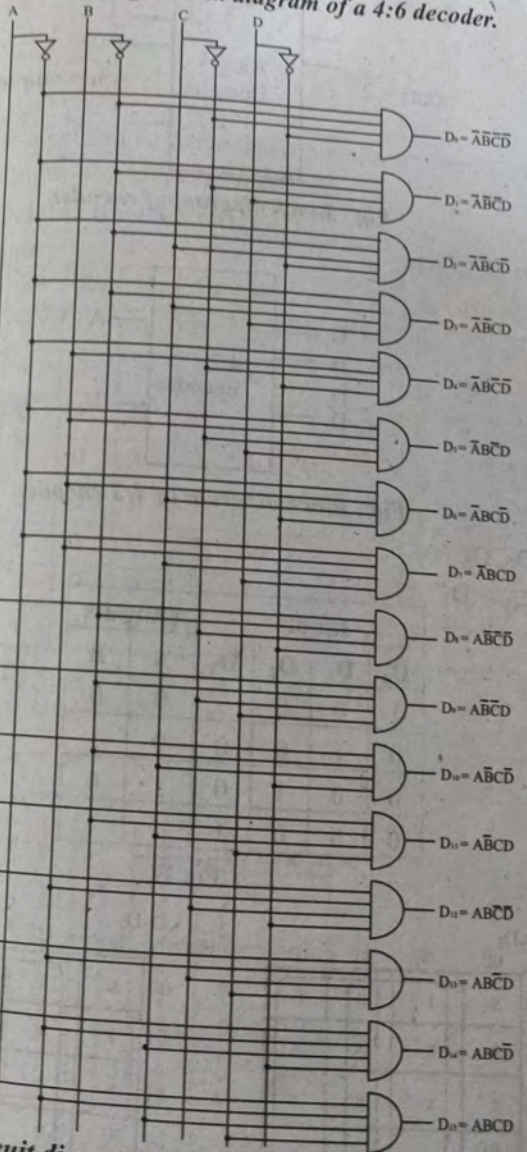


Fig.: Circuit diagram of 4×16 line decoder (binary to hexadecimal converter)

8. Encoder

An encoder is a combinational circuit that performs the inverse operation of a decoder. It has 2^n inputs only one of which is active and n outputs. It converts an active input signal into a coded output signal.

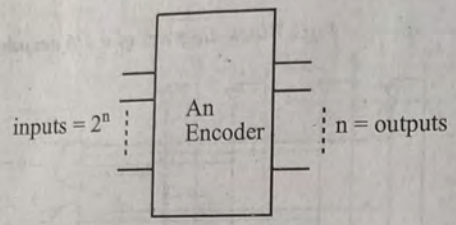


Fig.: Block diagram of encoder

4:2 Encoder:

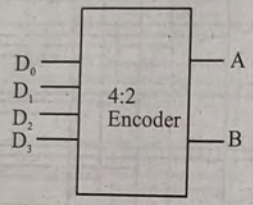


Fig.: Block diagram of 4:2 encoder

$A = D_2 + D_3$

$B = D_1 + D_3$

Inupts				Outputs	
D_0	D_1	D_2	D_3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

For A,

D_0D_1 \ D_2D_3	00	01	11	10
00	x	1	x	1
01	0	x	x	x
11	x	x	x	x
10	0	x	x	x

$A = D_2 + D_3$

For B,

D_0D_1 \ D_2D_3	00	01	11	10
00	x	1	x	0
01	1	x	x	x
11	x	x	x	x
10	0	x	x	x

$B = D_1 + D_3$

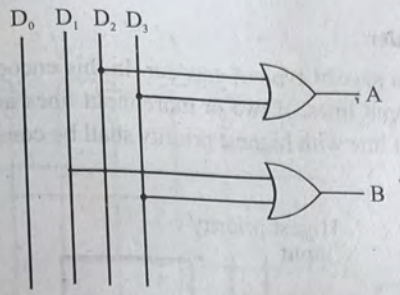


Fig.: 4×2 encoder

Limitations

Only one input can be enabled at a time. If two inputs are enabled at the same time, then output is undefined.

8:3 Encoder (Octal to Binary Converter)

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_3 + D_5 + D_7$

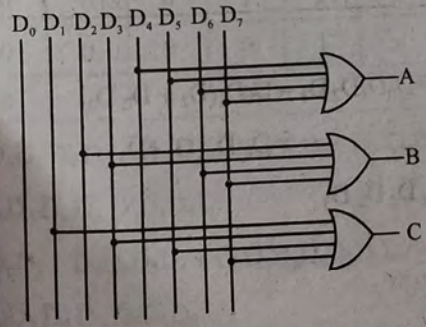


Fig.: Circuit diagram of an 8×3 encoder

Priority Encoder

This is a special type of encoder. In this encoder, priorities are given to the input lines. If two or more input lines are 1 at same time, then the input line with highest priority shall be considered.

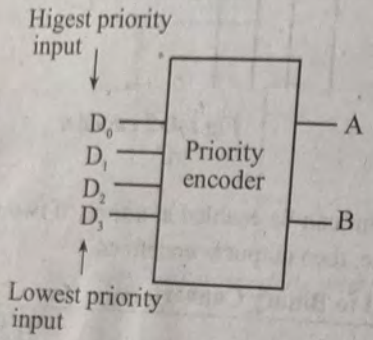


Fig.: Block diagram of a priority encoder.

4×2 Priority Encoder

Let D_0 has highest priority.

Truth table

Inputs				Outputs	
D_0	D_1	D_2	D_3	A	B
0	0	0	0	×	×
1	×	×	×	0	0
0	1	×	×	0	1
0	0	1	×	1	0
0	0	0	1	1	1

$$A = \overline{D}_0\,\overline{D}_1\,D_2 + \overline{D}_0\overline{D}_1\overline{D}_2\,D_3 = \overline{D}_0\overline{D}_1(D_2 + \overline{D}_2\,D_3)$$

$$= \overline{D}_0\,\overline{D}_1\,(D_2 + \overline{D}_3)$$

$$B = \overline{D}_0\,D_1 + \overline{D}_0\,\overline{D}_1\,\overline{D}_2\,D_3$$

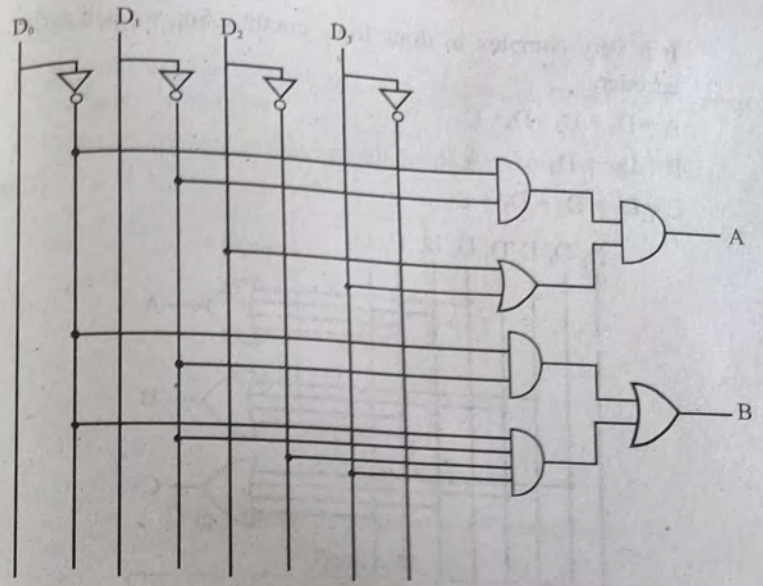


Fig.: Circuit diagram of a 4:2 priority enocoder

Construction of 8:3 Priority Encoder

Let D_7 has highest priority and D_0 has lowest priority

Truth table

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A	B	C
1	×	×	×	×	×	×	×	0	0	0
0	1	×	×	×	×	×	×	0	0	1
0	0	1	×	×	×	×	×	0	1	0
0	0	0	1	×	×	×	×	0	1	1
0	0	0	0	1	×	×	×	1	0	0
0	0	0	0	0	1	×	×	1	0	1
0	0	0	0	0	0	1	×	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$A = \overline{D}_7\,\overline{D}_6\overline{D}_5\,\overline{D}_4\,D_3 + \overline{D}_7\,\overline{D}_6\overline{D}_5\,\overline{D}_4\,\overline{D}_3\,D_2 + \overline{D}_7\,\overline{D}_6\overline{D}_5\,\overline{D}_4\,\overline{D}_3\,\overline{D}_2 D_1 +$$

$$\overline{D}_7\,\overline{D}_6\overline{D}_5\,\overline{D}_4\,\overline{D}_3\,\overline{D}_2\overline{D}_1 D_0$$

$$B = \overline{D}_7\,\overline{D}_6 D_5 + \overline{D}_7\overline{D}_6\overline{D}_5 D_4 + \overline{D}_7\overline{D}_6\overline{D}_5\overline{D}_4\overline{D}_3\overline{D}_2 D_1 +$$

$$\overline{D}_7\,\overline{D}_6\overline{D}_5\,\overline{D}_4\,\overline{D}_3\,\overline{D}_2\overline{D}_1 D_0$$

$$C = \overline{D}_7\,D_6 + \overline{D}_7\,\overline{D}_6\,\overline{D}_5\,D_4 + \overline{D}_7\overline{D}_6\overline{D}_5\overline{D}_4\,\overline{D}_3 D_2 + \overline{D}_7\,\overline{D}_6\overline{D}_5\,\overline{D}_4\,\overline{D}_3\,\overline{D}_2\overline{D}_1 D_0$$

It is very complex to draw logic circuit. So, we can write as b encoder.

$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_2 + D_5 + D_7$

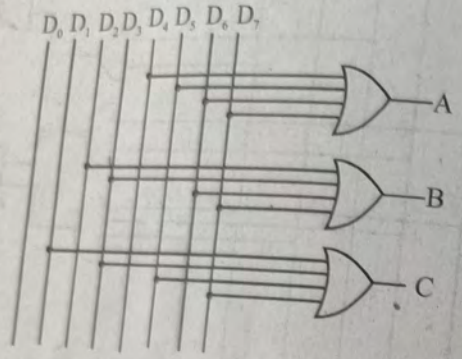


Fig.: Circuit diagram of an 8:3 encoder

4.3 Multiplexer (Data Selector)

Multiplex means 'many to one'. A multiplexer is a combinational circuit which selects single information from multiple inputs one at a time with help of selection line. Multiplexing is the process of transmitting a large number of information over a single line. For 'n' inputs, there are 'm' selection line and a single output where $2^m = n$.

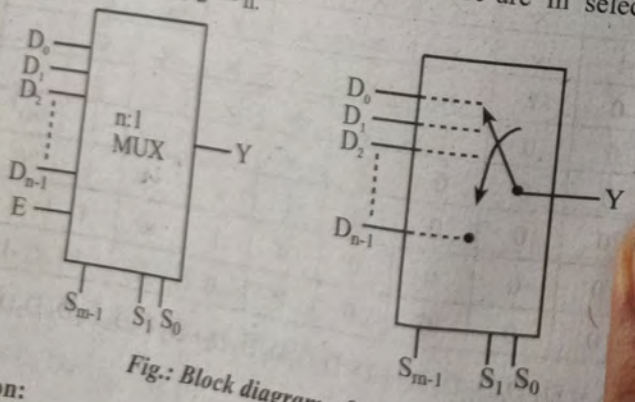


Fig.: Block diagram of a multiplexer.

Application:

- As a building block in the CPU.
- Used in the telecommunication at the transmitter side.

Advantage of multiplexer:

- It reduces the number of wire. Hence, it reduces the circuit complexity and cost.
- We can implement many combinational circuits using MUX.

4×1 MUX

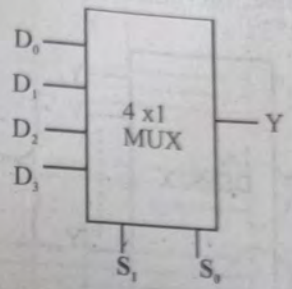


Fig.: Block diagram of a 4:1 multiplexer

Truth table

Inputs						Outputs
D_0	D_1	D_2	D_3	S_1	S_0	Y
1	0	0	0	0	0	1
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	0	0	1	1	1	1

$Y = \overline{S}_1 \overline{S}_0 D_0 + \overline{S}_1 S_0 D_1 + S_1 \overline{S}_0 D_2 + S_1 S_0 D_3$

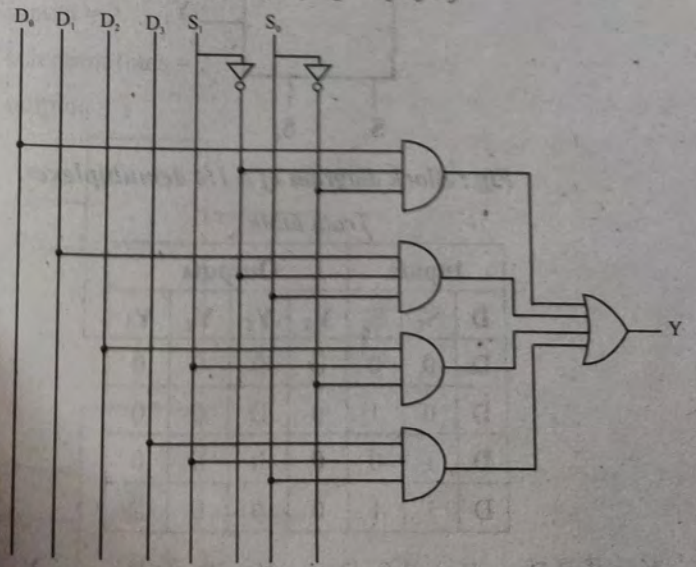


Fig.: Logic diagram of 4:1 MUX

4.4 Demultiplexer (Data Divider)

'Demultiplexer' means "one to many". A demultiplexer is a combinational circuit that receives information on a single input and transmits the same information over one of the possible output lines. Selection of output lines is controlled by selection line. For 'n' output, there is 'm' selection line and single input where $n = 2^m$.

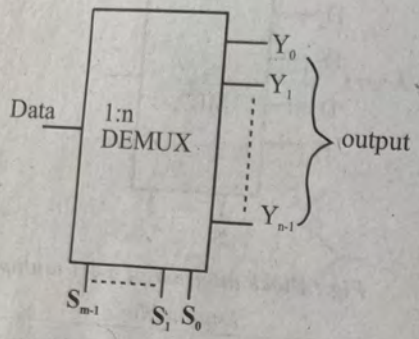


Fig.: Block diagram of a demultiplexer.

1:4 DEMUX

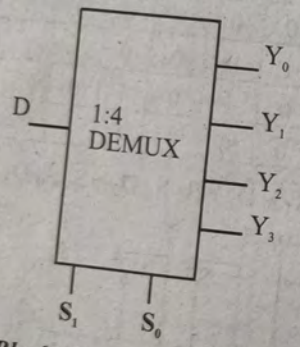


Fig.: Block diagram of a 1:4 demultiplexer.

Truth table

Inputs			Outputs			
D	S_1	S_0	Y_0	Y_1	Y_2	Y_3
D	0	0	D	0	0	0
D	0	1	0	D	0	0
D	1	0	0	0	D	0
D	1	1	0	0	0	D

$Y_0 = \overline{S}_1 \overline{S}_0 D_0$ $Y_1 = \overline{S}_1 S_0 D_1$ $Y_2 = S_1 \overline{S}_0 D_2$ $Y_3 = S_1 S_0$

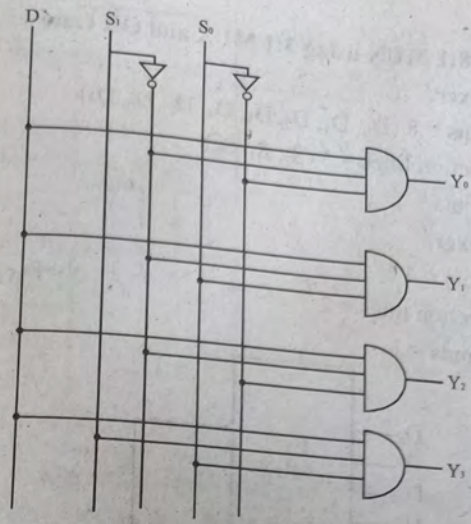


Fig.: Circuit diagram of a 1:4 demultiplexer

Construction of 4:1 Multiplexer using 2: 1 Multiplexer

For 4:1 multiplexer,

no. of inputs = 4 (D_0, D_1, D_2, D_3)

no. of selection lines = 2 (S_1, S_0)

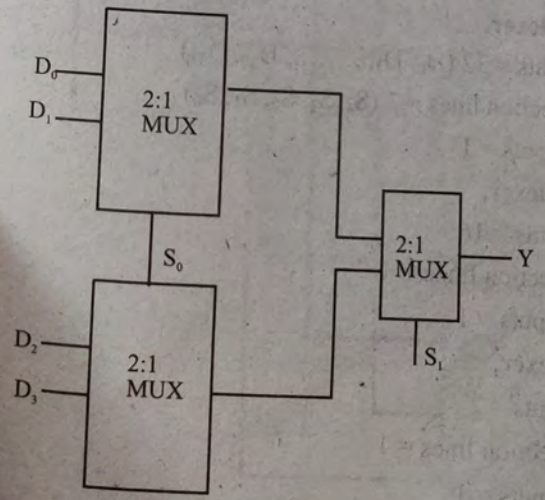
no. of outputs = 1

For 2:1 multiplexer,

no. of inputs = 2

no. of selection lines = 1

no. of outputs = 1



Construction of 8:1 MUX using 3:1 MUX and OR Gate

For 8:1 multiplexer,

no. of inputs = 8 (D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7)

no. of selection lines = 3 (S_2, S_1, S_0)

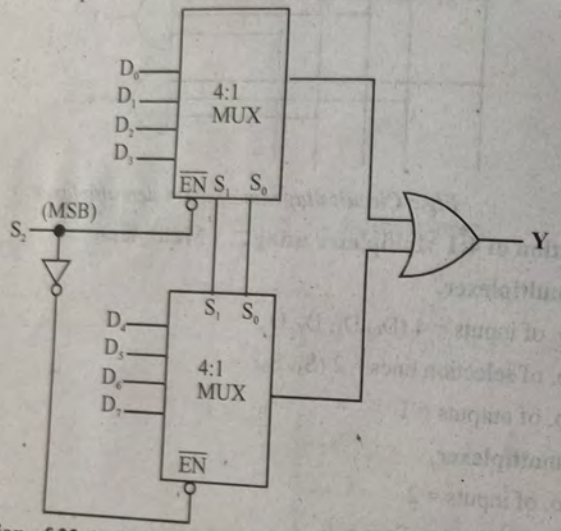
no. of outputs = 1

For 4:1 multiplexer,

no. of inputs = 4

no. of selection lines = 2

no. of outputs = 1



Construction of 32:1 MUX using 16 :1 MUX and 2:1 MUX

For 32:1 multiplexer,

no. of inputs = 32 (D_0, D_1, ……., D_{30}, D_{31})

no. of selection lines = 3 (S_4, S_3, S_2, S_1, S_0)

no. of outputs = 1

For 16:1 multiplexer,

no. of inputs = 16

no. of selection lines = 4

no. of outputs = 1

For 2:1 multiplexer,

no. of inputs = 2

no. of selection lines = 1

no. of outputs = 1

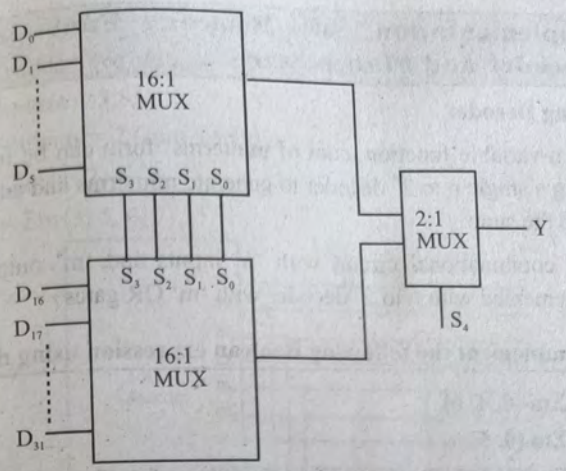
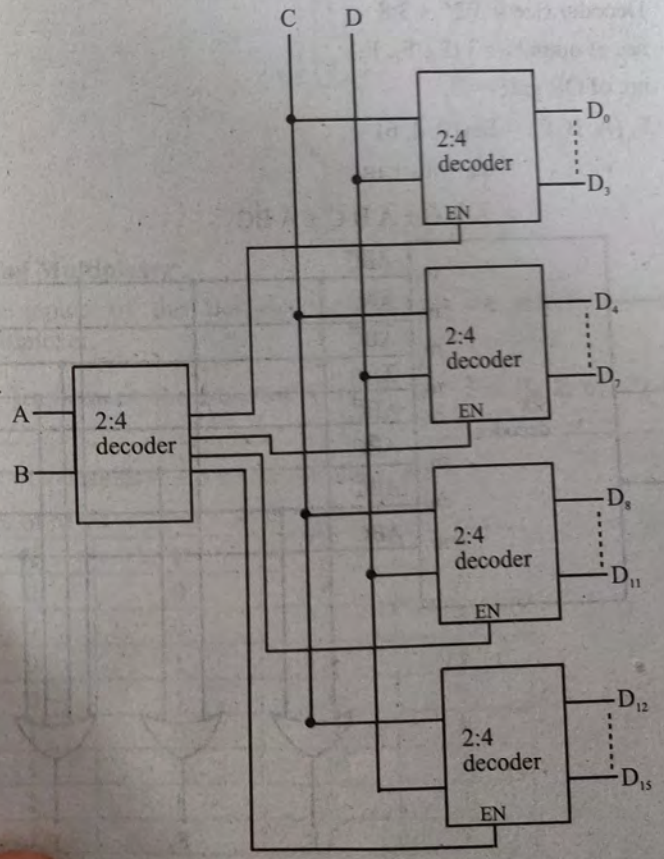


Fig.: 32:1 MUX using 16 :1 MUX and 2:1 MUX

Construction of 4:16 line decoder with five 2:4 line decoder with enable:



4.5 Implementation of Boolean Function using Decoder and Multiplexer

1. Using Decoder

For n-variable function, sum of minterms form can be implemented using a single n to 2^n decoder to generate minterms and an OR gate to form the sum.

Any combinational circuit with 'n' inputs and 'm' outputs can be implemented with n to 2^n decoder with 'm' OR gates.

Example: Implement the following Boolean expression using decoder.

$F_1 = \Sigma m\ (0, 4, 6)$

$F_2 = \Sigma m\ (0, 5)$

$F_3 = \Sigma m\ (1, 2, 3, 7)$ **where F is a function of A, B, and C**

⇒ No. of inputs = 3

Decoder size = $3:2^3 = 3:8$

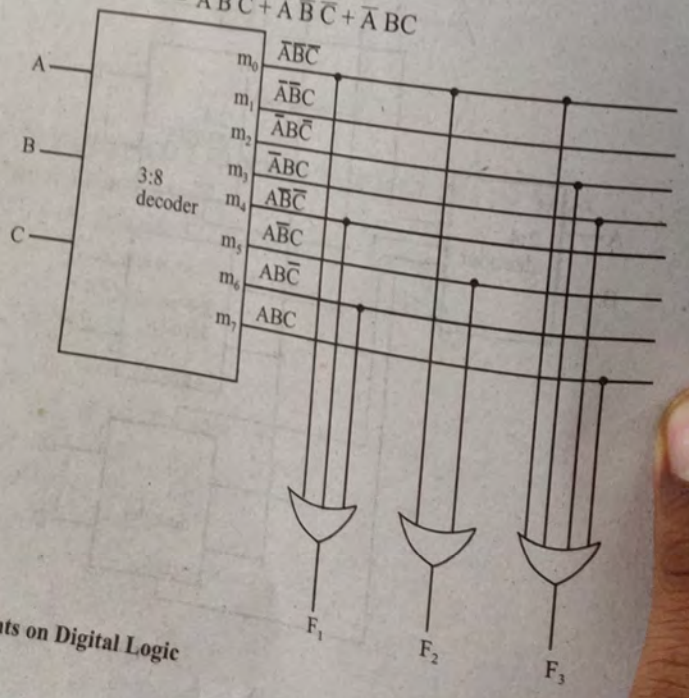
no. of outputs = 3 (F_1, F_2, F_3)

no. of OR gates = 3

$F_1 (A, B, C) = \Sigma m\ (0, 4, 6)$

$= m_0 + m_4 + m_6$

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



Example: Implement full adder using decoder.

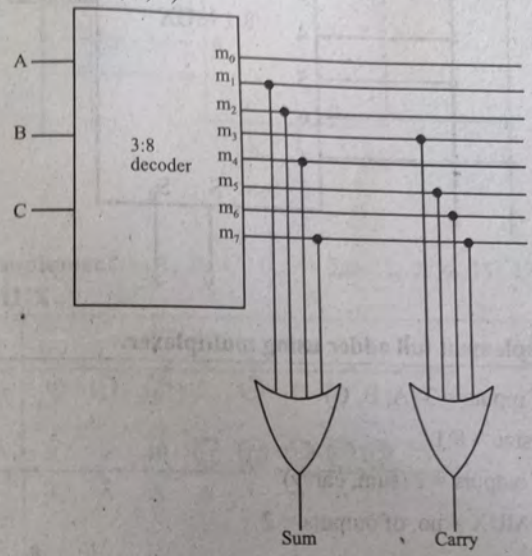
⇒ For full adder, no. of inputs = 3 (A, B, C)

Decoder size = 3:8

No. of outputs = 2 (sum, carry)

Sum = $\Sigma m\ (1, 2, 4, 7)$

Carry = $\Sigma m\ (3, 5, 6, 7)$



2. Using Multiplexer

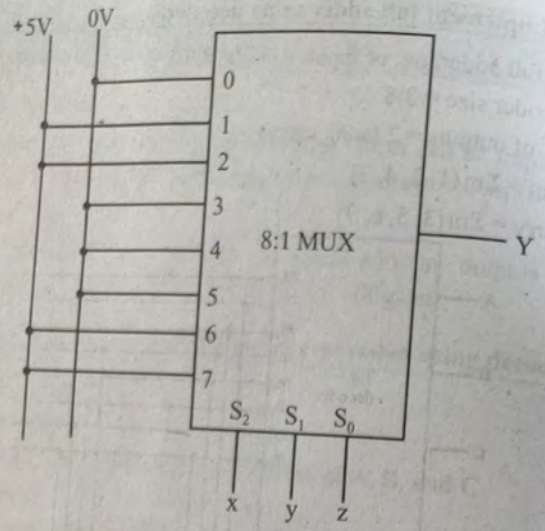
The inputs of the Boolean function are the selection lines of multiplexer.

Example: Implement the function F (x, y, z) = Σm (1, 2, 6, 7) using MUX.

⇒ No. of variables = 3 = no. of selection lines

Size of MUX = $2^3 \times 1 = 8 \times 1$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



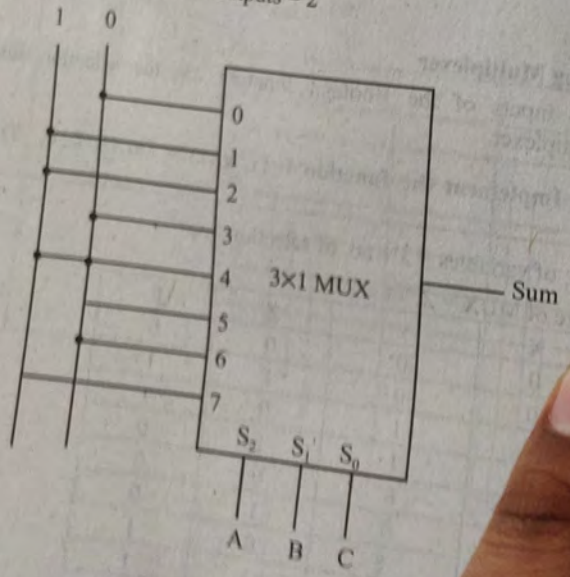
Example: Implement full adder using multiplexer.

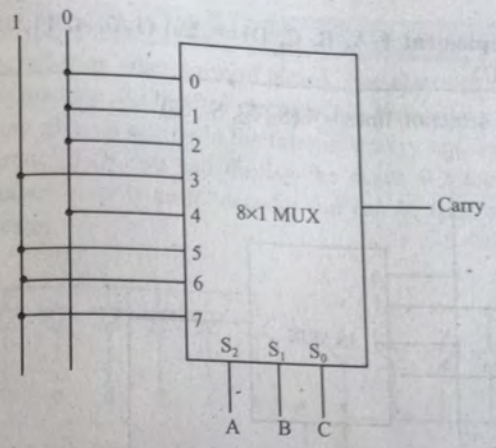
⇒ No. of inputs = 3 (A, B, C)

MUX size = 8:1

No. of outputs = 2 (sum, carry)

No. of MUX = no. of outputs = 2

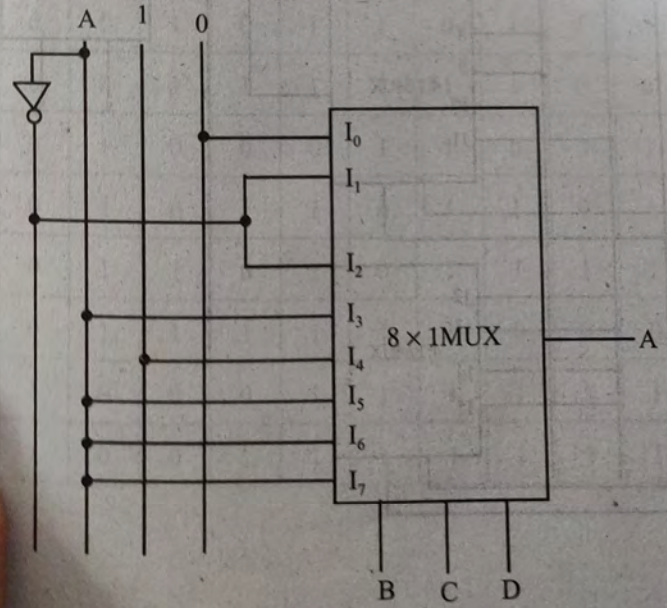




Example: Implement F(A, B, C, D) = Σm (1, 2, 4, 11, 12, 13, 14, 15) using 8×1 MUX.

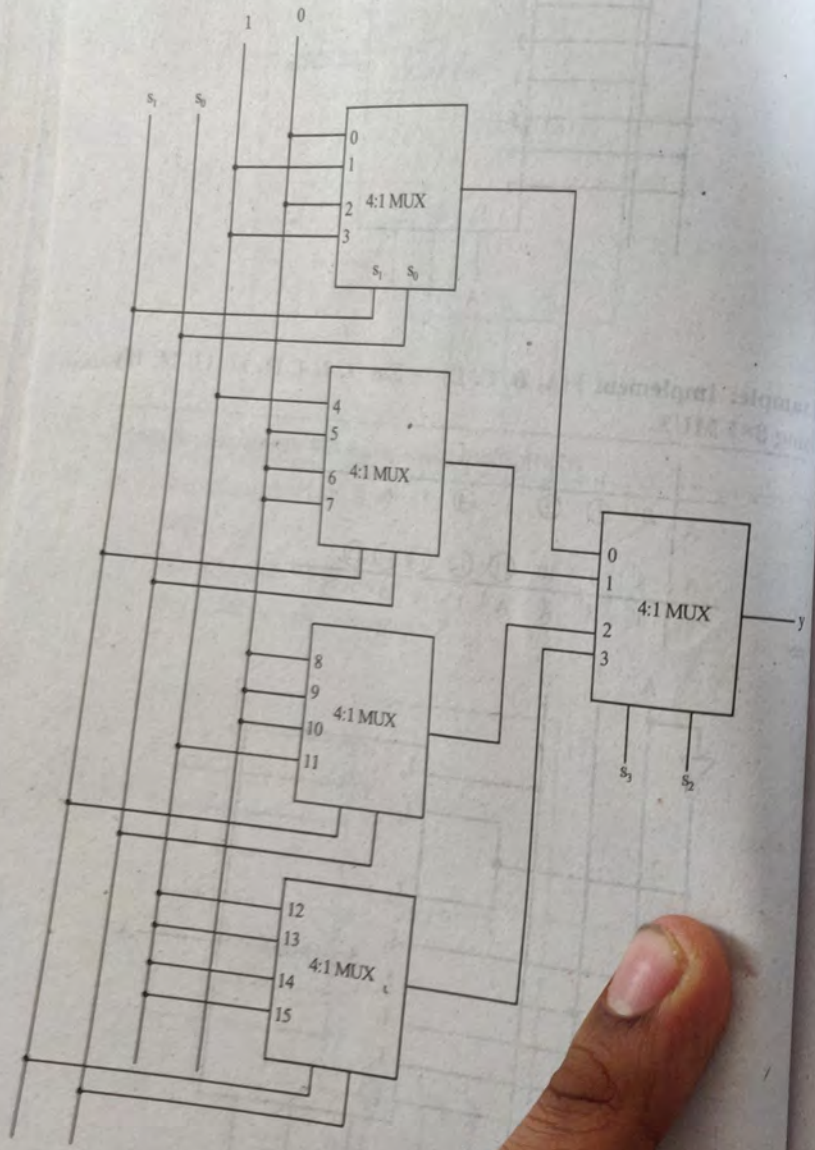
⇒

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\bar{A}$	0	①	②	3	④	5	6	7
A	8	9	10	⑪	⑫	⑬	⑭	⑮
	1	$\bar{A}$	$\bar{A}$	A	1	A	A	A



Example: Implement $F(A, B, C, D) = \Sigma m\ (1, 3, 4, 11, 12, 13, 14, 15)$ using 4×1 MUX.

⇒ No. of selection lines = 4 (S_3, S_2, S_1, S_0)



4.6 Seven Segment Decoder

A LED emits radiation when forward biased, free electrons recombine with holes near the junction. As the free electrons fall from higher energy level to lower one, they give up energy in the form of energy and light. By forward biasing different LEDs, we can display the digits 0 through 9. A seven segment decoder driver is an IC decoder that can be used to drive a seven segment indicator.

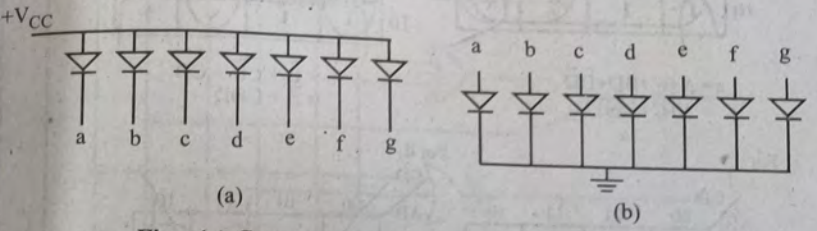
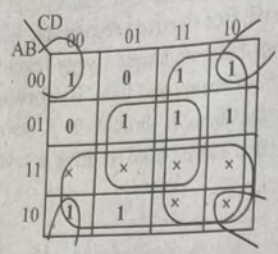


Fig.: (a) Common anode type (b) Common cathode type

Truth table for common cathode type

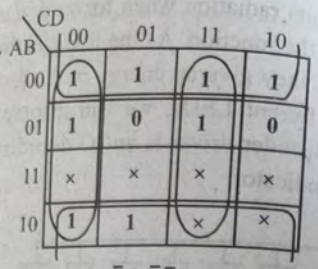
	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	1	1	1

For a,



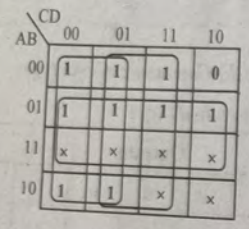
$a = A + C + BD + \bar{B}\bar{D}$
$= A + C + B \odot D$

For b.



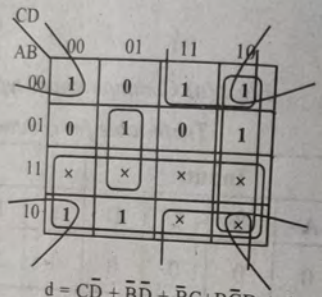
$b = \bar{B} + \bar{C}\bar{D} + CD$
$= \bar{B} + C \odot D$

For c,



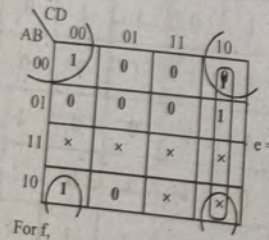
$c = \bar{C} + D + B$
$= B + \bar{C} + D$

For d,



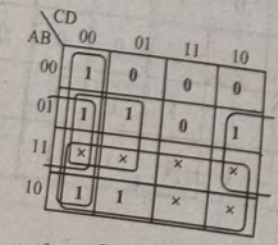
$d = C\bar{D} + \bar{B}\bar{D} + \bar{B}C + B\bar{C}D + A$
$= A + \bar{B}\bar{D} + \bar{B}C + C\bar{D} + B\bar{C}D$

For e,



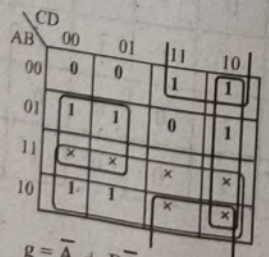
$e = C\bar{D} + \bar{B}\bar{D}$

For f,



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

For g,



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

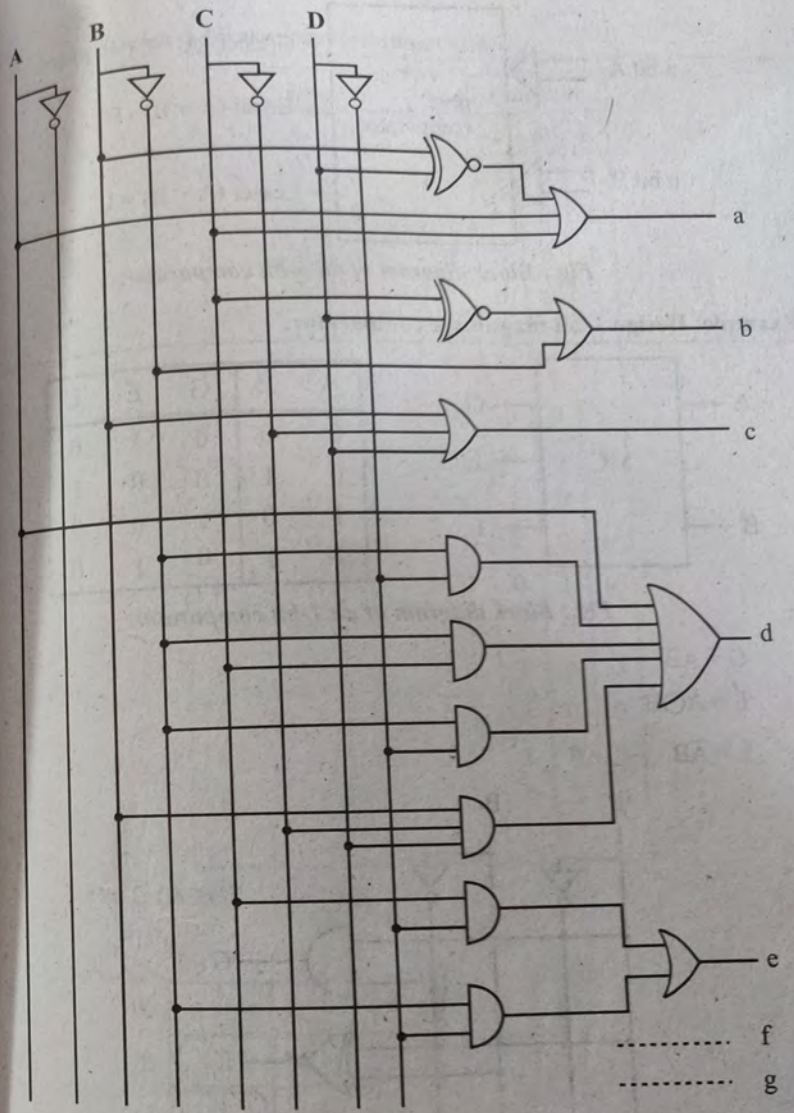


Fig.: Circuit diagram of a seven segment decoder

4.7 Magnitude Comparator

Magnitude comparator is a combinational circuit, designed to compare the two n-bit words applied as its input. The comparator has three outputs namely greater (A > B), lesser (A<B) and equal (A = B). Depending upon the result of comparison, one of these outputs will go high.

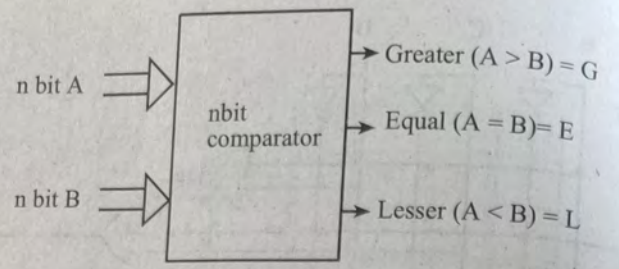
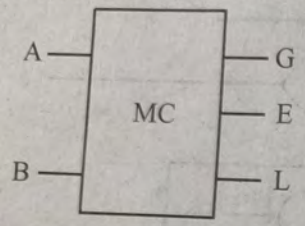


Fig.: Block diagram of an n-bit comparator.

Example: Design 1-bit magnitude comparator.



A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Fig.: Block diagram of an 1-bit comparator.

$G = A\overline{B}$

$E = A \odot B$

$L = \overline{A}B$

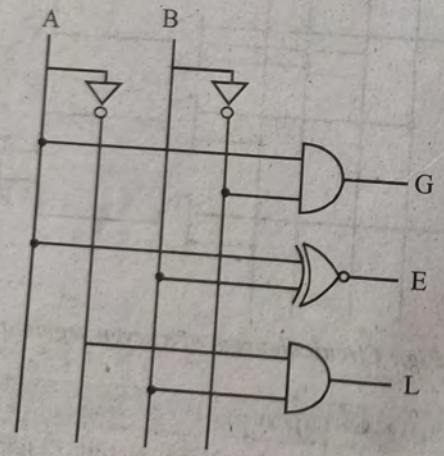
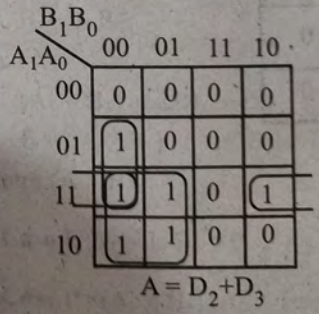


Fig.: Logic diagram of 1-bit magnitude comparator

Example: Design 2-bit magnitude comparator.

Inputs				Outputs		
A_1	A_0	B_1	B_0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

For G (A >B),



$A = D_2 + D_3$

$G = A_1\overline{B}_1 + A_0\overline{B}_1\overline{B}_0 + A_1A_0\overline{B}_0$

$= A_1\overline{B}_1 + A_0\overline{B}_0(\overline{B}_1 + A_1)$

For E,

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$E = \bar{A}_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + \bar{A}_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \bar{A}_0 B_1 \bar{B}_0$$

$$= \bar{A}_1 \bar{B}_1 (\bar{A}_0 \bar{B}_0 + A_0 B_0) + A_1 B_1 (A_0 B_0 + \bar{A}_0 \bar{B}_0)$$

$$= \bar{A}_1 \bar{B}_1 (A_0 \odot B_0) + A_1 B_1 (A_0 \odot B_0)$$

$$= (\bar{A}_1 \bar{B}_1 + A_1 B_1)(A_0 \odot B_0)$$

$$= (A_1 \odot B_1)(A_0 \odot B_0)$$

For L,

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$L = \bar{A}_1 B_1 + \bar{A}_1 \bar{A}_0 B_0 + \bar{A}_0 B_1 B_0$$

$$= \bar{A}_1 B_1 + \bar{A}_0 B_0 (\bar{A}_1 + B_1)$$

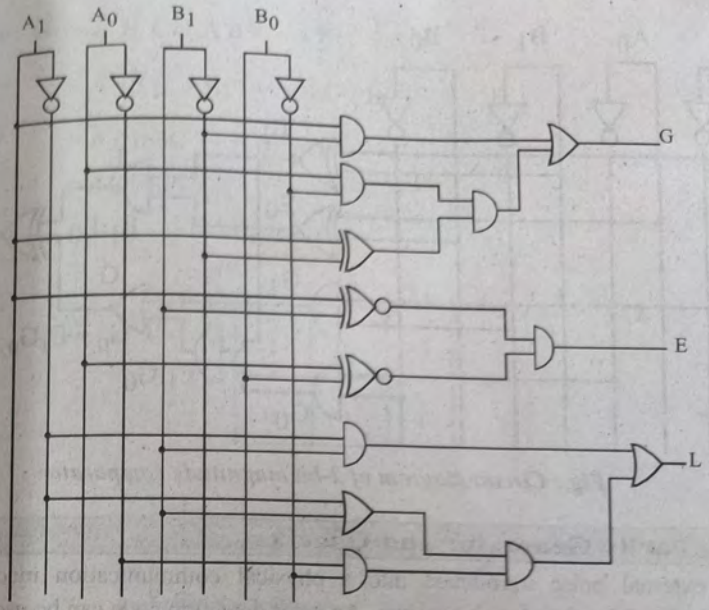


Fig.: Circuit diagram of 2-bit magnitude comparator

Alternative method for 2-bit magnitude comparator:

$$G = A\bar{B}$$

$$E = A \odot B$$

$$L = \bar{A} B \text{ or } (G + E)'$$

Let, $A = A_1 A_0$

$B = B_1 B_0$

For equal (A = B),

$(A_1 = B_1)$ and $(A_0 = B_0)$

$E = (A_1 \odot B_1) \cdot (A_0 \odot B_0) = E_1 \cdot E_0$

For greater (A > B),

Case I: $A_1 = 1, B_1 = 0 \Rightarrow A_1 \bar{B}_1$

Case II: $(A_1 = B_1)$ $(A_0 = 1$ and $B_0 = 0) \Rightarrow (A_1 \odot B_1) \cdot (A_0 \bar{B}_0)$

$\therefore G = A_1 \bar{B}_1 + (A_1 \odot B_1) A_0 \bar{B}_0 = G_1 + E_1 G_0$

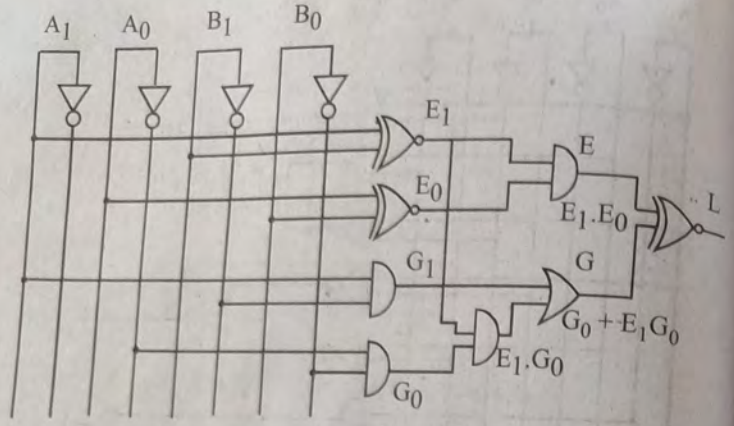


Fig.: Circuit diagram of 2-bit magnitude comparator

4.8 Parity Generator and Checker

Any external noise introduced into a physical communication medium changes bit value 0 to 1 or vice-versa. An error detection code can be used to detect error during transmission. The detected error cannot be corrected but its presence is indicated.

A parity bit is an extra bit included with a binary message to make a number of 1s either odd or even. The message including the parity bit is transmitted and then checked at the receiving end for errors. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.

Even parity means n-bit input has an even number of 1s. For example, 110011 has even parity. Odd parity means an n-bit input has an odd number of 1s. For example, 01011 has odd parity.

3-bit Odd Parity Generator

Information			Parity
A	B	C	P
0	0	0	1
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	0

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

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

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

$$= A \odot (B \oplus C)$$

For P

A \ BC	00	01	11	10
0	1	0	1	0
1	0	1	0	1

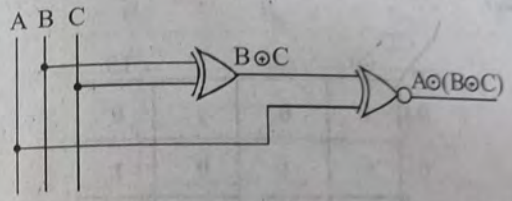


Fig.: Circuit diagram of 3-bit odd parity generator

The three input messages and the parity bit are transmitted to their destinations, where they are applied to a parity checker circuit. An error occurs if the parity of the four bit recurred is even, since the binary information transmitted was odd. So the output of the parity checker will be '1' when error occurs i.e., when the number of 1s in the four input is even.

4-bit Odd Parity Checker

A	B	C	D	Checker
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1

A	B	C	D	Checker
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Checker,

AB \ CD	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

$$\text{Checker} = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + ABCD + A\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$

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

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

$$= (A\odot B)(C\odot D) + (A\oplus B)(C\oplus D)$$

$$= \bar{X}\bar{Y} + XY \quad [X = A\oplus B,\ Y = C\oplus D]$$

$$= X\odot Y$$

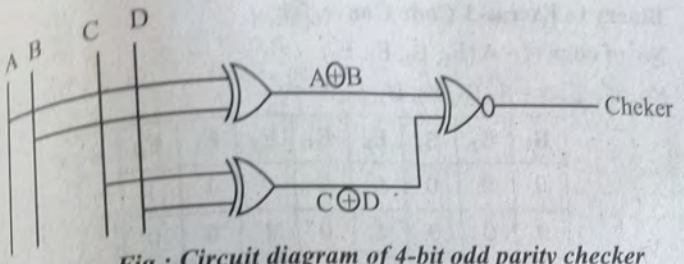


Fig.: Circuit diagram of 4-bit odd parity checker

4.9 Some Code Converter Circuits

1) 3-bit Binary to Gray Code Converter

No. of inputs = 3 (B_2, B_1, B_0)

No. of outputs = 3 (G_2, G_1, G_0)

Inputs			Outputs		
B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

For G_2

B_2 \ B_1B_0	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$G_2 = B_2$

For G_1

B_2 \ B_1B_0	00	01	11	10
0	0	0	1	1
1	1	1	0	0

$G_1 = \bar{B}_2B_1 + B_2\bar{B}_1 = B_2 \odot B_1$

For G_0

B_2 \ B_1B_0	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$G_0 = \bar{B}_1B_0 + B_1\bar{B}_0 = B_1 \odot B_0$

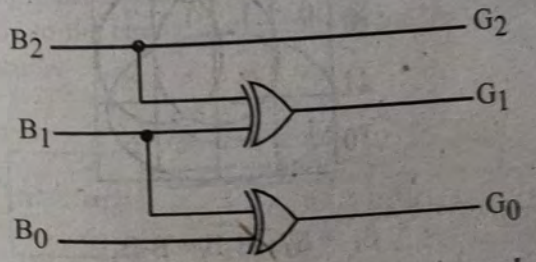


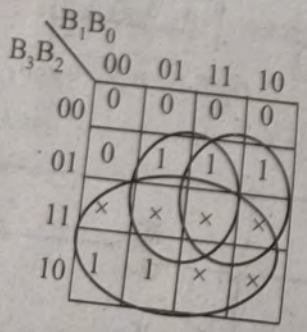
Fig.: Circuit diagram of 3-bit binary to gray code converter.

2) Binary to Excess-3 Code Converter

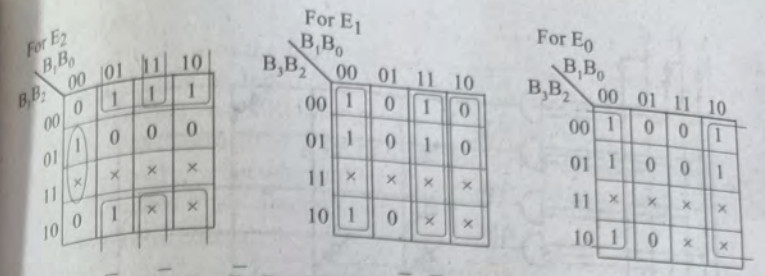
No. of outputs = 4 (E_3, E_2, E_1, E_0)

No. of inputs = 4 (B_3, B_2, B_1, B_0)

B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	×	×	×	×
1	0	1	1	×	×	×	×
1	1	0	0	×	×	×	×
1	1	0	1	×	×	×	×
1	1	1	0	×	×	×	×
1	1	1	1	×	×	×	×



$$E_3 = B_3 + B_2B_0 + B_2B_1$$
$$= B_3 + B_2(B_0 + B_1)$$



$E_2 = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$
$= B_2\bar{B}_1\bar{B}_0 + \bar{B}_2(B_0 + B_1)$

$E_1 = \bar{B}_1\bar{B}_0 + B_1B_0$
$= (B_1 \odot B_0)$

$E_0 = \bar{B}_0$

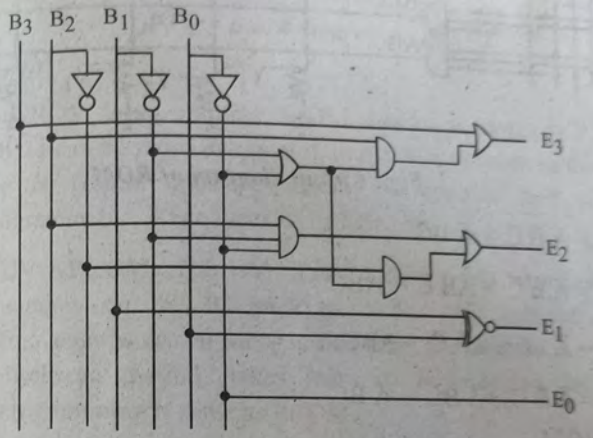


Fig.: Circuit diagram of binary to excess-3 code converter

4.10 Read Only Memory (ROM)

A ROM is essentially a memory (or storage) device in which a fixed set of binary information is stored. The binary information must be first specified by the user and then embedded in the unit to form the required inter connection pattern. ROM contains special internal links that can be fused or broken. Once a pattern is established for a ROM, it remained fixed even if the power/supply to the circuit is switched off and then switched on again i.e., it is non volatile.

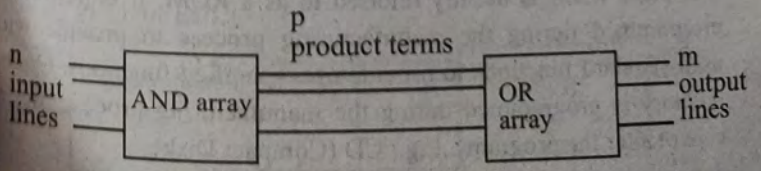


Fig.: Block diagram of ROM

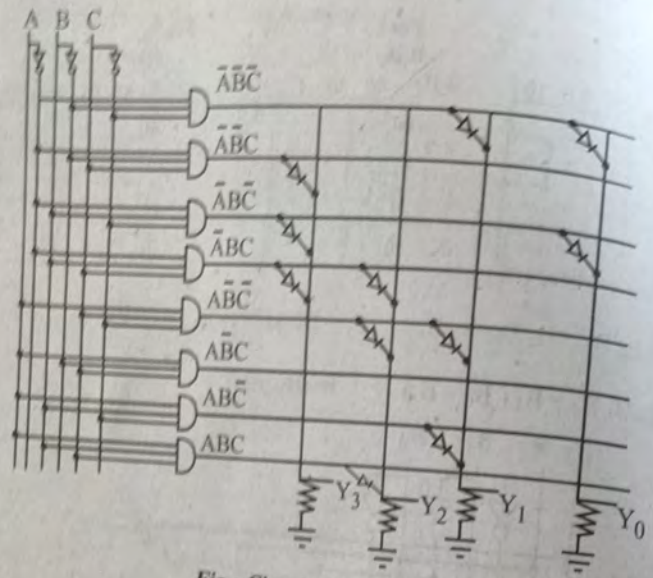


Fig.: Circuit diagram of ROM

$Y_0 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C}$

$Y_1 = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

$Y_2 = \bar{A}BC + A\bar{B}\bar{C} + ABC$

$Y_3 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC$

Types of ROM

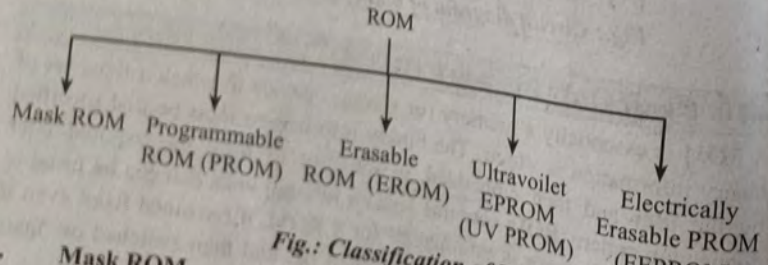


Fig.: Classification of ROM

1. **Mask ROM**

The mask ROM is usually referred to as a ROM. It is permanently programmed during the manufacturing process to provide widely used standard functions to provide user specified functions. Once the memory is programmed during the manufacturing process, the user cannot alter the programs. E.g., CD (Compact Disk)

2. **PROM**

The PROM uses the fusing process to store bits – the memory link is burned open or left intact to present 0 or 1. The fusing process is irreversible; once PROM is programmed, it cannot be changed.

A PROM consists of a set of fixed (non-programmable) AND gates connected as a decoder and a programmable OR array as shown in the generalized block diagram.

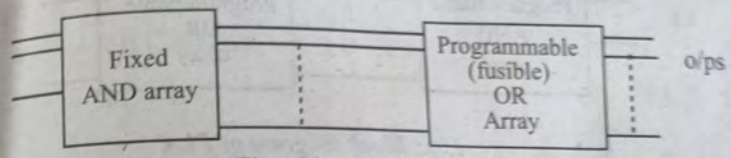


Fig.: Block diagram of PROM

3. **EPROM:**

An EPROM is an erasable PROM. Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased. Erasable PROM is erasable and electrically reprogrammable. Basic types of EPROM are:

(i) **UV EPROM:** The UV EPROM device has transparent quartz window on the IC package. When UV light of sufficient frequency is shown for specified time, the memory is erased. The discharge period takes several minutes to hours. The programming is same as PROM.

(ii) **EEPROM:** An electrically erasable PROM can be erased and programmed with electrical pulse. Since it can be both electrically written into and electrically erased, the EEPROM can be rapidly programmed, and erased in circuit for reprogramming.

4.11 Programmable Array Logic (PAL)

PAL is a programmable array of logic gates on a single chip. The basic PAL consists of a programmable AND array and a fixed OR array with output logic as shown in block diagram below. PAL is the most common one–time programmable (OTP) logic device and is implemented with bipolar technology (TTL or ECL).

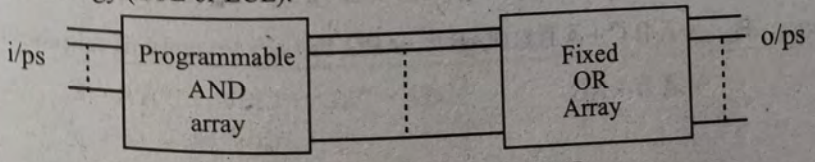


Fig.: Block diagram of PAL

4.12 Programmable Logic Arrays (PLA)

A PLA consists of a programmable AND array and a programmable OR array. The PLA is also called a Field Programmable Logic Array (FPLA) because the user in the field programs it, not the manufacturer.

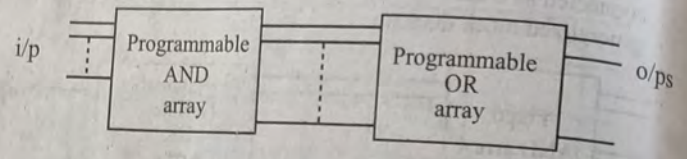


Fig.: Block diagram of PLA

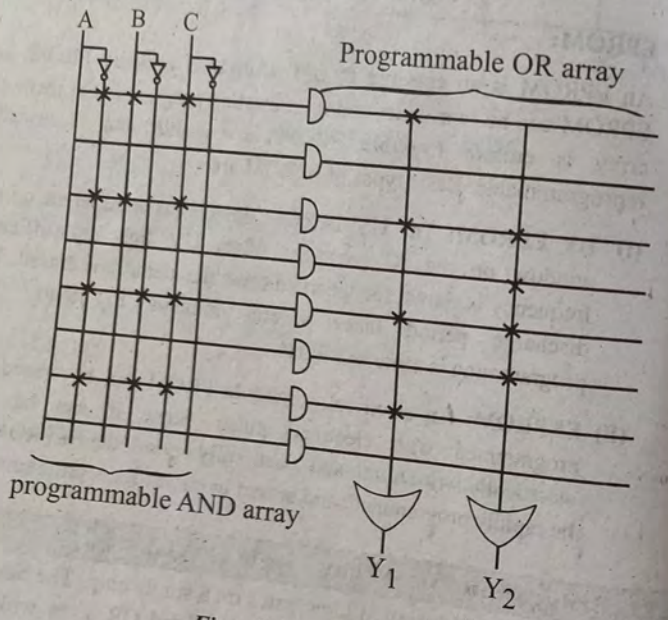


Fig.: Circuit diagram of PLA

Example: Implement function using PLA:

$F_0 = \Sigma_m (0, 1, 4, 6), F_1 = \Sigma_m (2, 3, 4, 6, 7), F_2 = \Sigma_m (0, 1, 2, 6)$

$\Rightarrow \quad F_0 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

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

$F_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C} + ABC$

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

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

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

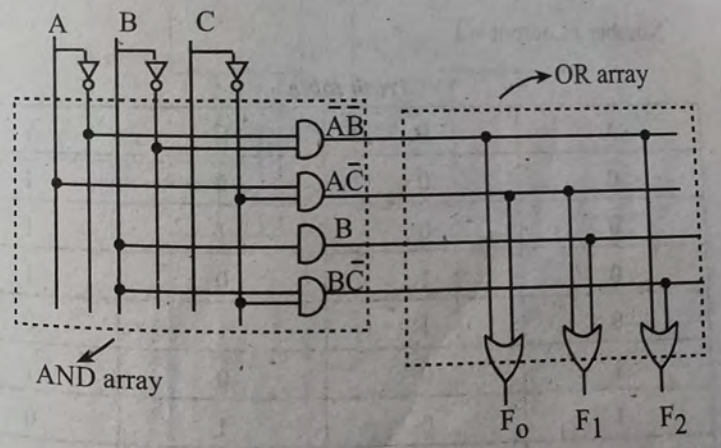
$= B + A\bar{C}$

$F_2 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}B\bar{C} + AB\bar{C}$

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

PLA table

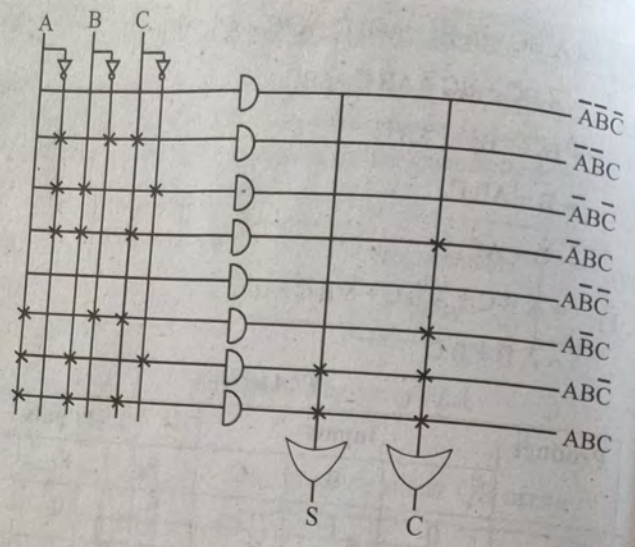
Product	Inputs			Outputs		
	A	B	C	F_0	F_1	F_2
$\bar{A}\bar{B}$	0	0	–	1	0	1
$A\bar{C}$	1	–	0	1	1	0
B	–	1	–	0	1	0
$B\bar{C}$	–	1	0	0	0	1



Example: Implement the full adder using PLA.

$\Rightarrow \quad S = A'B'C + A'BC' + ABC' + ABC$

$C = AB + AC + BC$



MORE WORKED OUT EXAMPLES

1. Design a combinational logic circuit with 3 input variables that will produce logic high output when more than one input variables are logic low. **[2071 Chaitra]**

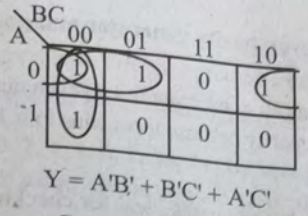
⇒ Total number of inputs = 3

Number of output = 1

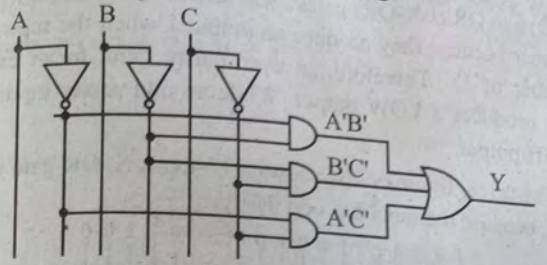
Truth table

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Implementing the K-map of above truth table, we get



$Y = A'B' + B'C' + A'C'$



2. Design a 32 to 1 multiplexer using 16 to 1 and 2 to 1 multiplexers.

⇒

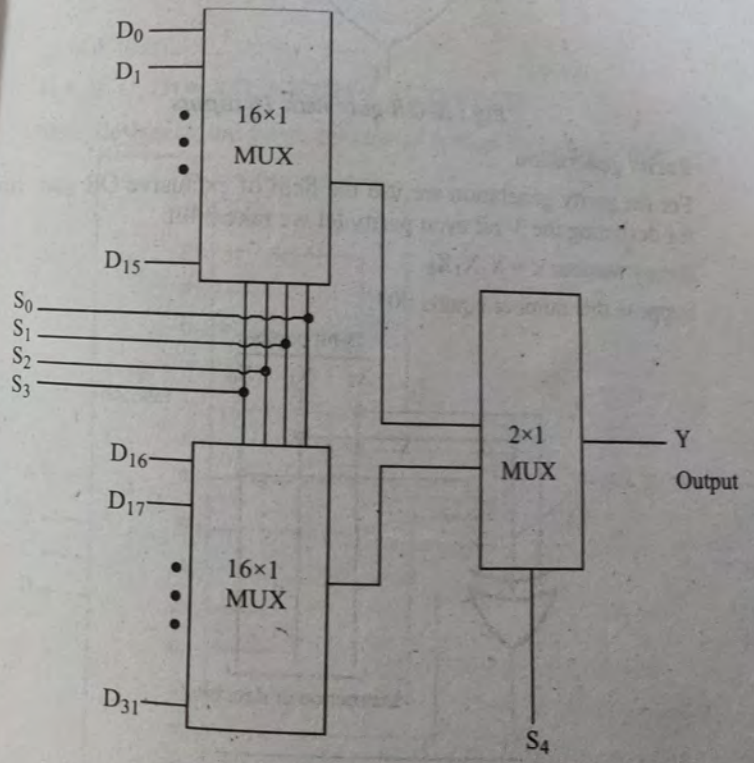


Fig.: Realization of 32×1 MUX by using 16×1 and 2×1 MUXes.

3. Design a 3-bit even parity generator and parity checker. **[2068 Shrawan]**

⇒ Even parity means an n–bit input has an even number of 1s. For instance, 110011 has even parity because it contains four 1s.

Parity checker

Exclusive OR (EX-OR) gates ideal for checking the parity of a binary number because they produce an output 1 when the input has an odd number of 1's. Therefore, an even parity input to an exclusive OR gate produces a LOW output. While an odd parity input produces a HIGH output.

For instance, figure below shows a 16-input X–OR gate the output is of 1, because the input has odd parity.

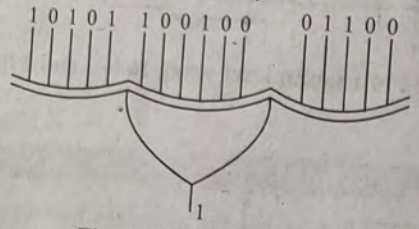


Fig.: X-OR gate with 16 inputs

Parity generation

For the parity generation we use the help of exclusive OR gate. Here for designing the 3–bit even parity let we take 3 bit

Binary number $x = X_2X_1X_0$

Suppose this number equals 001.

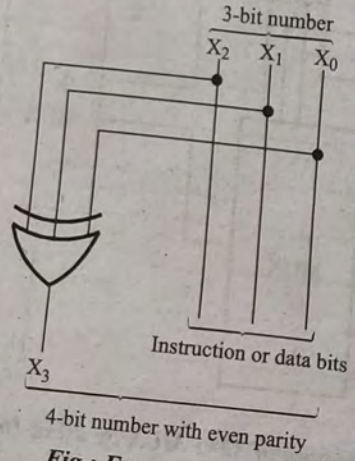


Fig.: Even parity generation

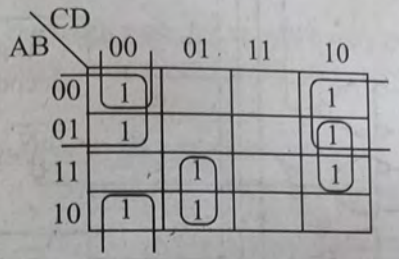
Since the 3 bits (input bits) are 001. Then the number has odd parity, which means the exclusive OR gate produces an output of 1.

The final 4–bit output in 1001. Notice that this has even parity.

4. Simplify F = Σ (0, 2, 4, 6, 8, 9, 13, 14) by using K–map. Implement the given circuit using decoder (use only the block diagram of decoder). **[2068 Shrawan]**

$F(A, B, C, D) = \Sigma(0, 2, 4, 6, 8, 9, 13, 14)$

⇒ Using K–map:



In SOP form,

$F(A, B, C, D) = A'D' + B'C'D' + AC'D + BCD'$

Now, designing the given functional system by using decoder.

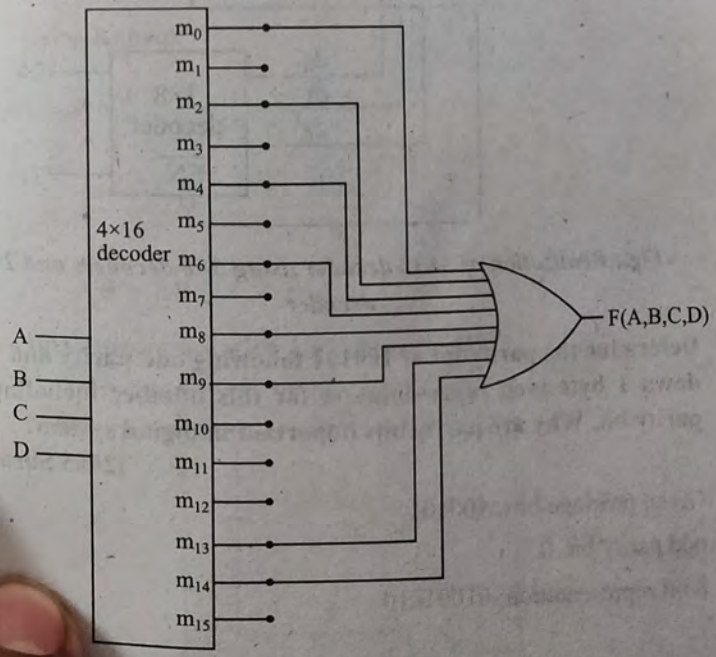


Fig.: 4×16 decoder circuit

5. **Construct a 5 × 32 decoder using 3 to 8 decoders and standard logic gates if necessary.** **[2067 Ashadh]**

⇒

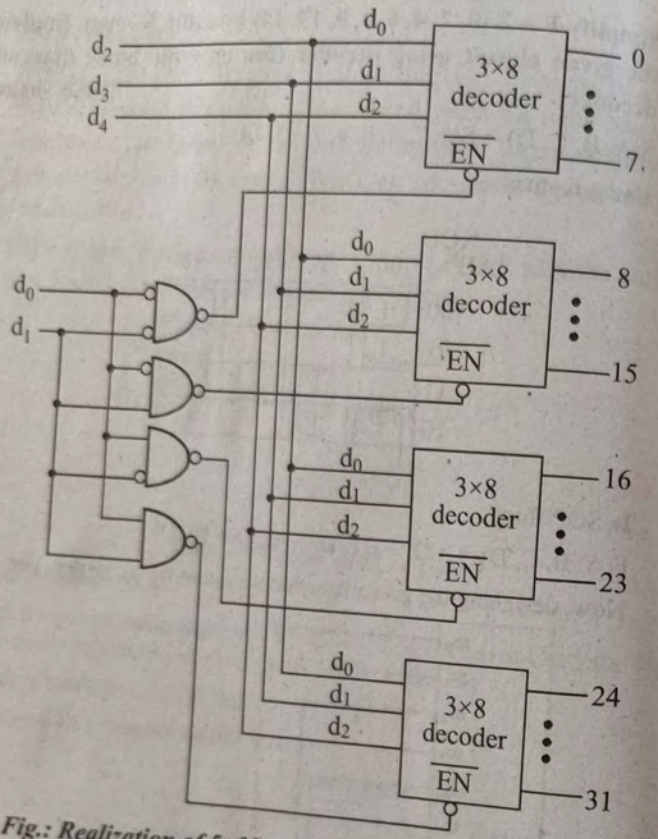


Fig.: Realization of 5×32 decoder using 3×8 decoders and 2×4 decoder

6. **Determine the parity bit of 100101 following odd parity and write down 1 byte long representation for this number including the parity bit. Why are parity bits important in digital system?** **[2065 Shrawan]**

⇒ Given message bits: 100101

odd parity bit: 0

8 bit representation: 01001010

Importance of parity bit in digital system:

A Parity bit is a scheme for detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even. The message, including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity doesn't to the one transmitted. The circuit that generates the parity bit is called parity generator and the circuit that checks the parity in the receiver is called a parity checker.

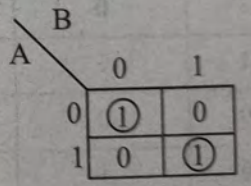
7. **Write down the truth table for an odd parity generator for any 2-bit word. Design your circuit using any universal gate of your choice.** **[2065 Shrawan]**

⇒

Truth table for odd parity generator

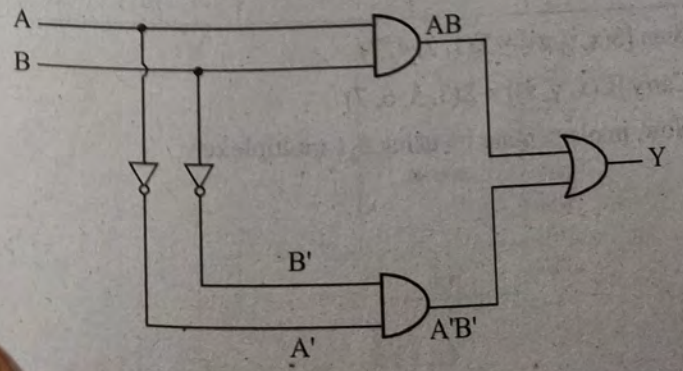
Message		Odd parity bit
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Using K–map.

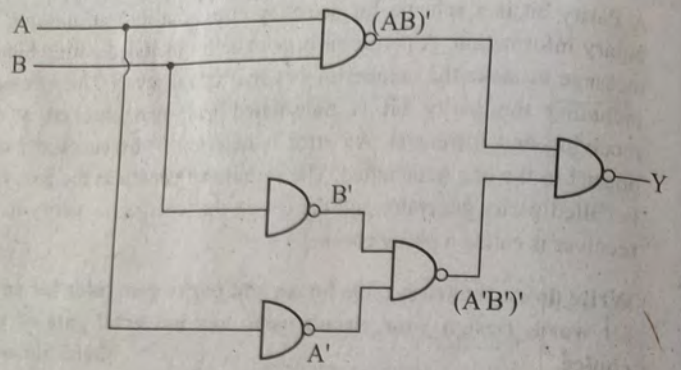


$Y = A'B' + A\bar{B} = A \odot B$

Implementation with AND and OR gates



Implementation with universal (NAND) gates only



8. **Implement outputs of a full adder circuit by using 8:1 multiplexer.** **[2064 Jestha]**

⇒ For full adder circuit; let us suppose that A, B and C are inputs and S and C are sum and carry respectively.

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

Sum $[S(x, y, z)] = \Sigma(1, 2, 4, 7)$

Carry $[C(x, y, z)] = \Sigma(3, 5, 6, 7)$

Now, implementing by using 8:1 multiplexer.

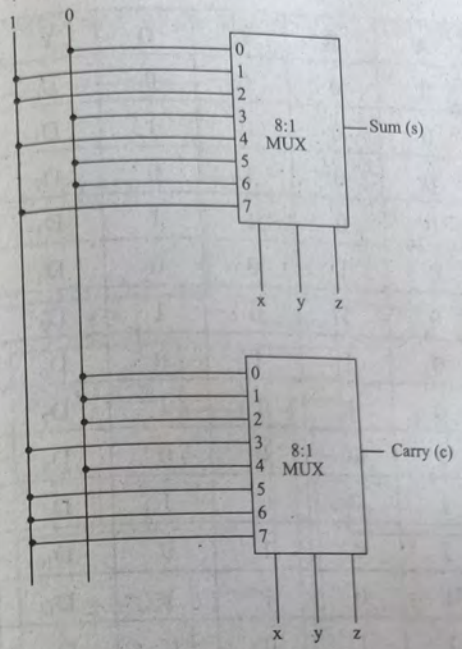
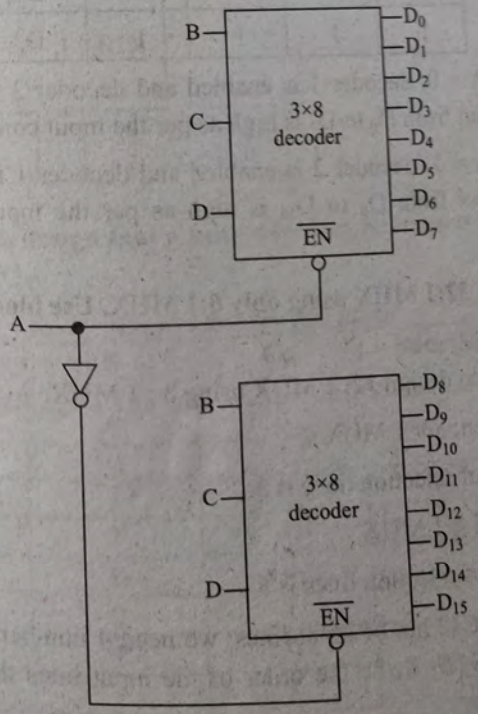


Fig.: Implementation of a full adder using 8:1 multiplexer

9. **Construct 4×16 decoder using 3×8 decoder.**



A	B	C	D	Y
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_2
0	0	1	1	D_3
0	1	0	0	D_4
0	1	0	1	D_5
0	1	1	0	D_6
0	1	1	1	D_7
1	0	0	0	D_8
1	0	0	1	D_9
1	0	1	0	D_{10}
1	0	1	1	D_{11}
1	1	0	0	D_{12}
1	1	0	1	D_{13}
1	1	1	0	D_{14}
1	1	1	1	D_{15}

When A = 0, decoder 1 is enabled and decoder 2 is disabled. Hence the output from D_0 to D_7 is high as per the input combination of BCD.

When A = 1, decoder 2 is enabled and decoder 1 is disabled. Hence the output from D_8 to D_{15} is high as per the input combination of ABC.

10. Design a 32:1 MUX using only 8:1 MUX. Use block diagrams. **[2071 Chaitra]**

⇒ We have to design 32: 1 MUX using 8 : 1 MUX.

Given: 8 × 1 MUX

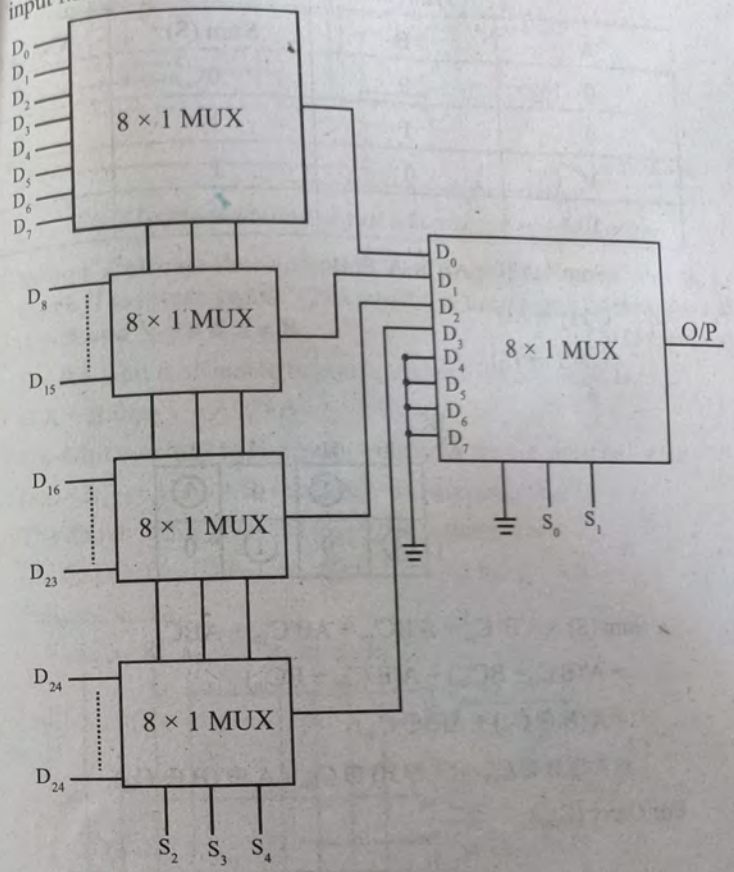
No. of selection lines is 3.

Design: 32 × 1 MUX

Here, no. of selection lines is 5

Thus to get 32 no. of input lines, we need 4 number of 8 × 1 MUX..

Let S_0 S_1 S_2 S_3 S_4 be the order of the input lines then S_0 S_1 acts as selector of MUX and $S_2S_3S_4$ acts as input to the selection line to the input MUX.



11. Show with design that a full-adder can be implemented using two half adders. **[2071 Chaitra]**

⇒ ***Truth table for full adder***

A	B	C_{in}	Sum (S)	Carry (C_{out})
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table of half adder

A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\therefore$ Sum = A'B + AB"= A $\oplus$ B

Carry = AB

For Sum (S),

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

$\therefore$ Sum (S) = $A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$

$= A'(B'C_{in} + BC'_{in}) + A(B'C'_{in} + BC_{in})$

$= A'(B \oplus C_{in}) + A(B \oplus C_{in})'$

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

For Carry (C_{out}),

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

$\therefore$ Carry (C_{out}) = $AB + BC_{in} + AC_{in}$

$= AB + ABC_{in} + A'BC_{in} + ABC_{in} + AB'C_{in}$

$= AB(1 + C_{in}) + A'BC_{in} + AB'C_{in}$

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

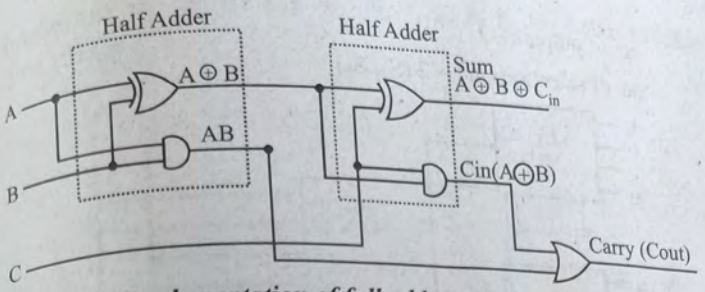


Fig.: Implementation of full adder using two half adders.

12. **Design a circuit that compares two 4- bit number, A and B, to check if they are equal. The circuit has one output so that x = 1 if A = B and X = 0 if A ≠ B** **[2073 Chaitra]**

⇒ For the 1-bit numbers to be equal, we have the condition as:

If A = B then $x = A \odot B$

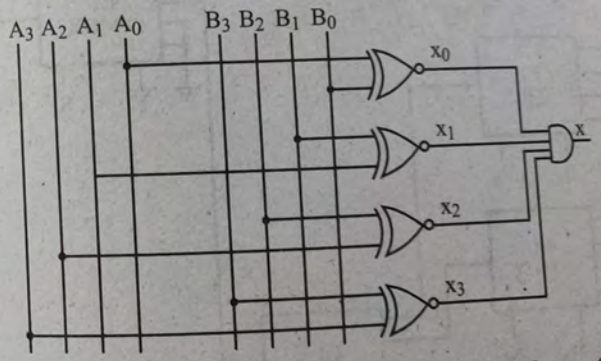
For 4-bit numbers to be equal (A = B), all the bits must be equal:

$(A_3 = B_3)$ and $(A_2 = B_2)$ and $(A_1 = B_1)$ and $(A_0 = B_0)$

The above result gives the logic gate formation as:

$(A_3 \odot B_3).(A_2 \odot B_2).(A_1 \odot B_1).(A_0 \odot B_0)$

$= x_3.x_2.x_1.x_0$



13. **Design the 32:1 multiplexer using 4:1 multiplexer tree concept and implement the function F = Σ(0, 1, 3, 8, 9, 13) using suitable multiplexer.** **[2074 Chaitra]**

⇒ For 32:1 multiplexer,

no. of inputs = 32(I_0, I_1, I_2..... I_{31})

no. of outputs = 1

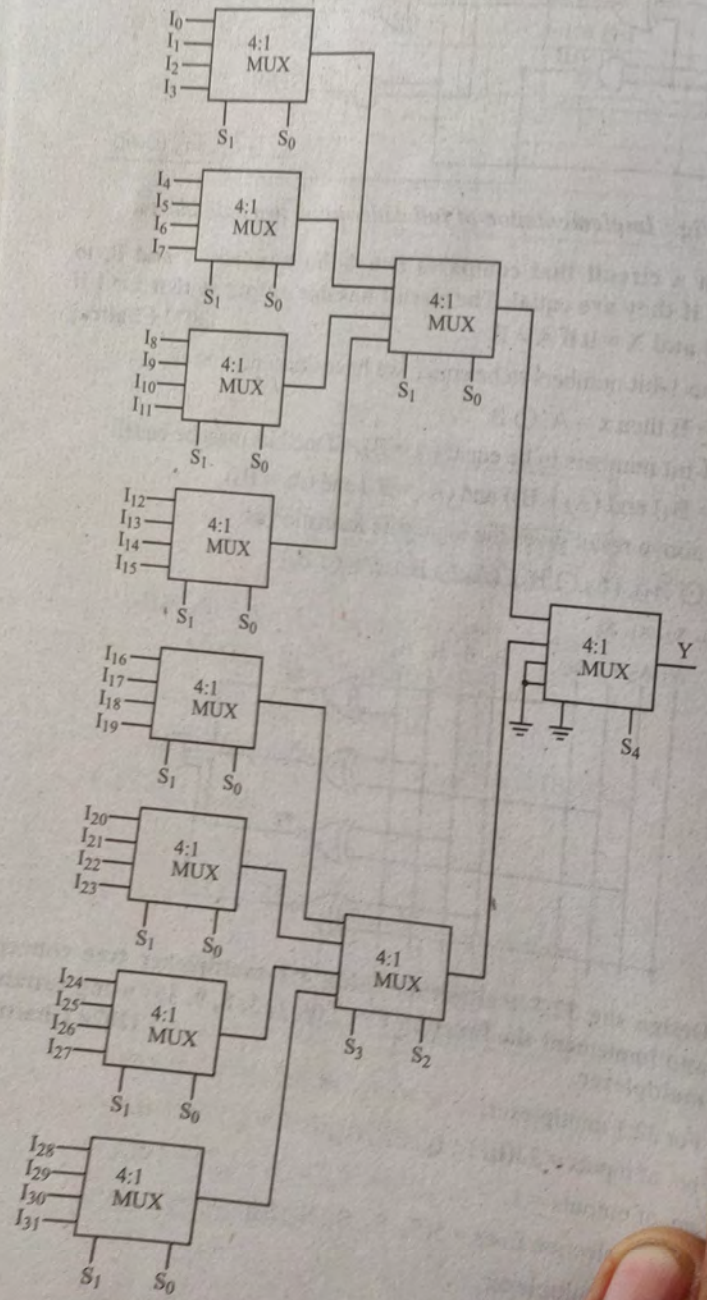
no. of selection lines = 5(S_4, S_3, S_2, S_1, S_0)

For 4:1 multiplexer,

no. of inputs = 4(I_0, I_1, I_2, I_3)

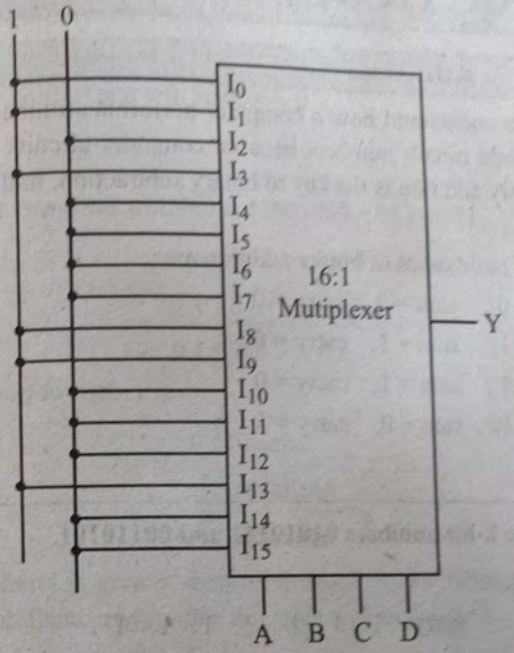
no. of outputs = 1

no. of selection line = 2 (S_1, S_0)



14. **Implement the function $F = \Sigma(0, 1, 3, 8, 9, 13)$ using suitable multiplexer.**

⇒



Chapter – 5

ARITHMETIC CIRCUITS

5.1 Binary Addition

Before we can understand how a computer performs arithmetic, we have to learn how to add binary numbers because computer circuits process binary numbers. Binary addition is the key to binary subtraction, multiplication, and division.

The four most basic cases of binary addition are:

$0 + 0 = 0$; sum = 0, carry = 0

$0 + 1 = 1$; sum = 1, carry = 0

$1 + 0 = 1$; sum = 1, carry = 0

$1 + 1 = 10$; sum = 0, carry = 1

EXAMPLE:

1. **Add two 8-bit numbers 01010111 and 00110101.**

⇒

carry	1	1	1		1	1	1	
	0	1	0	1	0	1	1	1
+	0	0	1	1	0	1	0	1
	1	0	0	0	1	1	0	0

2. **Add two 16-bit numbers 0000111110101100 and 0011100001111111.**

⇒

0000	1111	1010	1100
0011	1000	0111	1111
0100	1000	0010	1011

To add 16-bit numbers in first generation microcomputer, it takes two operations: first it adds lower 8 bits in one operation and then upper 8 bits in another.

Addition format:

Augend
Addend
\+ Sum

5.2 Binary Subtraction

The four cases of binary subtraction are:

$0 - 0 = 0$; difference = 0, borrow = 0

$0 - 1 = 1$; difference = 1, borrow = 1

$1 - 0 = 1$; difference = 1, borrow = 0

$1 - 1 = 0$; difference = 0, borrow = 0

EXAMPLE:

1. **Subtract two 8-bit numbers: 11001000 – 01111101**

	1100	1000
	0111	1101
	0100	1011

Subtraction format:

Minuend
– Subtrahend
Difference

If subtrahend is greater than minuend, then we subtract the minuend from subtrahend and put the negative sign in the result.

5.3 Unsigned Binary Numbers

The binary number which only represents the magnitude is an unsigned binary number. In this representation we forget '+' and '–' sign and concentrate on magnitude (absolute value) of a number. All the bits are used to represent magnitude.

For 8-bit number, the range is

(0000 0000) to (1111 1111)

or, $(00)_H$ to $(FF)_H$

or, $(0)_{10}$ to $(255)_{10}$

For 16-bit number, total range is

(0000 0000 0000 0000) to (1111 1111 1111 1111)

or, $(0000)_H$ to $(FFFF)_H$

or, $(0)_{10}$ to $(65535)_{10}$

5.4 Signed Magnitude Numbers

What to do when data have positive and negative values? In this case, we use signed magnitude number. A signed binary number consists of both sign and the magnitude information. In signed binary number representation, the leftmost bit is sign bit and the remaining is magnitude bits. For positive number, sign bit is 0 and for negative number, sign bit is 1.

Here are some examples of converting sign-magnitude numbers.

$+7 \rightarrow 0000\ 0111$

$-16 \rightarrow 1001\ 0000$

$+25 \rightarrow 0000\ 0000\ 0001\ 1001$

$-128 \rightarrow 1000\ 0000\ 1000\ 0000$

For 8-bit number, the negative numbers are

1000 0001 (−1) to 1111 1111 (−127)

Remember that for 8-bit number, positive numbers are

0000 0001 (+1) to 0111 1111 (+127)

So, for 8-bit arithmetic, the range is − 127 to + 127.

The main advantage of sign-magnitude numbers is their simplicity. Negative numbers are identical to positive numbers, except for the sign bit. However, sign-magnitude numbers have limited use because they require complicated arithmetic circuits.

5.5 Complements

Complements are used in digital computers to simplify the subtraction operation and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations. For base-r system, there are two types of complement.

The radix complement (r's complement)

The diminished radix complement ((r-1)'s complement)

1) **The r's Complement (Radix Complement)**

r's complement of $N = r^n - N$

where n = no. of digits, N = Number, r = radix

EXAMPLE:

1. **Find 10's complement of $(52520)_{10}$.**

⇒ 10's complement of $(52520)_{10} = 10^5 - 52520$
$= 47480$

Here, number of digits n = 5.

2. **Find 10's complement of $(305)_{10}$.**

⇒ 10's complement of $(305)_{10} = 10^3 - 305$
$= 1000 - 305$
$= 695$

3. **Find r's complement of $(0.252)_{10}$.**

⇒ As there is no integer part, n = 0
10's complement of $(0.252)_{10} = 10^n - N$
$= 10^0 - 0.252$
$= 1 - 0.252$
$= 0.748$

4. **Find 2's complement of $(10101)_2$.**

⇒ 2's complement of $(10101)_2 = 2^5 - 10101$
$= (100000) - (10101)_2$
$= 1011$

5. **Find 2's complement of $(0.0110)_2$**

⇒ Here, n = 0
2's complement of $(0.0110)_2 = 2^0 - 0.0110$
$= 1 - 0.0110$
$= 0.1010$

2) **The (r–1)'s Complement (Diminished Radix Complement)**

(r–1)'s complement of $N = r^n - r^{-m} - N$ where N = number, r = radix, n = an integer of n digits, m = a fractional part of m digits.

EXAMPLE:

1. **Find 9's complement of $(52520)_{10}$**

⇒ Here, n = 5, m = 0
So, 9's complement is $= 10^5 - 10^0 - 52520$
$= 100000 - 1 - 52520$
$= 47479$

2. **Find 9's complement of $(0.252)_{10}$**

⇒ 9's complement of $(0.252)_{10} = 10^0 - 10^{-3} - 0.252$

$= 1 - 0.001 - 0.252$

$= 0.999 - 0.252$

$= 0.747$

3. **Find 9's complement of $(25.639)_{10}$**

⇒ 9's complement of $(25.639)_{10} = 10^2 - 10^{-3} - 25.639$

$= 100 - 0.001 - 25.639$

$= 73.36$

4. **Find 1's complement of $(101100)_2$**

⇒ 1's complement of $(101100)_2 = 2^6 - 2^0 - 101100$

$= (2^6 - 1) - 101100$

$= 111111 - 101100$

$= 010011$

5. **Find 1's complement of $(0.0110)_2$**

⇒ 1's complement of $(0.0110)_2 = (2^0 - 2^{-4}) - 0.0110$

$= (1 - 2^{-4}) - 0.0110$

$= 0.1111 - 0.0110$

$= 0.1001$

Note: $(1 - 2^{-4}) = 0.9375$ and was converted into binary in the above problem.

5.6 2's Complement Representation

1's Complement

The 1's complement of a binary number is the number that results when we complement each bit. That is, 1's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s.

2's Complement

The 2's complement is the binary number that results when we add 1 to the 1's complement. That is,

2's complement = 1's complement + 1

Subtraction is done using 2's complement method widely. This leads to the simplest logic circuits for performing arithmetic.

Negative numbers are represented by 2's complement. Since negative number is the 2's complement of positive number, in order to represent –7,

+7 → 00000111

–7 → 11111000 + 1 = 11111001

EXAMPLE:

1. **Find the 2's complement of 11010011**

⇒

1 1 0 1 0 0 1 1	Binary number
0 0 1 0 1 1 0 0	1's complement
+1	
0 0 1 0 1 1 0 1	2's complement

2. **Express – 39 as an 8–bit sign number in sign magnitude form, 1's complement, and 2's complement form.**

⇒ In binary, +39 = 00100111

i) Sign magnitude form of –39 = 10100111

ii) 1's complement of –39 = 11011000

iii) 2's complement of –39 = 11011001

5.7 2's Complement Arithmetic

1) **Addition**

i. Both positive: Add + 83 and + 16

+ 83	→	0 1 0 1 0 0 1 1
+ 16	→	0 0 0 1 0 0 0 0
		0 1 1 0 0 0 1 1

ii. Positive and smaller negative: Add + 125 and –68

+ 125	→	0 1 1 1 1 1 0 1	⟶	0 1 1 1 1 1 0 1
– 68	→	– 0 1 0 0 0 1 0 0	2's complement ⟶	+ 1 0 1 1 1 1 0 0
57				(1) 0 0 1 1 1 0 0 1

For 8-bit arithmetic, discard the end carry, the result is a positive number.

Result = 00111001 = +57

iii. Positive smaller and larger negative: Add +37 and –115

+37	→	0 0 1 0 0 1 0 1	
+(–115)	→	1 0 0 0 1 1 0 1	**(2's complement)**
–(78)		0 1 0 1 1 0 0 1 0	i.e., sum is negative

So, the result is in 2's complement form.

Take the 2's complement of it and put a –ve sign infront of it

$$10110010 \xrightarrow{\text{2's complement}} 01001101 + 1$$
$$= 01001110$$
$$= -78$$

iv. Both negative: Add – 43 and –78

$$\begin{array}{rcl} (-43) & \rightarrow & 11010101 \\ +(-78) & \rightarrow & 10110010 \\ \hline -(121) & & 110000111 \end{array}$$

Sum is negative, so discard end carry. Take the 2's complement of remaining values and put minus sign in front to get the final result.

$$10000111 \xrightarrow{\text{2's complement}} 01111000 + 1 = 01111001 = -121$$

2) Subtraction

It is similar to addition.

5.8 Arithmetic Building Blocks

1) Half Adder (HA)

Half adder is a combinational logic circuit with two inputs and two outputs. It performs the addition of two 1-bit numbers and produces the output sum and carry.

Two inputs = A, B

Two outputs = Sum, Carry.

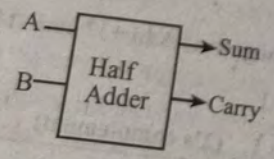
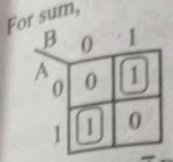


Fig.: Block diagram

Truth table

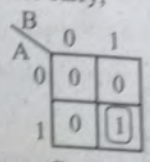
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

For sum,

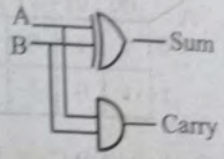


$\text{Sum} = \bar{A}B + A\bar{B}$
$= A \oplus B$

For carry,



$\text{Carry} = AB$



Drawback of half adder:

$A \rightarrow A_1 A_0$

$B \rightarrow B_1 B_0$

$$\begin{array}{r} A_1 \; A_0 \\ +\, B_1 \; B_0 \\ +\, C_0 \\ \hline C_1 \; S_1 \; S_0 \end{array}$$

To perform the addition of three bits is not possible by using a half adder circuit.

2) Full Adder (FA)

Full adder circuit overcomes the drawback of a half adder circuit. A combinational circuit that performs the addition of three bits (two significant bits and a previous carry) is called a full adder. The name of the circuit stems from the fact that two half adders can be employed to implement a full adder.

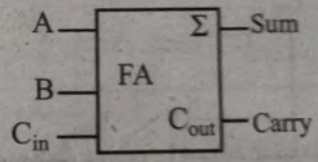


Fig.: Block diagram of a full adder

Truth table

Input			Output	
A	B	C_{in}	Sum	Carry
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

For sum,

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

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

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

$$= \bar{A}\,(B \oplus C_{in}) + A(\overline{B \oplus C})$$

$$= \bar{A}\,X + A\bar{X} \quad \text{where } X = B \oplus C_{in}$$

$$= A \oplus X$$

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

$$= A \oplus B \oplus C_{in}$$

For carry,

A \ BC cin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$\text{Carry} = BC_{in} + AC_{in} + AB$$

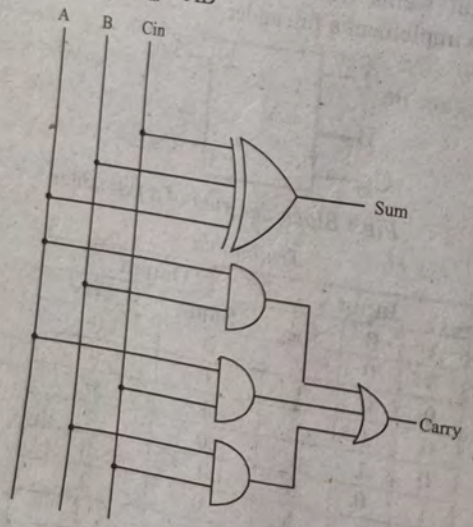


Fig.: Logic diagram of a full adder.

Construction of full adder using half adder:

Half Adder: Sum = $A \oplus B$, Carry = AB

Full Adder: Sum = $A \oplus B \oplus C_{in}$, Carry = $AB + AC_{in} + BC_{in}$

From K-map,

$$\text{Carry} = \bar{A}BC_{in} + A\bar{B}C_{in} + ABC_{in} + AB\bar{C}_{in}$$

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

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

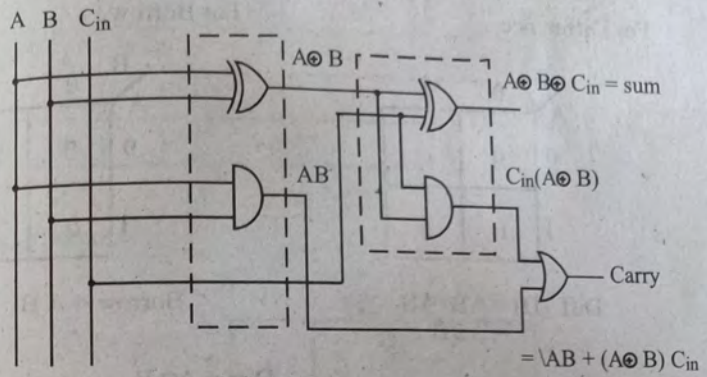


Fig.: Logic diagram of a full adder using half adder.

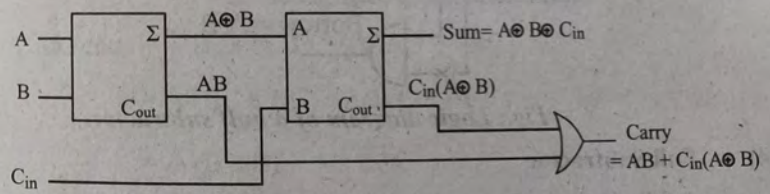
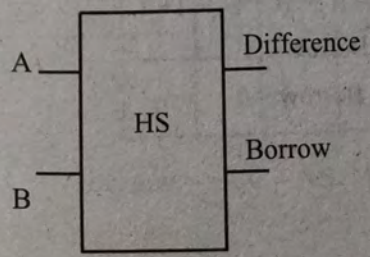


Fig.: Block diagram of a full adder using half adder.

3) Half Subtractor (HS)

A half subtractor is a combinational circuit that subtracts two bits and produce their difference. It also has an output to specify if a 1 has been borrowed.



Input		Output	
A	B	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

For Difference,

A \ B	0	1
0	0	1
1	1	0

Diff. (D) = $\overline{A}B + A\overline{B}$
= $A \oplus B$

For Borrow,

A \ B	0	1
0	0	1
1	0	0

Borrow = $\overline{A}\,B$

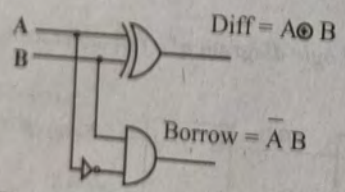


Fig.: Logic diagram of a half subtractor.

4) Full Subtractor

A full subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage. This circuit has three inputs and two outputs.

Diff = A – B – C = A – (B + C)

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

If A < (B + C), Borrow = 1

If A > (B + C), Borrow = 0

Truth table

Input			Output	
A	B	C	Difference	Borrow
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

For difference,

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Difference = $\overline{A}\,\overline{B}C + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + ABC$

= $\overline{A}\,(\overline{B}C + B\overline{C}) + A(\overline{B}\,\overline{C} + BC)$

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

= $A \oplus (B \oplus C)$

= $A \oplus B \oplus C$

For borrow,

A \ BC	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Borrow = = $\overline{A}\,C + \overline{A}B + BC$

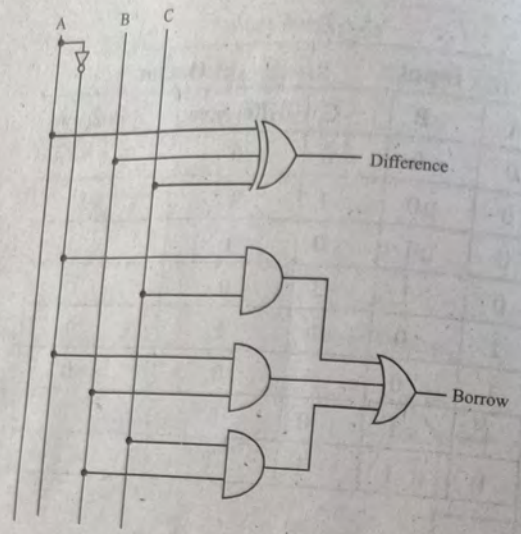


Fig.: Logic diagram of a full subtractor

Construction of full subtractor using half subtractor:

Half subtractor: Difference = A⊕B, Borrow = $\overline{A}B$

Full subtractor: Difference = A⊕B⊕C, Borrow = $\overline{A}C + \overline{A}B + BC$

From K-map,

Borrow $= \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}B\overline{C} + ABC$

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

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

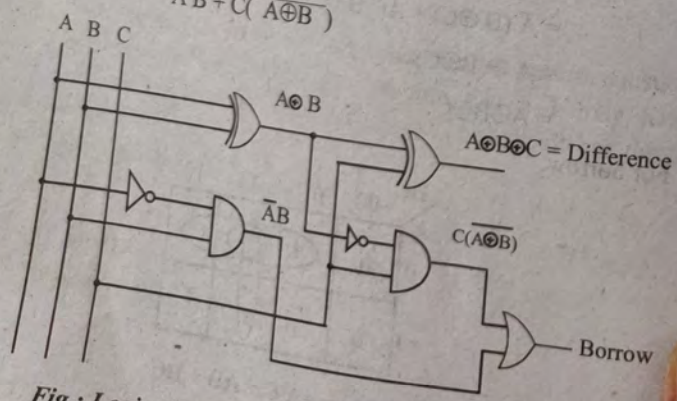


Fig.: Logic diagram of a full subtractor using half subtract

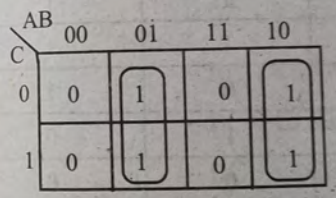
Design of half adder and half subtractor in a single circuit:

When $C = 0$, circuit performs addition

$C = 1$, circuit performs subtraction

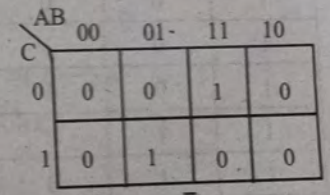
Inputs			Output1 (Y_{out1})	Output2 (Y_{out2})
C	A	B	sum/ diff	carry/ borrow
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

For Y_{out1},



$Y_{out1} = \overline{A}B + A\overline{B}$
$= A \odot B$

For Y_{out2},



$Y_{out2} = \overline{C}A\overline{B} + CA\overline{B}$
$= B(\overline{C}A + C\overline{A})$
$= B(C \odot A)$

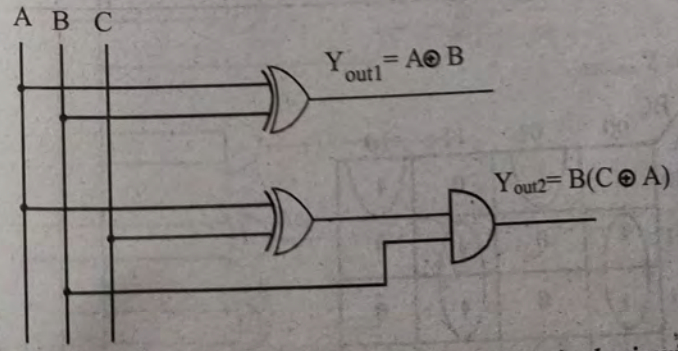


Fig.: Half adder/half subtractor in a single circuit.

Design of full adder/full subtractor in a single circuit:

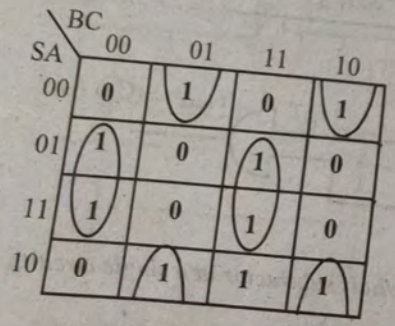
Select (S) = Control input

When S = 0, circuit performs addition

When S = 1, circuit performs subtraction

Input				Output	
S	A	B	C	$Y_{sum/diff}$	$Y_{carry/borrow}$
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	1

For $Y_{sum/diff}$,



$Y_{sum/diff} = A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + ABC + \overline{A}\,B\,\overline{C}$

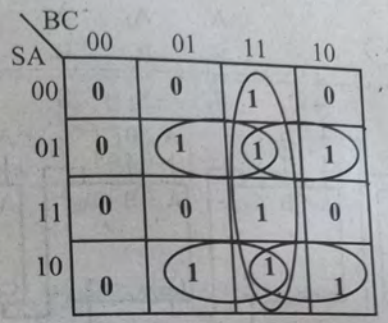
$= \overline{B}\,(A\overline{C} + \overline{A}C) + B(AC + \overline{A}\,\overline{C})$

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

$= B \oplus (A \oplus C)$

$= A \oplus B \oplus C$

For $Y_{carry/borrow}$,



$Y_{carry/borrow} = BC + \overline{S}AC + S\overline{A}B + S\overline{A}C + S\overline{A}B$

$= BC + \overline{S}AC + S\overline{A}C + \overline{S}AB + S\overline{A}B$

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

$= BC + C(A \oplus S) + B(S \oplus A)$

$= BC + (A \oplus S)(B + C)$

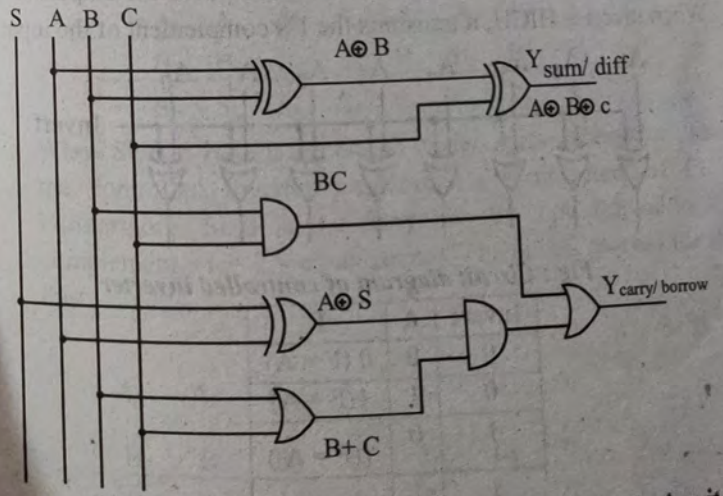


Fig.: Full adder/full subtractor in a single circuit.

5) Binary Parallel Adder (BPA)

A binary parallel adder is a digital function that produces the arithmetic sum of two binary numbers in parallel. It consists of full adders connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder. An 'n' bit parallel adder requires 'n' full adders.

Construction of 4-bit binary parallel adder:

4-bit binary parallel adder requires 4 full adders.

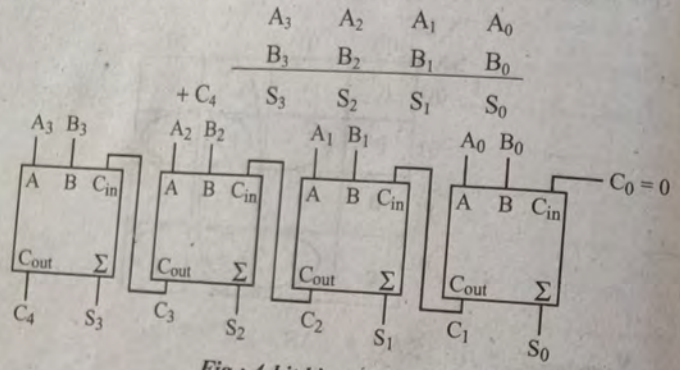


Fig.: 4-bit binary parallel adder.

Above binary parallel adder uses a ripple carry method in which the carry output of each full adder is connected to the carry input of the next higher order stage.

Controlled Inverter:

When invert is LOW, it transmits the 8-bit input to the output.

When invert is HIGH, it transmits the 1's complement of the input.

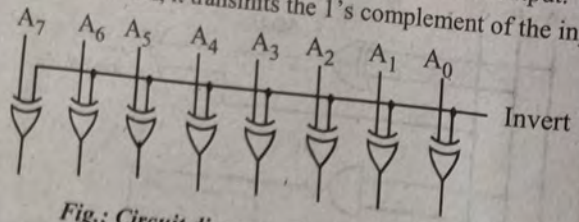


Fig.: Circuit diagram of controlled inverter

Invert	A	F
0	0	0 (F = A)
0	1	1(F = A)
1	0	1(F = $\bar{A}$)
1	1	0(F = $\bar{A}$)

Importance:

During subtraction, we first need to take 2's complement of the subtrahend then we add the complemented subtrahend to obtain the answer. With a controlled inverter, we can produce a 1's complement.

6) Binary Adder/Subtractor

A binary adder-subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.

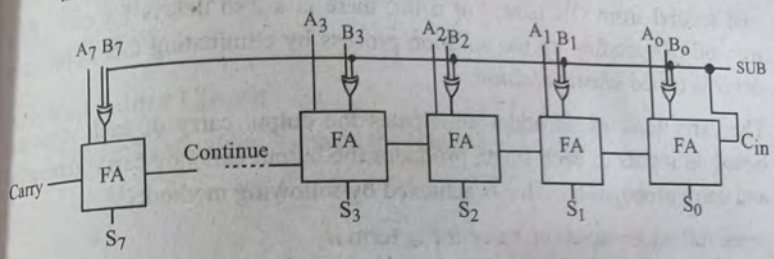


Fig.: 8-bit adder/subtractor

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	B7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
+	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

When SUB = LOW, the circuit performs the addition. The binary numbers B_7 to B_0 passes through the controlled inverter with no change. The full adders then produce the correct output sum as

0←SUB

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
+	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

When SUB = HIGH, the circuit performs the subtraction. Therefore, the controlled inverter produces 1's complement of B_0 to B_7. Furthermore, SUB is the carry into the first full adder (i.e. 1's complement +1 = 2's complement). The circuit process the data like this.

1 ←SUB

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
+	$\bar{B}_7$	$\bar{B}_6$	$\bar{B}_5$	$\bar{B}_4$	$\bar{B}_3$	$\bar{B}_2$	$\bar{B}_1$	$\bar{B}_0$
	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

5.9 Fast Adder

Fast adder is also called parallel adder or carry look ahead adder because that is how it attains high speed in addition operation. In previous ripple carry addition method, the sum and the output carry of any stage cannot be produced until the input carry occurs; this causes the time delay in the addition process.

The minimum delay required from carry generation ($C_{out} = AB + AC_{in} + BC_{in}$) in each stage is two gate delays, one coming from AND gate (1st level) and second from OR gate. For n bit, there is a 2×n delays for carry. So a method of speeding up the addition process by eliminating this ripple carry delay is called *ahead addition.*

The carry look ahead adder anticipates the output carry of each stage and based on inputs of each stage, produces the output carry by carry generation and carry propagation. This is achieved by following method:

From full adder equation, carry for i_{th} term is

$$C_i = A_iB_i + A_iC_{i-1} + B_iC_{i-1}$$
$$= A_iB_i + C_{i-1}(A_i+B_i)$$

This can be written as

$$C_i = G_i + P_iC_{i-1} \text{ where } G_i = A_iB_i \text{ and } P_i = A_i + B_i$$

Here, G_i stands for generation of carry and P_i stands for propagation of carry in a particular stage depending on input to that stage. If $A_iB_i = 1$, then i^{th} stage will generate a carry, no matter previous stage generates it or not. And if $A_i + B_i = 1$, then this stage will propagate carry if available from previous stage to next stage. Here, G_i and P_i are obtained after one gate delay once A and B are placed.

So, starting from LSB

$C_0 = G_0 + P_0C_{-1}$ [C_{-1} will be 0 normally since initial input carry is 0]

$C_1 = G_1 + P_1C_0$

$= G_1 + P_1 (G_0 + P_0 C_{-1})$

$= G_1 + P_1G_0 + P_1P_0C_{-1}$

$C_2 = G_2 + P_2C_1$

$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_{-1})$

$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{-1}$

Similarly,

$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1 P_0C_{-1}$

These equations can be realized with multi-input AND (2nd Level) and OR gate (3rd level) in two level. Maximum delay = 1+2 = 3 gate delay for carry i.e.,

1st level (one gate delay) → to produce G_0, P_0, G_1, P_1 .....

2nd level (one gate delay) → ANDing them i.e., P_1G_0, $P_1P_0C_{-1}$

3rd level (one gate delay) → ORing them

For sum = $G_i \oplus P_i \oplus C_{i-1}$, it requires 2 gate delay for two XOR gates. Therefore, total gate delay = 3+2 = 5. But for ripple carry adder, it requires (2n+2) gate delay.

2-bit Fast Adder Circuit

For 2-bit fast adder, we need the relation of C_0 and C_1

$C_0 = G_0 + P_0C_{-1}$ and $C_1 = G_1 + P_1G_0 + P_1P_0C_{-1}$

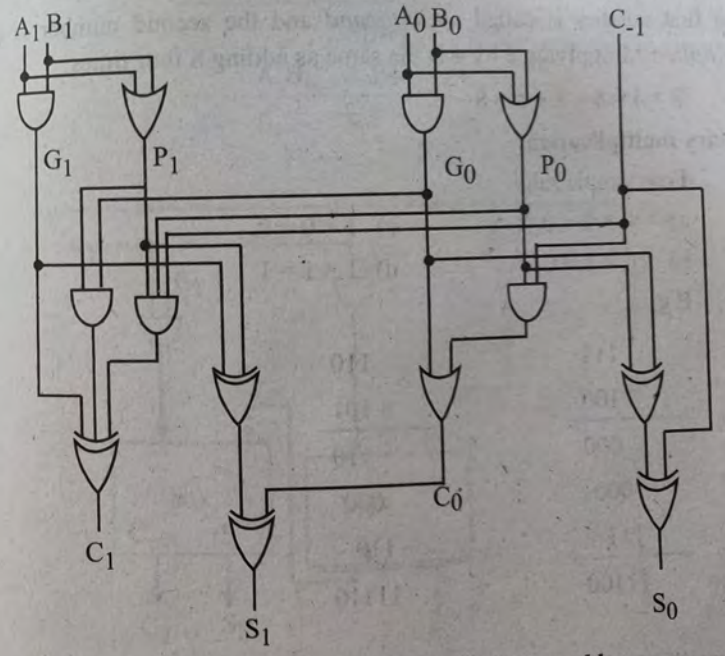


Fig.: Logic circuit for 2-bit fast adder

5.10 Arithmetic Logic Unit (ALU)

ALU is a multifunctional device that can perform both arithmetic and logic function. It is an integral part of Central Processing Unit (CPU) of a computer. It comes in various forms with wide range of functionality. Other

than normal addition, subtraction; it can perform increment, decrement operations. As logic unit, it performs usual AND, OR, NOT, EX-OR and many other complex logic functions. It also comes with PRESET and CLEAR option which makes the function outputs 1 and 0 respectively. Normally, a mode selector input (M) decides whether ALU performs a logic operation or an arithmetic operation. In each mode, different functions are choosen by appropriately activating a set of selection input.

5.11 Binary Multiplication and Division

Multiplication is done with addition instructions and division with subtraction instructions. Therefore, an adder-subtractor is all that is needed for addition, subtraction, multiplication, and division.

For example, multiplication is equivalent to repeated addition.

$8 \times 4 = ?$

The first number is called *multiplicand* and the second number is called *multiplier*. Multiplying 8 by 4 is the same as adding 8 four times:

$8 \times 4 = 8 + 8 + 8 + 8$

Binary multiplication:

Four simple rule:

a) $0 \times 0 = 0$

b) $0 \times 1 = 0$

c) $1 \times 0 = 0$

d) $1 \times 1 = 1$

E.g.,

```
   111          110
 × 100        × 101
 -----        -----
   000          110
   000          000
   111          110
 -----        -----
 11100        11110
```

Binary division:

Binary division is obtained as decimal division.

a. $0 \div 1 = 0$ b. $1 \div 1 = 1$


```
10)11010(1101          11)101001(1101.01
  -10                    -11
  ---                    ---
   10                     100
  -10                    - 11
  ---                    ---
   010                    101
  -10                     -11
  ---                     ---
   ×                      100
                          -11
                          ---
                           100
                          -11
                          ---
                            1
```

2-bit Multiplier

		A_1	A_0
		B_1	B_0
		A_1B_0	A_0B_0
	A_1B_1	A_0B_1	×
C_2	S_2	S_1	S_o

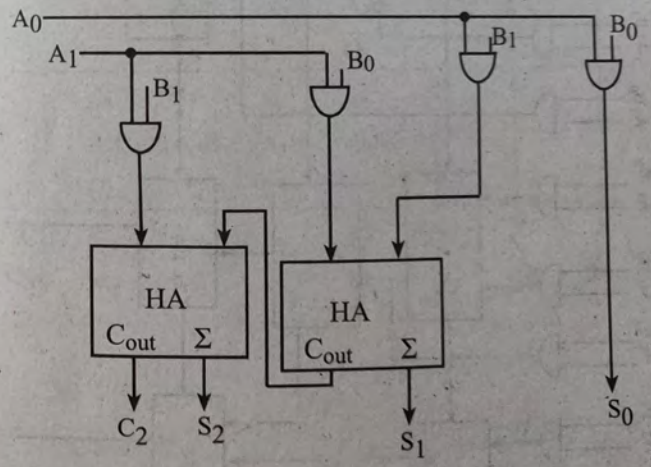


Fig.: Circuit diagram of 2-bit multiplier

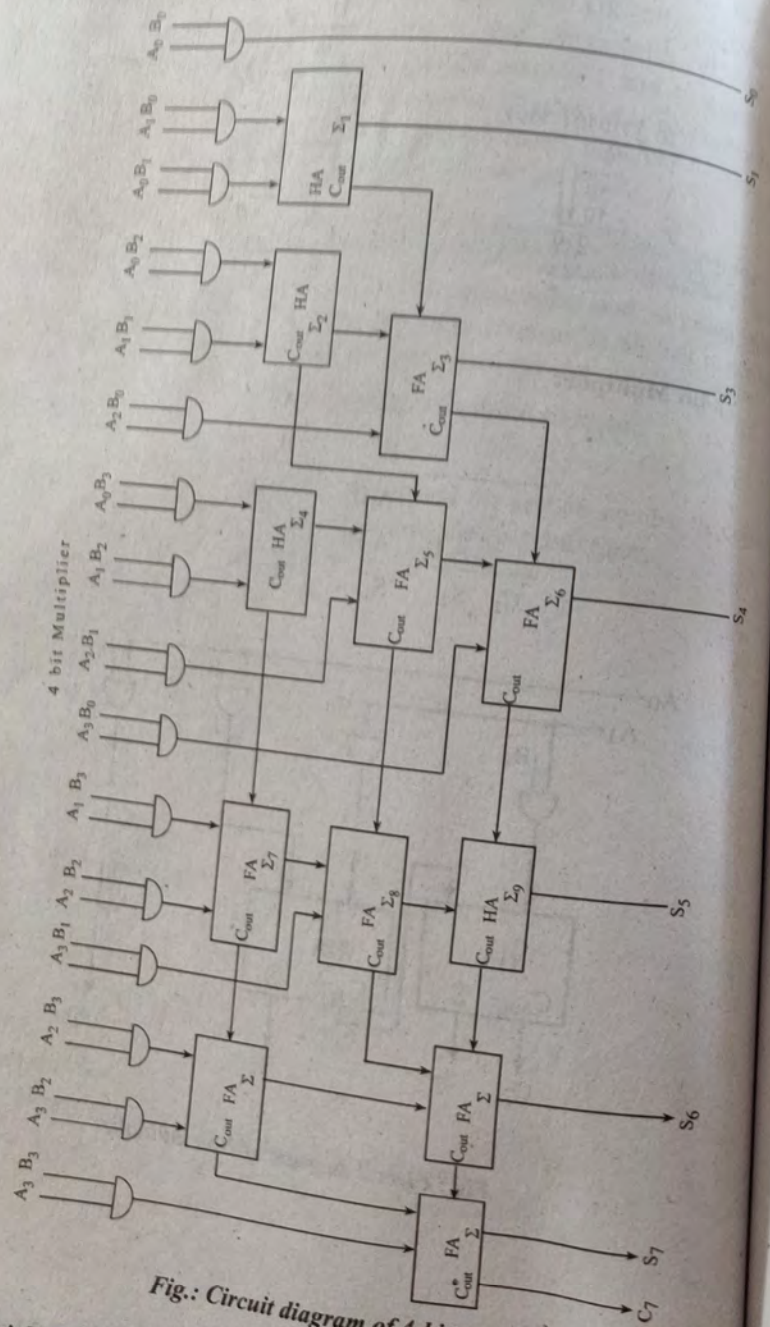


Fig.: Circuit diagram of 4-bit multiplier

			A_3	A_2	A_1	A_0	
		×	B_3	B_2	B_1	B_0	
			A_3B_0	A_2B_0	A_1B_0	A_0B_0	
		A_3B_1	A_2B_1	A_1B_1	A_0B_1		
	A_3B_2	A_2B_2	A_1B_2	A_0B_2			
A_3B_3	A_2B_3	A_1B_3	A_0B_3				
C_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Multiplication using Binary Parallel Adder

			A_3	A_2	A_1	A_0	
			B_3	B_2	B_1	B_0	
			A_3B_0	A_2B_0	A_1B_0	A_0B_0	
		A_3B_1	A_2B_1	A_0B_1	A_0B_0		
		C_2	C_1	C_0			
	C_3	S_3	S_2	S_1	S_0		
	A_3B_1	A_2B_2	A_1B_2	A_0B_2			
	C_6	C_5	C_4				
C_7	S_7	S_6	S_5	S_4			
A_3B_3	A_2B_3	A_1B_3	A_2B_2				
C_{10}	C_9	C_3					
C_{11}	S_{11}	S_{10}	S_9	S_8			
M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

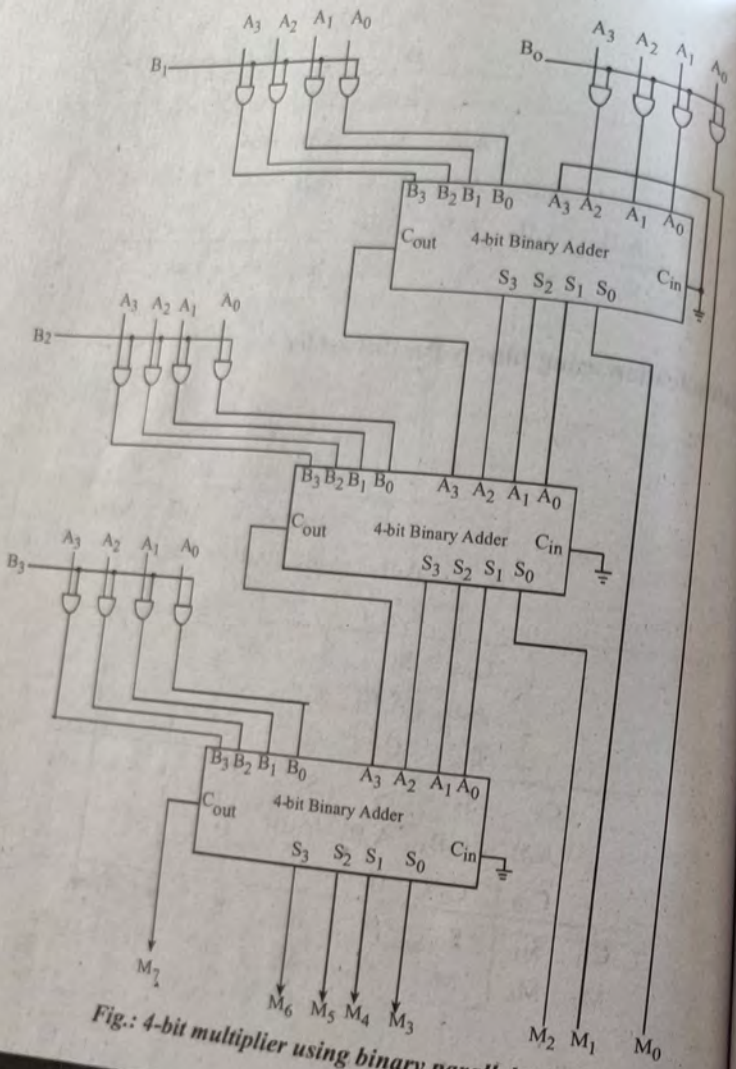


Fig.: 4-bit multiplier using binary parallel adder

MORE WORKED OUT EXAMPLES:

1. **What are the problems associated with RCA? Explain how these problems can be eliminated.** **[2068 Baishakh]**

⇒ For analysis of such mechanism let us take a worst case scenario when two four bit numbers A: 1111 and B: 0001 are added. This generates a carry in the first stage that propagates to the last stage as shown next.

Carry	:	111
	:	1111
A	:	+ 0001
B	:	10000
Sum		

This adding mechanism process resembles ripple carry addition however it reveals from the adder equation result of every stage depend on the availability of carry from previous stage. The minimum delay required for carry generation in each stage is two gate delays. One coming from AND Gate *(1st level)* and Second from OR Gate *(2nd level)*. For 32 bit serial addition there will be 32 stages working in serial. In worst case, it will require $2 \times 32 = 64$ gate delays to generate the final end carry. Through each delay is of nanosecond order. Serial addition definitely limits the speed of high speed computing devices. But parallel adder increases the speed by generating the carry in advance and there is no need to wait for the result from previous stage.

2. **Perform the subtraction with the following decimal and binary numbers using 9's and 1's complement respectively.**

a. 3570 – 2100 (using 9's complement)

b. 10010 – 10011 (using 1's complement) **[2067 Ashadh]**

⇒ a. 9's complement of 2100 = 9999 – 2100 = 7899

∴ 3570 + 7899 = 11469

Here, the end carry occurs; so add 1 to 1469

$$\begin{array}{r} 1469 \\ +1 \\ \hline 1470 \end{array}$$

∴ Ans = 1470

b. 1's compiement of 10011 = 01100

$$\therefore \begin{array}{r} 10010 \\ +01100 \\ \hline 11110 \end{array}$$

∴ Ans = 00001

3. **Subtract 8 from 6 using binary representation using 1's complement numbers whenever necessary verify your answer by comparing it to decimal subtraction.** **[2066 Bhadra]**

⇒ In binary, 6→0110

8 → 1000

1's complement of 8 → 0111

2's complement of 8 → 0111

$$\begin{array}{r} +1 \\ \hline 1000 \end{array}$$

Adding 0110 and 1000

$$\begin{array}{r} 0110 \\ +1000 \\ \hline 1110 \end{array}$$

Since, there is no carry, negative of 2's complement of 1110 will be the answer.

1's complement of 1110 → 0001

2's complement of 1110 → 0001

$$\begin{array}{r} +1 \\ \hline 0010 \end{array}$$

∴ Required answer is –0010 i.e. –2

Decimal subtraction.

$$\begin{array}{r} 6 \\ -8 \\ \hline -2 \end{array}$$

Hence, verified.

4. **Design a full subtractor. Why is it called so? Explain carry propagation that occurs in binary parallel adder and its effect on the output.** **[2066 Bhadra]**

⇒ A combinational logic circuit that performs the subtraction of three bits is called full subtraction of three bits is called full subtractor.

Hence, a full subtractor is a three input circuit.

Truth table

A	B	C	Difference (D)	Borrow (B)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Logic expression:

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

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

$= A'(B \oplus C) + A(B \oplus C)'$

$= A \oplus B \oplus C$

$B = A'B'C + A'BC' + A'BC + ABC$

$= A'(B \oplus C) + B(A + B'C')$

$= A'(B \oplus C) + A(B \oplus A')$

$= A'(B \oplus C) + BC$

Logic circuit:

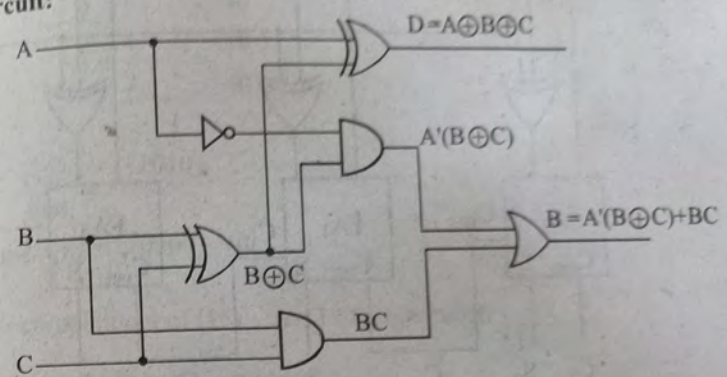


Fig.: Circuit diagram of full subtractor

Also in terms of block diagram the 3 bit full adder/subtractor is drawn below. This helps to explain the carry propagation and general overview of system work out.

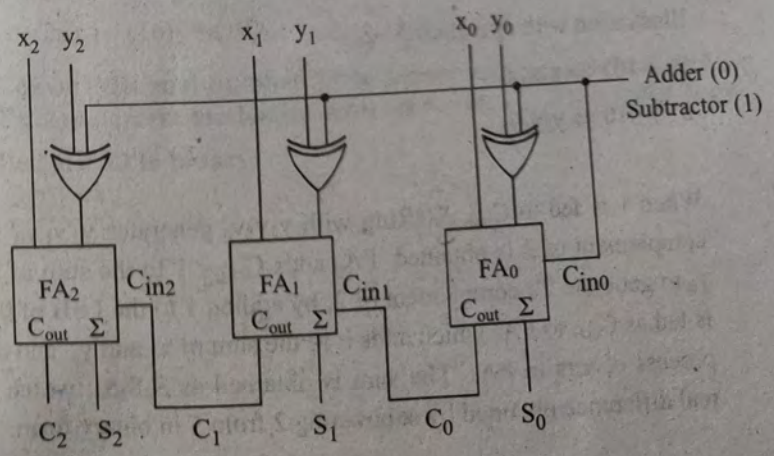


Fig.: 3-bit adder/subtractor

For the first adder, input carry is fed '0' and the full adder adds two bit numbers along with the C_{in} and outputs carry and sum. The sum will be the LSB of total sum and the output carry is fed as carry input to second adder and the cycle continuous.

5. **Design a 3-bit subtractor that operate on 2's complement numbers using full adders, and explain its operation when 2 is subtracted from 5.** **[2065 Shrawan]**

⇒

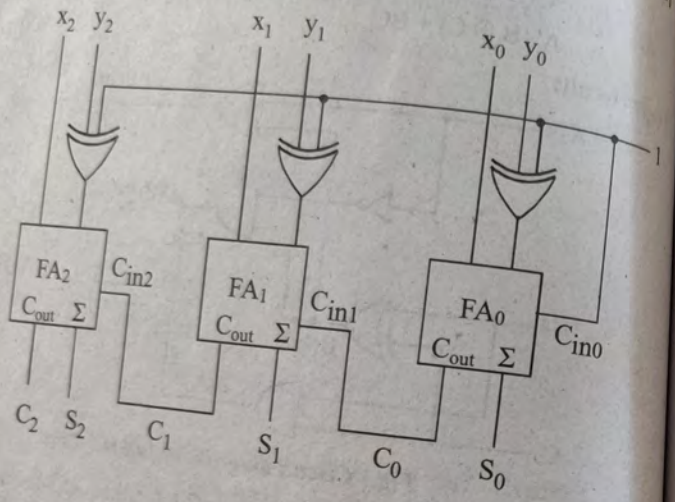


Fig.: 3-bit subtractor

Illustration with operation 5 –2

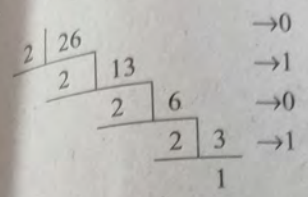
$5 \rightarrow 101 \Rightarrow x_2x_1x_0$

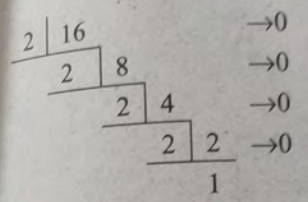
$2 \rightarrow 010 \Rightarrow y_2y_1y_0$

When 1 is fed to C_{in0}, XORing with $y_2y_1y_0$ generates $y_2'y_1'y_0'$ i.e. 1's complement of 2 is obtained. FA_0 adds $C_{in0} = 1$ to the sum of x_0 and y_0 to generate 2's complement of 2, by adding 1 to the LSB of 010. C_0 is fed as C_{in1}, to FA_1 which adds it to the sum of x_1 and y_1' and similar process occurs in FA_2. The sum is obtained as $S_2S_1S_0$, which is the real difference obtained by subtracting 2 from 5 in binary form.

6. **Subtract $(16)_{10}$ from $(26)_{10}$ using 2's complement binary method.** **[2064 Falgun]**

⇒





$\therefore (26)_{10} = (11010)_2$

And

$(16)_{10} = (10000)_2$

2's complement of $(16)_{10}$ i.e. $(10000)_2 = 10000 \left\{ \begin{matrix} 01111 \\ +1 \\ \hline 10000 \end{matrix} \right.$

Adding $(11010)_2$ and $(10000)_2$ we get

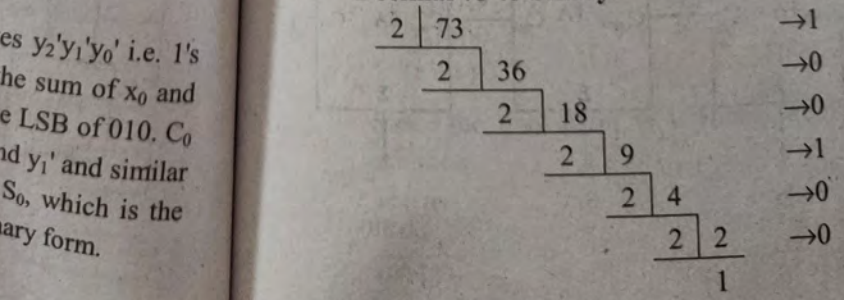
$$\begin{array}{r} 11010 \\ +10000 \\ \hline 101010 \end{array}$$

End carry → neglected

$\therefore (26)_{16} - (16)_2 = (10)_2$

7. **Convert decimal number 73 to binary and hexadecimal and use 2's complement method to subtract 5 – 16.** **[2064 Jestha]**

⇒ **Decimal 73 to binary**



$\therefore (73)_{10} = (1001001)_2$

Decimal 73 to hexadecimal

We know, $(73)_{10} = (1001001)_2$

$= (0100\ 1001)_2 = (49)_{16}$

Now, to subtract 16 from 5

$(16)_{10} = (10000)_2$

1's complement of $(16)_{10} = (01111)_2$

Calculating 2's complement of 16,

As, 1's complement of $(16)_{10} + (1)_2$

$$\begin{array}{r} 01111 \\ +1 \\ \hline (10000)_2 \end{array}$$

Add $(5)_{10}$ with $(10000)_2$ i.e.

$$\begin{array}{r} 00101 \\ +10000 \\ \hline (10101)_2 \end{array}$$

Since there is no end carry, take 2's complement of above result and introduce a '–' sign

$\therefore$ 2's complement of $(10101)_2$ = 1's complement $(10101)_2 + (1)_2$

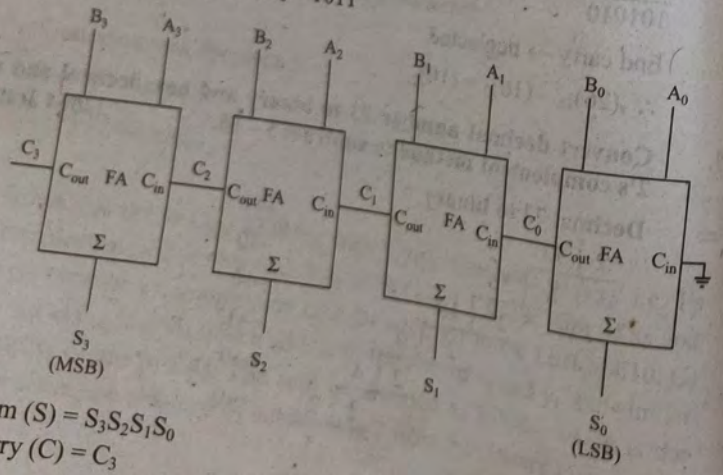
$= (01010) + (1)_2 = (01011)_2$

$\therefore (5)_{10} - (16)_{10} = (-01011)_2$

8. How do you add 1101 and 1011 by using full adders and half adder? Show by block diagram. **[2064 Jestha]**

⇒ Here, let $A = A_3A_2A_1A_0 = 1101$

$B = B_3B_2B_1B_0 = 1011$



Sum $(S) = S_3S_2S_1S_0$

Carry $(C) = C_3$

9. Design a combinational circuit that multiplies 2-bit numbers A_0, A_1 and B_0, B_1 to produce 4-bit product $C_0\ C_1\ C_2\ C_3$.

⇒

$$\begin{array}{cccc} & & A_1 & A_0 \\ & & B_1 & B_0 \\ \hline & & A_1B_0 & A_0B_0 \\ & B_1A_1 & B_1A_0 & \times \\ \hline C_0 & C_1 & C_2 & C_3 \end{array}$$

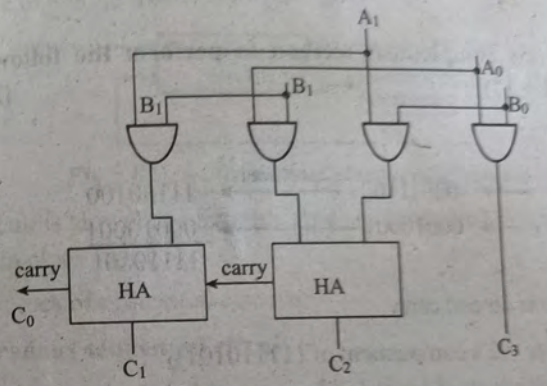


Fig.: 2-bit multiplier

10. Perform the following addition using 2's complement -5+12. **[2071 Chaitra]**

⇒ Binary form of 5 is $(101)_2 \rightarrow (0101)_2$

Binary form of 12 is $(1100)_2$

1's complement of 5 is 1010

$$\therefore \begin{array}{r} 1010 \\ +1100 \\ \hline (1)0110 \end{array}$$

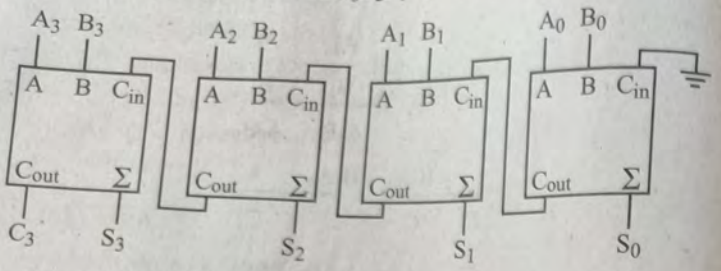
Since the final carry is 1. Hence, adding the carry to the result of addition.

$$\begin{array}{r} 0010 \\ +1 \\ \hline (0111)_2 \end{array}$$

Thus, $(0111)_2$ is the final result. $(7)_{10}$ is the result of $-5 + 12$.

11. Draw the circuit to add following bits: 1011 and 1100. [2074 Chaitra]

⇒ Let $A_3A_2A_1A_0 = 1011$ and $B_3B_2B_1B_0 = 1100$



12. Use 2's complement method to perform the following addition (−28 + 17). [2074 Ashwin]

⇒

$-28 \longrightarrow -00011100 \xrightarrow{\text{2's complement}} +11100100$

$+17 \longrightarrow 00010001 \longrightarrow +00010001$

11110101

There is no end carry.

Answer = 2's complement of $(11110101)_2$

$= -(0001011)_2$

$= -(11)_{10}$

Chapter – 6

FLIPFLOPS

6.1 Sequential Circuit

The logic circuits whose outputs at any instant of time depend not only on the present inputs but also on past outputs are called sequential circuits. A sequential circuit consists of a combinational circuit to which storage elements are connected to form a feedback path. The storage elements are devices capable of storing binary information.

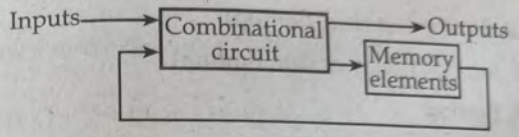


Fig.: Block diagram of a sequential circuit

Sequential circuit is slower in operation than combinational circuit. It may or may not contain clock input.

There are two types of sequential circuits:

i. Synchronous sequential circuit

It is a system whose behavior can be defined from the knowledge of its signals at discrete instant of time (i.e., by the clock signal parallelly driven by one clock signal).

ii. Asynchronous sequential circuit

It is a system whose behavior depends in the order in which its input signals change and can be affected at any instant of time (one's output is given to clock of another).

6.2 Flipflop

A flipflop is a sequential circuit which is popularly known as a basic digital memory circuit. A flipflop stores 1 bit, therefore, it is called as 1-bit memory cell. It has two stable states: logic 1 and logic 0. It can flip from one state to another and then flop back, so it is called a flipflop and also known as a bistable multivibrator. The outputs of circuit (Q and $\bar{Q}$) will always be complementary. This means that if Q.= 0, then $\bar{Q} = 1$ and vice-versa. They will never be equal ; $Q = \bar{Q} = 0$ or 1 is an invalid state. If $Q = 1$, $\bar{Q} = 0$, it is

called 1 state or SET state. If $Q = 0, \bar{Q} = 1$, it is called 0 state or RESET state. If the circuit is in reset state, then it continues to be in reset state and if it is in set state, then it continues to be in set state. This characteristic of the circuit illustrates that it can store 1 bit of digital information.

Application of flipflops include:

1. As a memory element
2. In various types of registers
3. In counters/timers
4. As a delay element

The different types of flipflop are explained below:

i. SR flipflop (set reset flipflop)
ii. D flip flop (delay or data flipflop)
iii. JK flipflop
iv. T flipflop (toggle flipflop)
v. Master slave flipflop

6.3 SR Flipflop

SR flipflop has two inputs namely SET (S) and RESET (R) and two outputs Q and $\bar{Q}$. It can be implemented using NOR and NAND gates. The flipflop is often called a *latch* since it will hold, or latch, in either stable state.

NOR-based SR latch

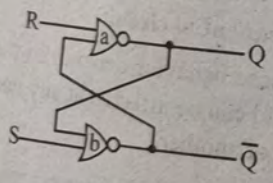


Fig.: NOR latch

Truth Table

S	R	Q	$\bar{Q}$	Remarks
1	0	1	0	Set
0	0	1	0	No change
0	1	0	1	Reset
0	0	0	1	No change
1	1	?	?	Invalid/ Forbidden

Case I: S = 1, R = 0

As S = 1, output of NOR gate b i.e., $\bar{Q} = 0$. This makes both inputs of NOR gate a at 0, therefore Q = 1. Now, both inputs of NOR gate b are 1. Hence, for S = 1, R = 0, Q = 1 and $\bar{Q} = 0$ (Set condition).

Case II: S = 0, R = 1

As R = 1, the output of NOR gate a i.e., Q = 0. This makes both inputs of NOR gate b are 0, therefore $\bar{Q} = 1$. Now, both inputs of NOR gate a are 1. Therefore, for S = 0, R = 1, Q = 0, $\bar{Q} = 1$ (Reset condition).

Case III: S = 0, R = 0

We know that the output of NOR gate a i.e., $Q = \overline{R + \bar{Q}}$

As R = 0, $Q = \overline{0 + \bar{Q}} = \bar{\bar{Q}} = Q$

Again the output of NOR gate b i.e., $\bar{Q} = \overline{S + Q}$

As S = 0, $\bar{Q} = \overline{0 + Q} = \bar{Q}$

Therefore, for S = 0, R = 0, Q and $\bar{Q}$ do not change their state.

Case IV: S = 1, R = 1

If S = 1, R = 1 then the outputs Q and $\bar{Q}$ both are forced to 0. Actually this is indeterminate state which must be avoided (since $Q = \bar{Q}$ which violates the basic requirements of flip–flop i.e., Q and $\bar{Q}$ must be complement of each other).

NAND-based SR latch

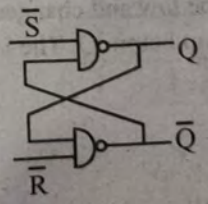


Fig.: NAND latch

Truth Table

$\bar{S}$	$\bar{R}$	Q_{n+1}	Remarks
0	1	1	Set
1	1	1	No change
1	0	0	Reset
1	1	0	No change
0	0	?	Forbidden

Case I: $\bar{S} = 0, \bar{R} = 0$

When any input of NAND gate is 0, the output is forced to 1. Here S = R = 0, therefore, Q and $\bar{Q}$ are forced to be equal to 1 which is indeterminate and must be avoided.

Case II: S = 0, R = 1 (i.e., $\bar{S} = 1, \bar{R} = 0$)

As $\bar{R} = 0$, output of NAND gate b i.e., $\bar{Q} = 1$. This makes both inputs of NAND gate a at 1, therefore, Q= 0. Hence, for S = 0, R = 1, the output Q = 0 and $\bar{Q} = 1$ (Reset condition).

Case III : S = 1, R = 0 (i.e., $\bar{S} = 0, \bar{R} = 1$)

As $\bar{S} = 0$, output of NAND gate a i.e., Q = 1. This makes both inputs of NAND gate b at 1, therefore, $\bar{Q} = 0$. Hence, for S = 1, R = 0, the output Q = 1 and $\bar{Q} = 0$ (Set condition).

Case IV: S = 0, R = 0, (i.e., $\bar{S} = 1, \bar{R} = 1$)

The output of NAND gate a, $Q = \overline{\bar{S}.\bar{Q}} = \overline{1.\bar{Q}} = \bar{\bar{Q}} = Q$

The output of NAND gate b, $\bar{Q} = \overline{\bar{R}.Q} = \overline{1.Q} = \bar{Q}$

Thus, there is no change in the output if $\bar{S} = \bar{R} = 1$.

6.4 Gated Flipflop

1) Gated RS Flipflop

The addition of two AND gates at the R and S inputs will result in a flipflop that can be enabled or disabled. When the ENABLE input is HIGH, information at the R and S will be transmitted directly to the outputs. The latch is said to be enabled. When the ENABLE input is LOW, the AND gate outputs must both be low and changes in neither R nor S will have any effect on the flipflop output Q. The latch is said to be disabled.

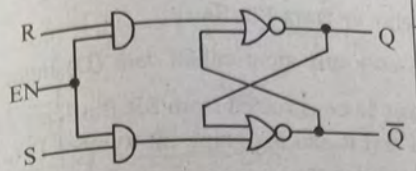


Fig.: Gated RS flipflop (NOR)

Truth Table

EN	S	R	Q_{n+1}
0	×	×	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Invalid

When two inputs signal R and S are applied to the circuit, there is a time delay between these two signals, which may lead to a wrong output result. In order to overcome this problem of unequal propagation delay time of input signals, the gated flipflop is used.

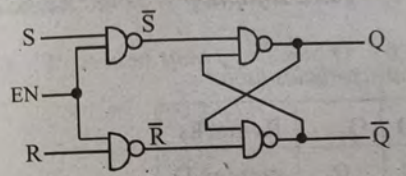


Fig.: Gated RS flipflop (NAND)

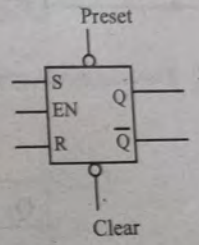
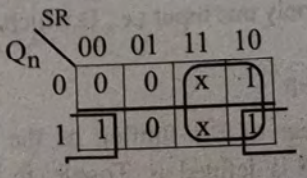


Fig.: Logic symbol

The characteristic table of gated RS flipflop is given below.

Q_n	S	R	Q_{n+1}	Remarks
0	0	0	0	No change
0	0	1	0	Reset
0	1	0	1	Set
0	1	1	×	Invalid
1	0	0	1	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	×	Invalid



$Q_n+1 = S + Q_n\bar{R}$

This is the characteristic equation.

2) Gated D Flipflop (Delay or Data Flipflop)

The data flipflop has only one input called data (D) input and two outputs Q and $\bar{Q}$. It can be constructed from SR flipflop by inserting an inverter between S and R and assigning the symbol D to S input. The output Q is same as D input.

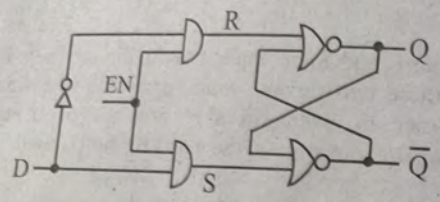


Fig.: Logic diagram of a gated D flipflop

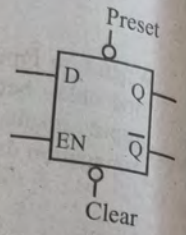


Fig.: Logic symbol

Characteristics table:

Q_n	D	Q_{n+1}	Remarks
0	0	0	same as D
0	1	1	same as D
1	0	0	same as D
1	1	1	same as D

Q_n \ D_o	0	1
0	0	1
1	0	1

$Q_{n+1} = D$

Advantages of D flipflop over RS flipflop:

In some events, both inputs become high in RS flipflop which is undesirable condition. This drawback of RS flipflop is overcome in D flipflop. There is only one input i.e., D which drive the flipflop.

3) Gated JK Flipflop

It is the refinement of RS flipflop as the indeterminate state (or invalid state) of RS is defined as 'Toggle' in JK. A JK flipflop can be

constructed using two cross coupled NOR gates and two AND gates as shown in below figure.

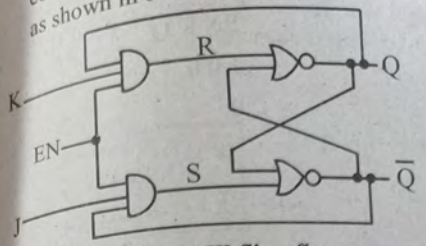


Fig.: JK flip - flop

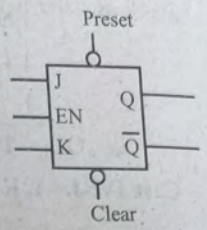


Fig.: Logic Symbol

Case I: J = 0, K = 0

If J = 0, K = 0, the output of two AND gates i.e., R and S are equal to 0. We know that if R = S = 0, the output Q and $\bar{Q}$ i.e., $Q_{n+1} = Q_n$ and $\overline{Q_{n+1}} = \overline{Q_n}$

Case II: J = 0, K = 1

(i) Provided that Q = 1 and $\bar{Q}$ = 0

S = EN.J. $\bar{Q}$ = 1.0.0 = 0

R = EN.K.Q = 1.1.1 = 1

i.e., $Q_{n+1} = 0$

(ii) Provided that, Q = 0 and $\bar{Q}$ = 1

S = EN.J. $\bar{Q}$ = 1.0.1 = 0

R = EN.K.Q = 1.1.0 = 0

i.e., remain in reset state, $Q_{n+1} = 0$

Case II: J = 1, K = 0

(i) Provided that Q = 1 and $\bar{Q}$ = 0

S = EN.J. $\bar{Q}$ = 1.1.0 = 0

R = EN.K.Q = 1.0.1 = 0

i.e., $Q_{n+1} = 1$

(ii) Provided that $Q = 0$ and $\bar{Q} = 1$

$S = EN.J.\bar{Q}$ $\quad R = EN.K.\bar{Q}$

$= 1.1.1$ $\quad = 1.0.0$

$= 1$ $\quad = 0$

i.e., $Q_{n+1} = 1$

Case IV: J = 1, K = 1

(i) Provided that $Q = 1$ and $\bar{Q} = 0$

$S = EN.J.\bar{Q}$ $\quad R = EN.K.Q$

$= 1.1.0$ $\quad = 1.1.1$

$= 0$ $\quad = 1$

i.e., $Q_{n+1} = 0$

(ii) Provided that $Q = 0$ and $\bar{Q} = 1$

$S = EN.J.\bar{Q}$ $\quad R = EN.K.Q$

$= 1.1.1$ $\quad = 1.1.0$

$= 1$ $\quad = 0$

i.e., $Qn+1 = 1$

The outputs are inverted i.e., toggle from reset to set [$Q = 0$ to $Q = 1$]

Characteristics table:

Q_n	J	K	Q_{n+1}	Remarks
0	0	0	0	No change
0	0	1	0	Reset
0	1	0	1	Set
0	1	1	1	Toggle
1	0	0	1	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	0	Toggle

JK

Q_n \	00	01	11	10
0	0	0	1	1
1	1	0	0	1

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

Gated Toggle Flipflop (T flipflop)

4) It is made by shorting the J and K input of JK flipflop to single input T. The output toggles each time for T = 1.

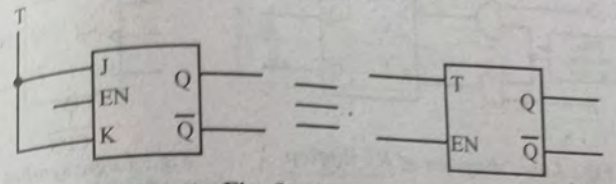


Fig: Logic symbol

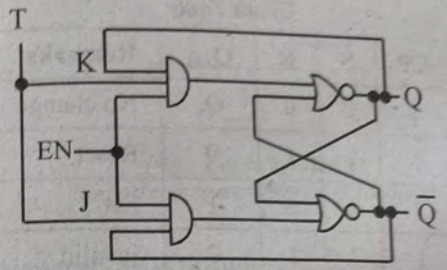


Fig.: Logic diagram of T flipflop

Characteristic table:

Q_n	T	Q_{n+1}	Remarks
0	0	0	No change
0	1	1	Toggle
1	0	1	No change
1	1	0	Toggle

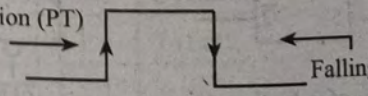
Q_n \ T	0	1
1	1	0

$Q_{n+1} = \bar{Q}_n T + Q_n \bar{T}$

$= (Q_n \oplus T)$

6.5 Edge Triggered Flipflop

Rising edge or positive transistion (PT)



Falling edge or Negative transistion (NT)

Fig.: clock

Everything is same as gated (or level triggered), only the difference is the clock pulse type which makes the flip–flop enable. Another is the symbol given for edge triggered.

1) Edge Triggered RS Flipflop

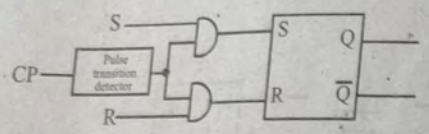


Fig.: Logic diagram of RS flipflop

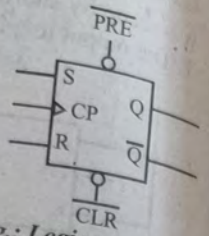


Fig.: Logic symbol of positive edge triggered RS flipflop

Truth Table

CP	S	R	Q_{n+1}	Remarks
↑	0	0	Q_n	No change
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	?	Invalid

Similarly, for negative edge triggered RS flipflop, the logic symbol is

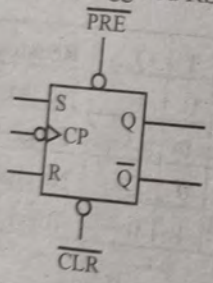


Fig.: Logic symbol for negative edge triggered RS flipflop

2) Edge Triggered D Flipflop

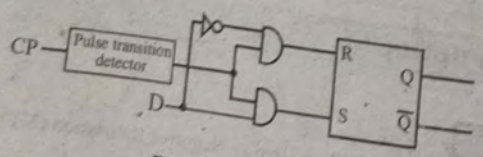


Fig.: Logic diagram of D flipflop

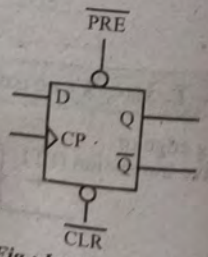


Fig.: Logic symbol of positive edge triggered D flipflop

Truth Table

CP	D	Q_{n+1}
0	×	Q_n
↑	0	0
↑	1	1

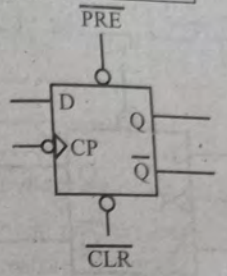


Fig.: Logic symbol of negative edge triggered D flipflop

3) Edge Triggered JK Flipflop

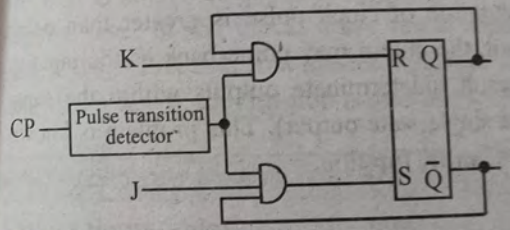


Fig.: Logic diagram of JK flipflop

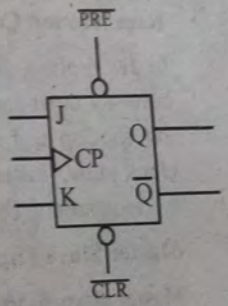


Fig.: Logic symbol of positive edge triggered JK flipflop

CP	J	K	Q_{n+1}
↑	0	0	Q_n (Last state)
↑	1	0	0 (Reset)
↑	0	1	1 (Set)
↑	1	1	$\overline{Q}$ (Toggle)

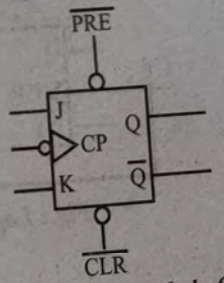


Fig.: Logic symbol of negative edge triggered JK flipflop

Asynchronous Inputs (Direct Inputs)

PRESET and CLEAR are called asynchronous inputs because they activate the flipflop independent of clock. It may be active low or high.

If $\overline{CLR}$ is LOW, then the output is reset. If $\overline{PRE}$ is LOW, then the output is set.

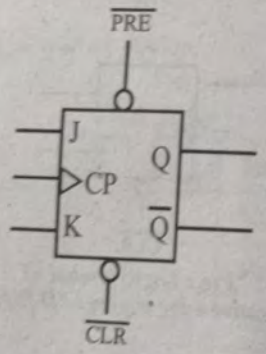


Fig.: Asynchronous inputs in JK flipflop

Race Around Condition

In JK flipflop, if the duration of clock pulse is greater than delay between input and output, the output may come back as the input to the flipflop and may result indeterminate outputs within the same clock pulse (instead of a single state output). This problem is known as "race around condition" in JK flip flop.

4) **Master Slave Flipflop**

Master slave flipflop is required to remove the race around condition. Master slave flipflop consists of two flipflops; one serve as master and another as slave. Master is positive edge triggered and slave is negative edge triggered.

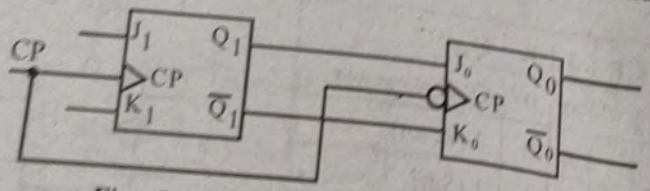


Fig.: Logic symbol of master slave JK flipflop

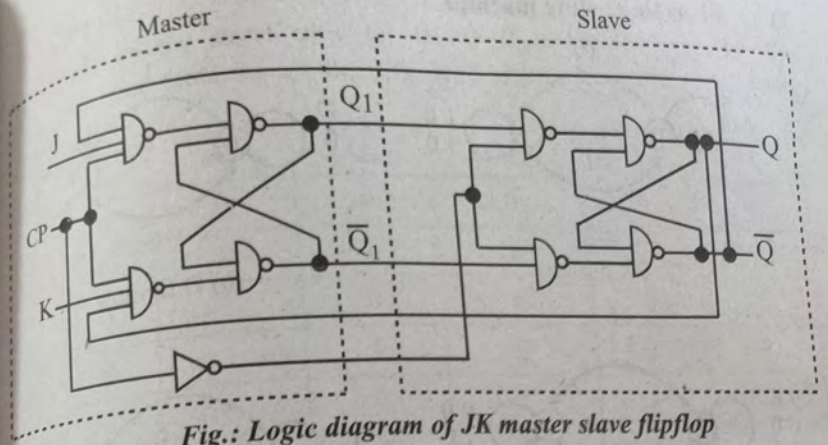


Fig.: Logic diagram of JK master slave flipflop

To begin with, the master is positive-edge triggered and the slave is negative-edge triggered. Therefore, the master responds to its J and K inputs before the slave. If J=1 and K=0, the master sets. The high Q output of the master drives the J input of the slave, so on negative edge of the clock (this makes slave positive-edge triggered), the slave sets copying the action of the master. If J = 0, K = 1, the master resets when it is positive-edge triggered.

The high $\bar{Q}$ output of the master goes to the K input of the slave. So, on negative level of the clock, slave resets. Again, the slave has copied the master. If J = 1, K = 1, first master toggles its output and later slave does the same. If J = K = 0, the flip-flop is disabled and Q remains unchanged.

6.6 Various Representation of Flipflops

Various Representation of Flipflops

i. Characteristic equation of FF

ii. FF as finite state machine

iii. FF excitation table

1) **Characteristic equation of FF**

SR : $Q_{n+1} = S + Q_n \bar{R}$

D : $Q_{n+1} = D$

JK : $Q_{n+1} = \bar{Q}_n J + Q_n \bar{K}$

T : $Q_{n+1} = \bar{Q}_n T + Q_n \bar{T}$

2) **FF as finite state machine**

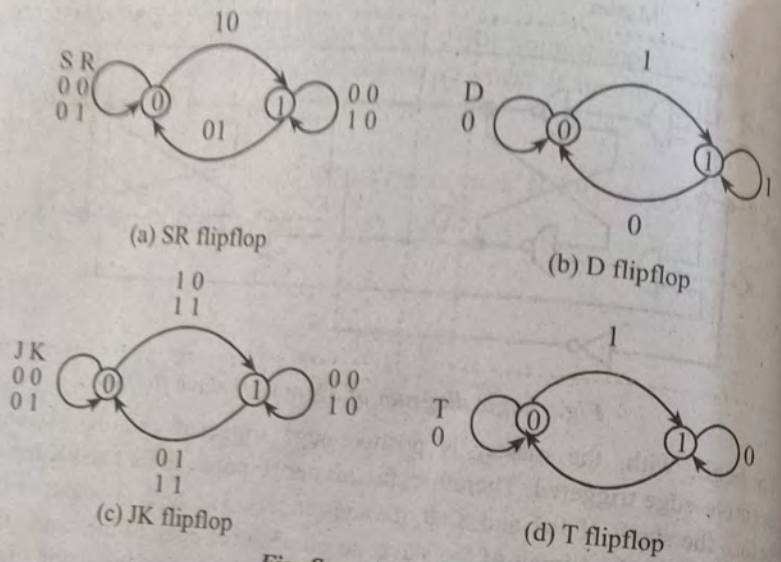


Fig: State transition diagram

3) **Flipflop excitation table**

It is very important in analysis of problem. It is just a reverse of truth table. Here, based on present output (Q_n) and next output (Q_{n+1}) the input of flipflop is generated.

Excitation table of flipflops

Transition		RS flipflop		JK flipflop		D flipflop	T flipflop
$Q_n \rightarrow Q_{n+1}$		S	R	J	K	D	T
0	0	0	×	0	×	0	0
0	1	1	0	1	×	1	1
1	0	0	1	×	1	0	1
1	1	×	0	×	0	1	0

EXAMPLE:

1. **Convert SR flipflop to JK flipflop.**

⇒ **Step 1:**

First we generate the truth table for JK flip–flop i.e., note $Q_n \rightarrow Q_{n+1}$ transition for a given combination of JK input.

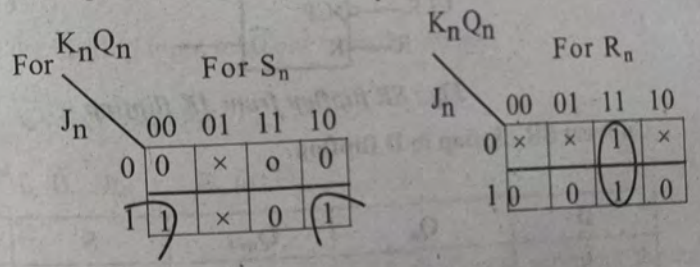
Step 2:

Next on the same table for $Q_n \rightarrow Q_{n+1}$ transition identify the excitation table for SR flip–flop such tables are known as synthesis table.

J_n	K_n	Q_n	Q_{n+1}	S_n	R_n
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	×	0
1	1	0	1	1	0
1	1	1	0	0	1

Step 3:

Write SR inputs as a function JK inputs and present state Q_n. It is done through K-map.



$S_n = J_n \overline{Q}_n$ $\quad R_n = K_n Q_n$

Step 4:

Design the circuit using SR flipflop and input JK.

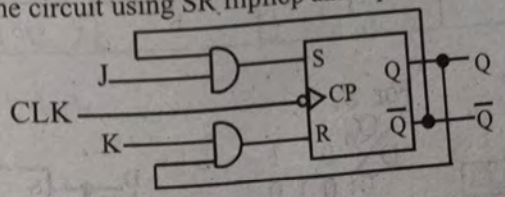


Fig.: JK flipflop from SR flipflop

2. Convert JK flipflop to SR flipflop.

⇒

S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	×	1
1	0	0	1	1	×
1	0	1	1	×	0

for, J

S \ RQn	00	01	11	10
0	0	×	×	0
1	1	×	×	×

J = S

for, K

S \ RQn	00	01	11	10
0	×	0	1	×
1	×	0	×	×

K = R

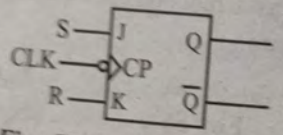


Fig.: SR flipflop from JK flipflop

3. Convert SR flipflop to D flipflop.

⇒

D	Q_n	Q_{n+1}	S	R
0	0	0	0	×
0	1	0	0	1
1	0	1	1	0
1	1	1	×	0

For R,

D \ Q_n	0	1
0	×	1
1	0	0

$R = \overline{D}$

For S,

D \ Q_n	0	1
0	0	0
1	1	×

S = D

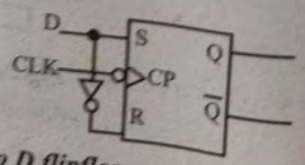


Fig.: SR flipflop to D flipflop

6.7 Analysis of a Sequential Circuit

A sequential logic circuit contains flipflops as memory elements and also contains logic gates as combinational circuits. Analysis of a circuit helps to explain its performance.

EXAMPLE:

1. **Consider the sequential circuit as shown below. It has only CLK and output 'X' is generated from flipflop outputs as shown. Analyse the circuit.**

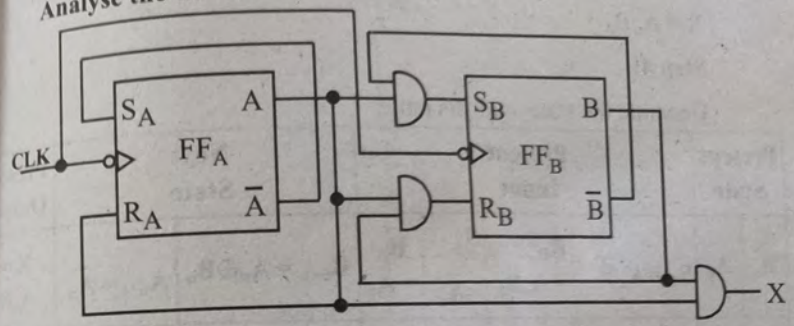


Fig.: A sequential logic circuit for analysis purpose

Step 1:

Note the flipflop input relations from the circuit.

$S_A = \overline{A}_n$, $R_A = A_n$ for FF_A

$S_B = A_n \overline{B}_n$, $R_B = A_n B_n$ for FF_B

Step 2:

Use the characteristic equation of SR flipflop to represent the next output i.e., $Q_{n+1} = S + Q_n \overline{R}$.

For FF_A,

$$A_{n+1} = S_A + A_n \overline{R}_A$$

$$= \overline{A}_n + A_n \overline{A}_n$$

$$= \overline{A}_n$$

For FF_B,

$$B_{n+1} = S_B + B_n \overline{R}_B$$

$$= A_n \overline{B}_n + B_n \overline{A_n B_n}$$

$= A_n \bar{B}_n + B_n [\bar{A}_n + \bar{B}_n]$

$= A_n \bar{B}_n + \bar{A}_n B_n + B_n \bar{B}_n$

$= A_n \bar{B}_n + \bar{A}_n B_n$

$= A_n \oplus B_n$

Step 3:

Write the relation for output.

$X = A_n B_n$

Step 4:

Generate the state analysis table.

Present State		Present Input				Next State		Present Output
B_n	A_n	$S_B = A_n \bar{B}_n$	$R_B = A_n B_n$	$S_A = \bar{A}_n$	$R_A = A_n$	$B_{n+1} = A_n \oplus B_n$	$A_{n+1} = \bar{A}_n$	$X = A_n B_n$
0	0	0	0	1	0	0	1	0
0	1	1	0	0	1	1	0	0
1	0	0	0	1	0	1	1	0
1	1	0	1	0	1	0	0	1
0	0	0	0	1	0	0	1	0
0	1	....	....	....	....	....	....	Repeat

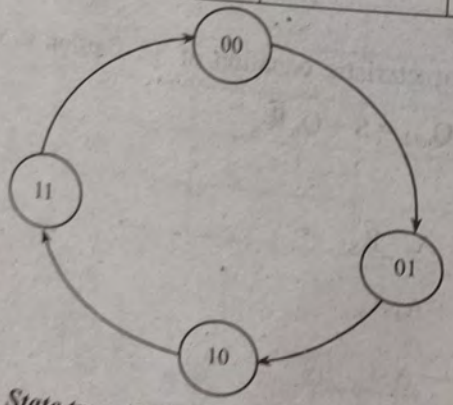


Fig.: State transition diagram of the sequential circuit

6.8 Switch Contact Bounce Circuits

Contact bounce (also called a chatter) as a common problem with mechanical switches and relays. When the contacts strikes together, their momentum and elasticity act together to cause bounce. The result is a rapidly pulsed electric volt instead of a clean transition from 0 V to full voltage. The effect is usually an important in power circuits but causes problems in digital logical circuits that respond fast enough to misinterpret the ON, OFF pulses as a data stream..

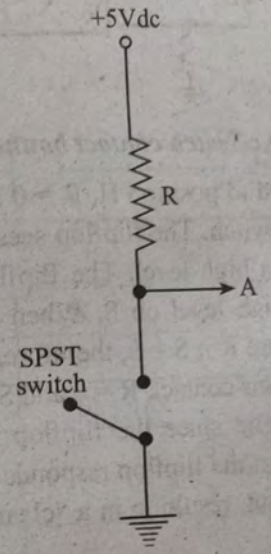


Fig.: A single-pole-single-throw (SPST) switch for illustrating contact bounce

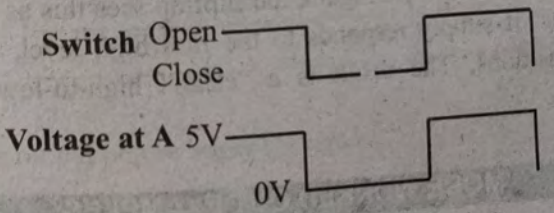


Fig.: Ideal voltage at A

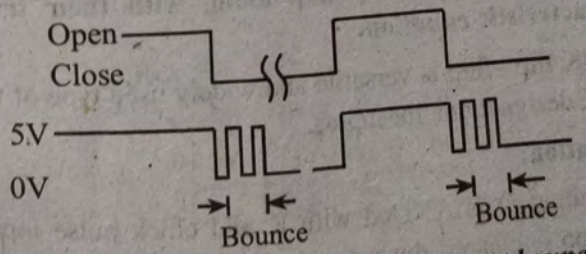


Fig.: Voltage at A showing contact bounce

When the switch is open, the voltage at point A is +5 V dc. When the switch is closed, the voltage at A must be 0 V ideally. But there occurs bounce which results the voltage level as shown in figure. This bounce problem can be eliminated with a simple RS latch debounce circuit.

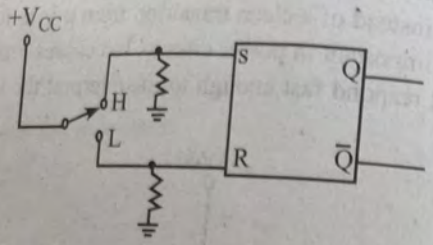


Fig.: Switch contact bounce eliminator

When the switch is moved to position H, R = 0 and S = 1. Bouncing occurs at the S input due to the switch. The flipflop sees this as a series of high and low inputs, settling with a high level. The flipflop will immediately be set with Q = 1 at the first high level on S. When the switch bounces, losing contact, the input signals are R = S = 0, therefore, the flipflop remains set (Q =1). When the switch regains contact, R = 0 and S = 1; this causes an attempt to again set the flipflop. But since the flipflop is already set, no changes occur at Q. The result is that the flipflop responds to the first, and only to the first, high level at its S input, resulting in a "clean" low-to-high signal at its output (Q).

When the switch is moved to position L, S = 0 and R =1. Bouncing occurs at the R input due to the switch. Again, the flipflop sees this as a series of high and low inputs. It simply responds to the first high level, and ignores all following transitions. The result is a "clean" high-to-low signal at the flipflop output.

MORE WORKED OUT EXAMPLES:

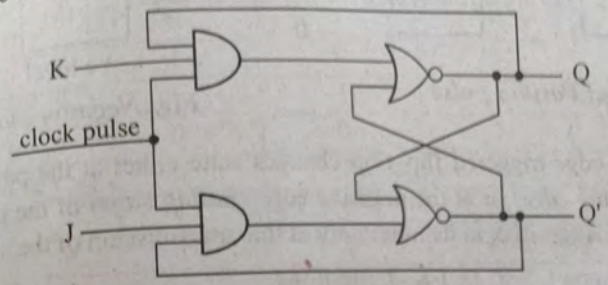
1. Explain about JK flip–flop along with their truth table and characteristic equation. **[2067 Ashadh]**

⇒ The JK flip –flop is versatile and widely used type of flip–flop. The J and K designations for inputs.

Operation:

The output Q is ANDed with K and clock pulse inputs so that the flip–flop is cleared during a clock pulse only if Q was previously 1. Similarly, output Q' is ANDed with J and clock pulse inputs so that the flip–flop is set with clock pulse only if Q' was previously 1.

When both J and K are 1, the clock pulse is transmitted through one AND gate only the one whose input is connected to the flip–flop output. Which is presently equal to 1 upon application of a clock pulse and the flip-flop is cleared. If Q = 1, the output of the lower AND gate becomes 1 and the flip–flop is set. In either use, the output state of flip–flop is complemented.



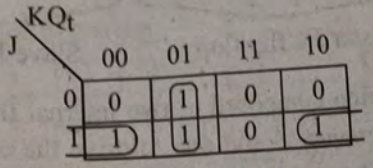
Characteristic table:

J	K	Q_t	Q_{t+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

Q_t = output before apply the input

Q_{t+1} = Output after applying the input

Using K –map



∴ Characteristics equation = $\bar{K}\,Q_t + J\bar{Q}_t$

2. **Differentiate between edge triggered and pulse triggered flip-flops. Explain about master slave flip–flop.** [2064 Falgun]

⇒

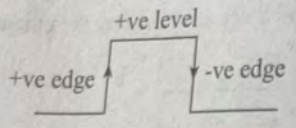


Fig.: Positive pulse

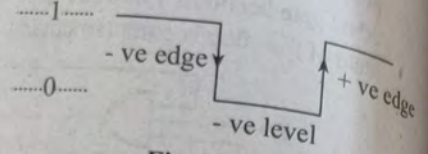


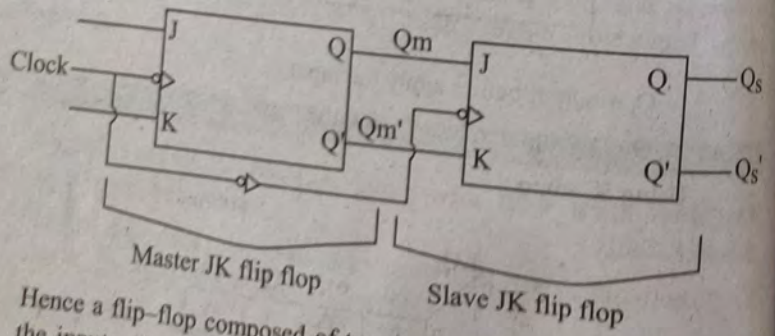
Fig.:Negative pulse

An edge triggered flip–flop changes state either at the positive edge *(rising edge)* or at the negative edge *(falling edge)* of the clock pulse and is sensitive to its input only at this transmission of the clock.

Examples. S-R, D, J-K, T flip flops.

A pulse triggered flip–flop changes state either at the positive pulse *(positive level of pulse)* or at the negative pulse *(negative level of pulse)* of the applied clock pulse.

A master salve flip–flop is constructed from two separate flip–flops one circuit serves as a master and other as a slave and the overall circuit is referred to as the master slave flip-flop. For example, let we take master slave JK flip-flop. This is the type of flip-flop which is composed of two sections master and slave. The master section is basically gated latch and slave is also the same except that it is clocked on inverted clock pulse and is controlled by output of master section rather than by external JK input.



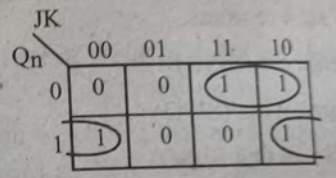
Hence a flip–flop composed of two internal flip–flops one to receive the inputs *(the master)* and one to drive the outputs *(the slave)*. The master flip–flop receives its information during the leading edge of a clock pulse, and the slave or output flip–flop receives its information from the master during the failing edge of the pulse.

3. **Derive characteristic equation of a JK flip flop. How do you make it a toggle flip flop? Draw the input and output wave form of JK flip flop.** [2071 Chaitra]

⇒ **Characteristics table for JK flip flop is**

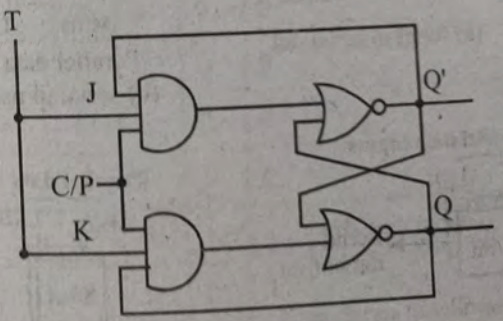
Q_n	J	K	Q_{n+1}	Remarks
0	0	0	0	No change
0	0	1	0	Reset
0	1	0	1	Set
0	1	1	1	Toggle
1	0	0	1	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	0	Toggle

For characteristic equation,

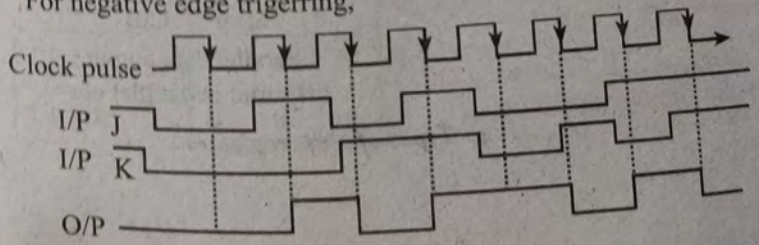


$\therefore \quad Q_{n+1} = Q'_nJ + Q_nK'$

Toggle flip-flop is made by shorting the input J and K terminals of JK flip-flop.



For negative edge trigerring,



Chapter – 7

REGISTERS

7.1 Introduction

A register is a group of binary cells (flipflops) suitable for holding information. It is a group of flipflops (FF) sensitive to pulse transition. A register is a digital circuit with two basic functions: data storage and data movement. A register consists of one or more flipflops used to store and shift data. An 'n' bit register consists of a group of 'n' flipflops capable of storing 'n' bits of binary information. A register capable of shifting its binary information in one or more direction is called shift register.

7.2 Types of Shift Registers

There are four types of shift registers:

i. Serial in serial out (SISO)

ii. Serial in Parallel out (SIPO)

iii. Parallel in Serial Out (PISO)

iv. Parallel in Parallel Out (PIPO)

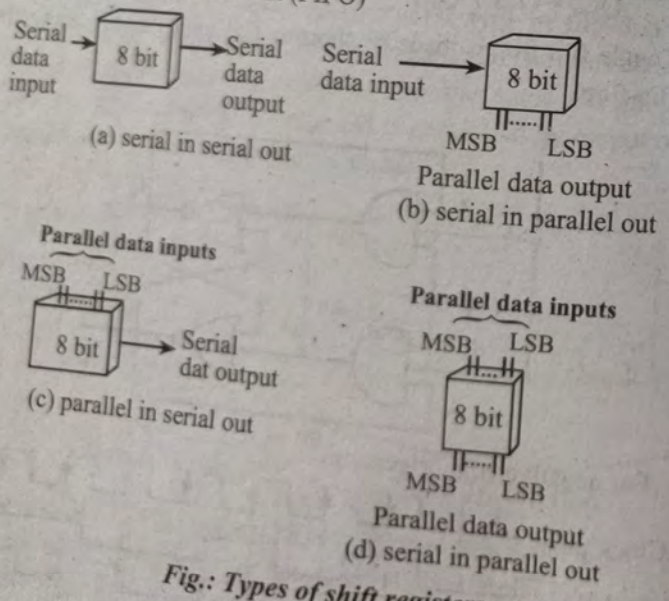


Fig.: Types of shift registers

i) Serial in Serial Out (SISO) Shift Register

SISO shift registers accepts data serially i.e., one bit at a time and output taken from it is also in serial form.

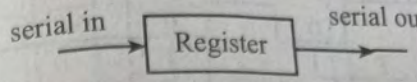


Fig.: Block diagram of right shift register

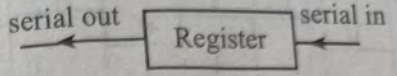


Fig.: Block diagram of left shift register

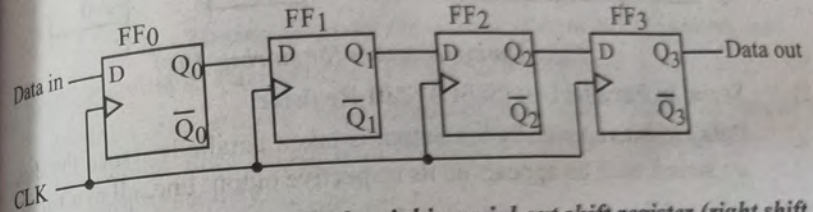


Fig.: Logic diagram of serial in serial out shift register (right shift register).

Let's explain its operation by storing and retrieving "1011". Let the initially the register content is clear i.e., 0000. The rightmost data is entered first.

Clock	Register content				
	Q_0	Q_1	Q_2	Q_3	
Initially	0	0	0	0	
CLK 1	1	0	0	0	
CLK 2	1	1	0	0	
CLK 3	0	1	1	0	
CLK 4	1	0	1	1	All data stored after 4th clock
CLK 5	0	1	0	1	
CLK 6	0	0	1	0	
CLK 7	0	0	0	1	
CLK 8	0	0	0	0	Register clear after 8th clock

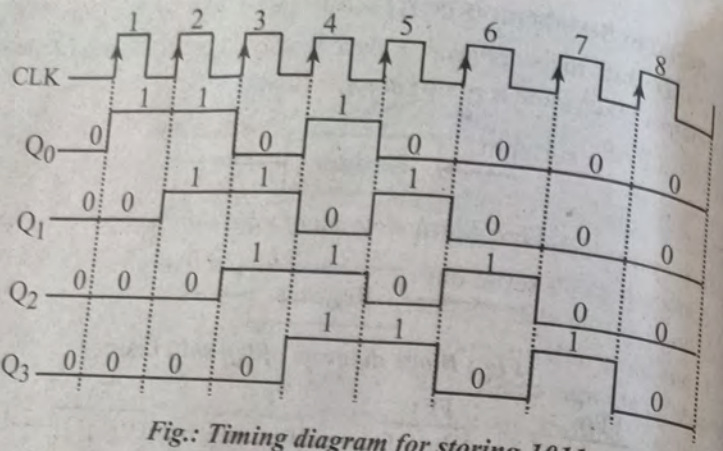


Fig.: Timing diagram for storing 1011

2) Serial in Parallel Out (SIPO) Shift Register

Data are entered serially but output is taken parallelly. Once the data are stored, each bit appears on its respective output line, all at a time.

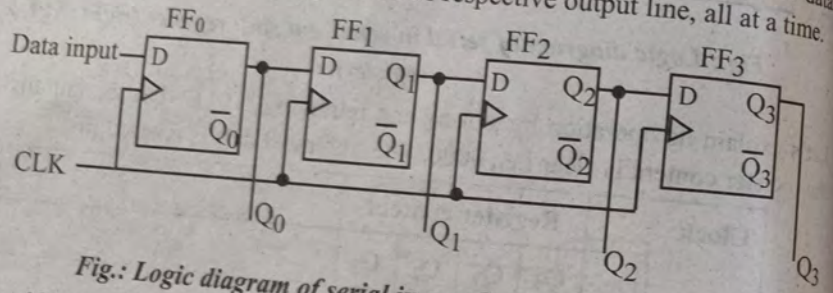


Fig.: Logic diagram of serial in parallel out (SIPO) shift register.

Let 1010 is stored from rightmost bit.

CLK	Register content				
	Q_0	Q_1	Q_2	Q_3	
Initially	0	0	0	0	Data is completely stored after 4th clock. Output is available all at a time parallelly.
CLK 1	0	0	0	0	
CLK 2	1	0	0	0	
CLK 3	0	1	0	1	
CLK 4	1	0	1	0	

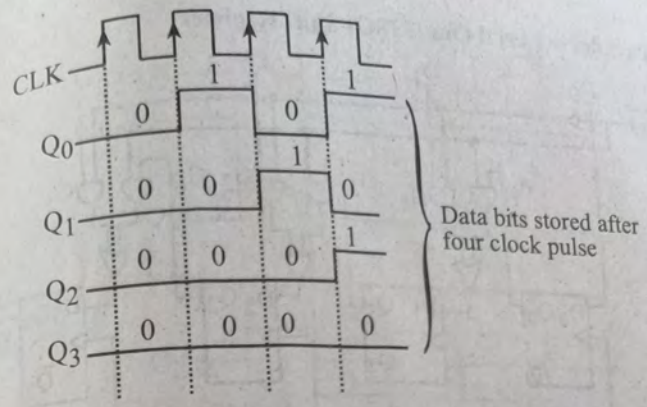


Fig.: Timing diagram for storing 1010 in SIPO shift register.

3) Parallel in Parallel Out (PIPO) Shift Register

Data are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis. Output is available simultaneously.

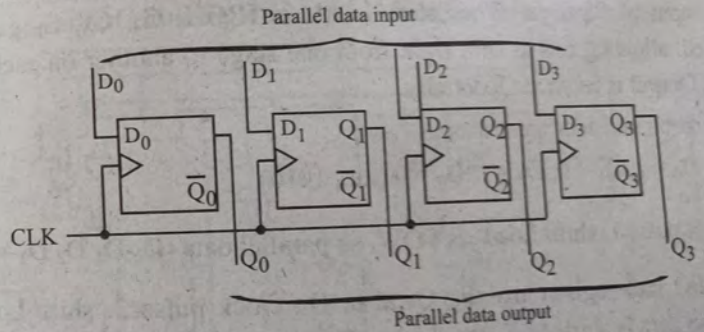


Fig.: Logic diagram of 4-bit parallel in parallel out (PIPO) shift register.

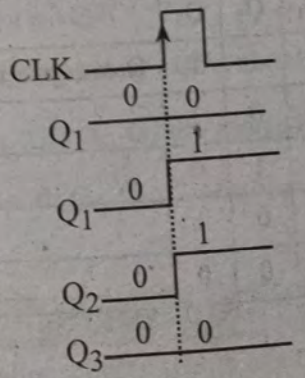


Fig.: Timing diagram for storing 0110 in PIPO shift register

4) Parallel in Serial Out (PISO) Shift Register

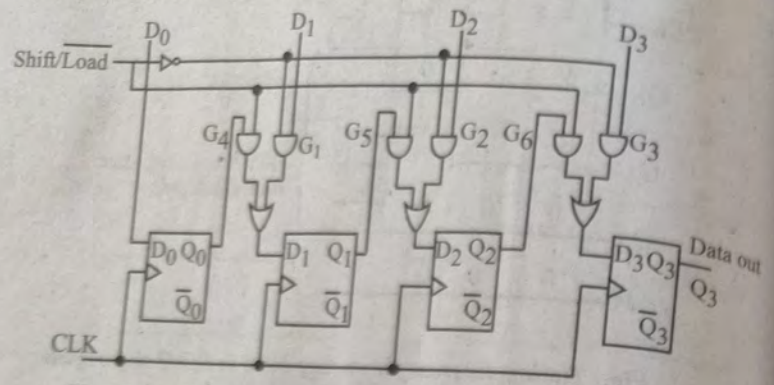


Fig.: Logic diagram of parallel in serial out (PISO) shift register.

Data are entered parallelly but taken in a serial mode. When shift/$\overline{\text{Load}}$ is LOW; G_1, G_2, G_3 gates are enabled so it allows data D_1, D_2, D_3 to be applied at D input of flipflops. When shift/$\overline{\text{Load}}$ is HIGH; G_4, G_5, G_6 gates are enabled, allowing bits to shift right from one stage to another on each clock pulse. Output is taken at Q_3 serially.

Let's consider input data as

$D_0 = 1, D_1 = 0, D_2 = 1, D_3 = 0$ (i.e., 1010)

On clock pulse 1, shift/$\overline{\text{Load}}$ is LOW, so parallel data ($D_0\ D_1\ D_2\ D_3 = 1010$) are loaded into register making Q_3 at 0. On clock pulse 2, shift/$\overline{\text{Load}}$ is HIGH, so data is shifted i.e., the 1 from Q_2 is right shifted onto Q_3 and so on.

	Register content				
CLK	Q_0	Q_1	Q_2	Q_3	
1	1	0	1	0	Data is entered parallelly
2	0	1	0	1	
3	0	0	1	0	
4	0	0	0	1	
5	0	0	0	0	

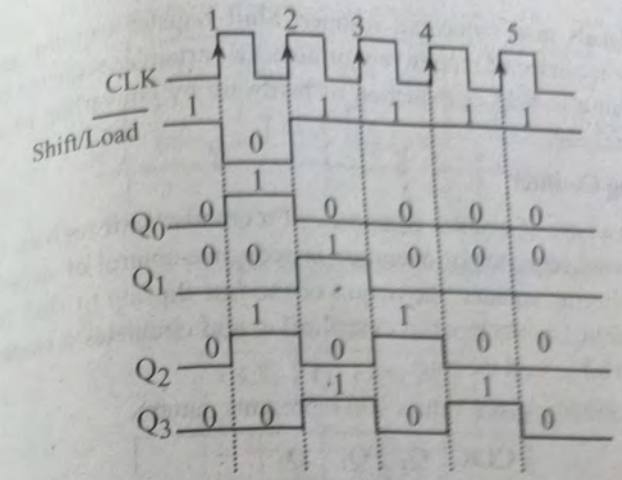


Fig.: Timing diagram for storing 1010 in PISO shift register

7.3 Bidirectional Shift Register (Universal Register)

Data can be shifted either right or left in the bidirectional shift register.

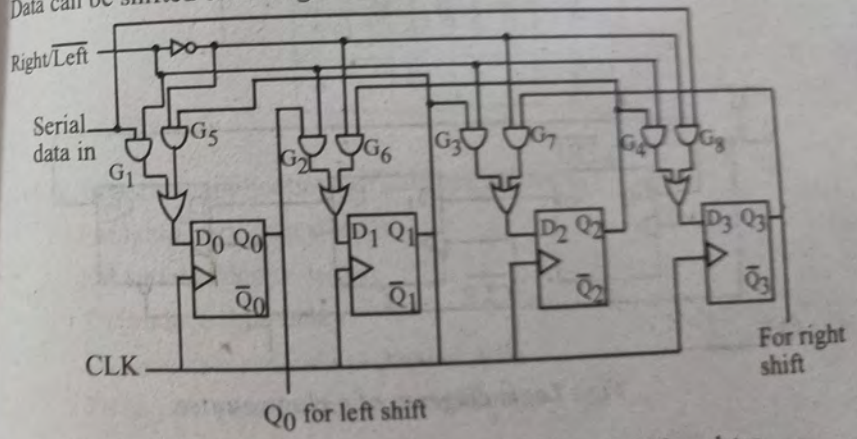


Fig.: Logic diagram of bidirectional shift register.

When Right/$\overline{\text{Left}}$ = 1; G_1, G_2, G_3, G_4 are enabled and right shift of data occurs. When Right/$\overline{\text{Left}}$ = 0; G_5, G_6, G_7, G_8 are enabled and left shift of data occurs.

7.4 Application of Shift Register

Shift registers are used in almost every sphere of a digital logic system. Shift register can be used to count number of pulses entering into a system as ring counter or switched-tail counter. As ring counter, it can generate various

control signals in a sequential manner. Shift register can also generate a prescribed sequence of repetitively or detect a particular sequence from data input. It can also help in reduction of hardware by converting parallel data feed to serial one.

1) Ring Counter

It is a type of counter composed of a circular shift register. It is used to count sequence of operation in sequence control of stepper motor, etc. In ring counter, the output of the last flipflop of shift register is connected to the input of first flipflop and circulates a single one (or zero) bit around the ring.

E.g., Initial register values 100 represents pattern.

CLK	Q_0	Q_1	Q_2	
0	1	0	0	Initially
1	0	1	0	
2	0	0	1	
3	1	0	0	

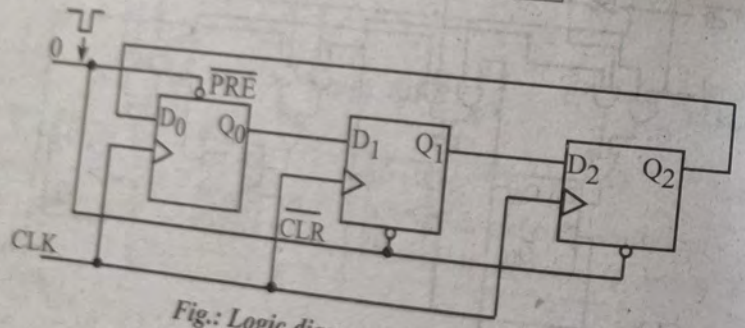


Fig.: Logic diagram of a ring counter.

Initially $\overline{PRE} = 0$, $\overline{CLR} = 0$ should be given i.e., $100 = Q_0\, Q_1\, Q_2$ and $\overline{PRE} = 1$, $\overline{CLR} = 1$ is given. Now at every clock pulse, the 1 is shifted to next stage.

Drawbacks:

The requirement of initialization is a disadvantage of ring counter.

2) Johnson Counter (Switched-Tail Counter)

This counter eliminates the limitation of ring counter. In Johnson counter, complement output of last flipflop is connected to input of first flipflop.

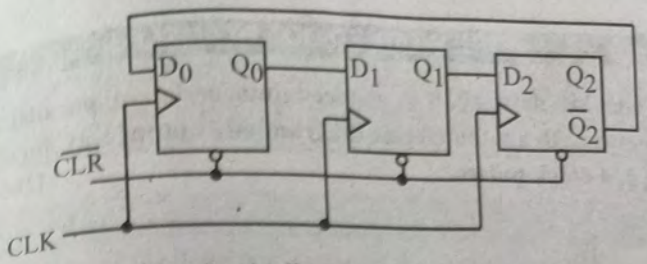


Fig.: Logic diagram of Johnson counter.

Initially $\overline{CLR} = 0$, is given to reset to 000 state.

CLK	Q_0	Q_1	Q_2
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	0	0	1
6	0	0	0

Hence, important applications of shift registers are:

- For temporary data storage
- For serial adder
- To produce time delay
- To convert serial data to parallel data
- To convert parallel data to serial data
- As ring counter
- As Johnson counter

MORE WORKED OUT EXAMPLES:

1. **Four bit data 1010 is entered into serial in parallel out shift register. Draw the circuit diagram and output waveforms after 1, 2, 3, 4 clock pulses.** **[2064 Jestha]**

⇒

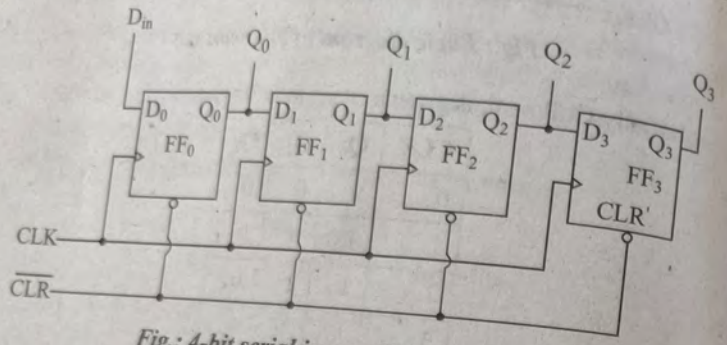
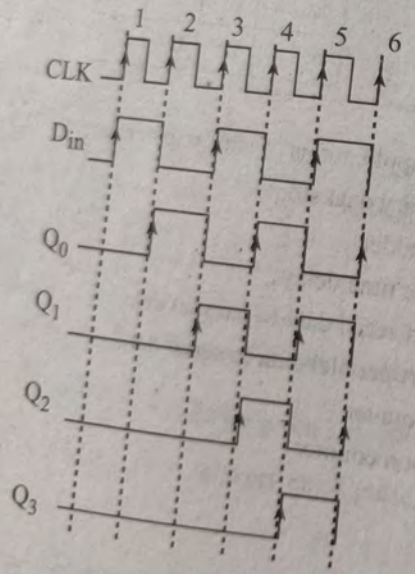


Fig.: 4-bit serial in parallel out shift register

Now, timing diagram for above circuit is



Chapter – 8
COUNTERS

8.1 Introduction

The combination of flipflops that performs the counting operation are known as counter. A counter is probably one of the most useful and versatile subsystems in a digital system. It is a sequential circuit that passes through predefined number of states.

Counters are classified into two broad categories according to the way they are clocked:

i. Asynchronous (or ripple or serial) counter

ii. Synchronous (or parallel) counter.

1. **Asynchronous (or Ripple or Serial) Counter**

   In this counter, the first flipflop is clocked by the external clock pulse and then each successive flipflop is clocked by the output of the preceding flip–flop. Asynchronous counter is simple and straightforward in operation and its construction usually requires a minimum of hardware. It does have a speed limitation.

2. **Synchronous (or Parallel) Counter**

   In synchronous counter, the clock input is connected to all of the flipflops so that they are clocked simultaneously. An increase in speed of operation can be achieved by use of synchronous counter.

8.2 Asynchronous Counters

3-bit Ripple Up Counter (or MOD-8 Up Counter)

The 3-bit ripple up counter constructed from JK flipflop is shown below. The JK input are tied together to $+V_{CC}$ such that at each negative transition of its clock input, the flipflop toggles its states.

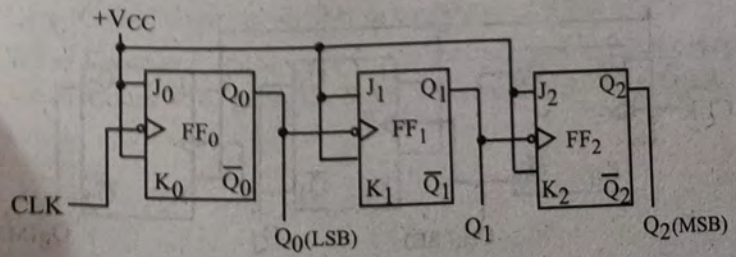


Fig.: A 3-bit binary ripple up counter

Truth table

Clock (NT)	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

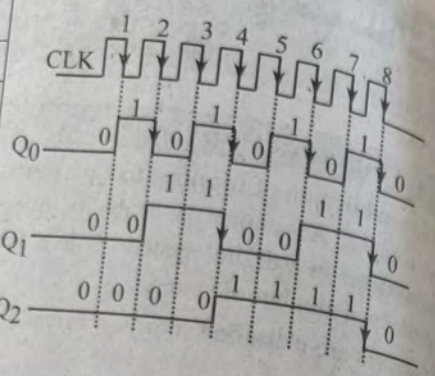


Fig.: Timing diagram

Since it is a 3-bit up counter, it counts from 000 – 111 in binary (i.e., 0 to $2^3 - 1$ in decimal). Modulus of a counter is the total number of states through which the counter can progress. The maximum number of states = $2^n = 2^3 = 8$. For 3-bit counter, 3 flipflops are required. Therefore, 3-bit counter is also known as MOD-8 counter.

The waveforms illustrated above show the action of the counter as the clock runs. Initially, all the flipflops are reset to produce 0 outputs. If we consider Q_0 to be the LSB and Q_2 MSB, we can say the contents of the counter is Q_2 Q_1 Q_0 = 000. Every time there is a clock NT, FF_0 will change state. Since Q_0 acts as the clock for FF_1, each time the waveform at Q_0 goes low, FF_1 will toggle. Since Q_1 acts as the clock for FF_2, each time the waveform at Q_1 goes low, FF_2 will toggle.

3-bit Ripple Down Counter (MOD-8 Ripple Down Counter)

Only first flipflop is given clock pulse from the external source. The remaining flipflop gets clock from the output of preceding flipflop i.e., $\overline{Q}$.

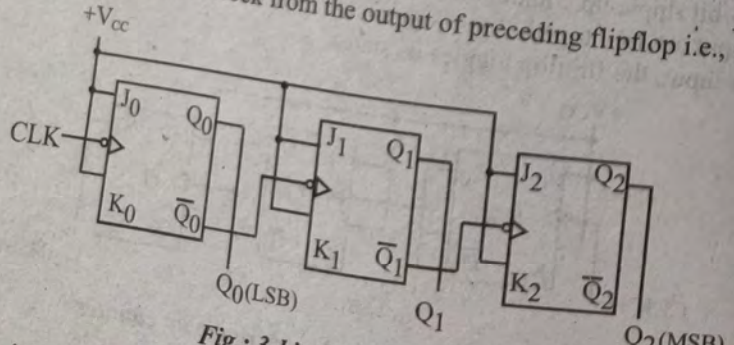


Fig.: 3-bit ripple down counter

Truth table

Clock (NT)	Q_2	Q_1	Q_0	Count
0	1	1	1	7
1	1	1	0	6
2	1	0	1	5
3	1	0	0	4
4	0	1	1	3
5	0	1	0	2
6	0	0	1	1
7	0	0	0	0

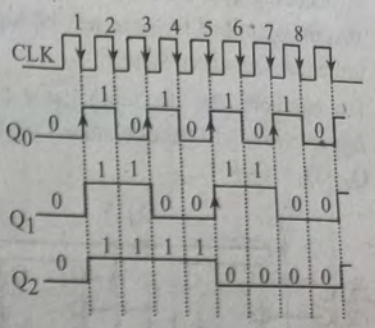


Fig.: Wavefoms

3-bit Ripple Up/Down Counter

It is the combination of the two counter discussed previously. As from above two counter, we observe that for

(a) up-counter, output Q is connected to CLK of successive FF

(b) down-counter, output $\overline{Q}$ is connected to CLK of successive FF

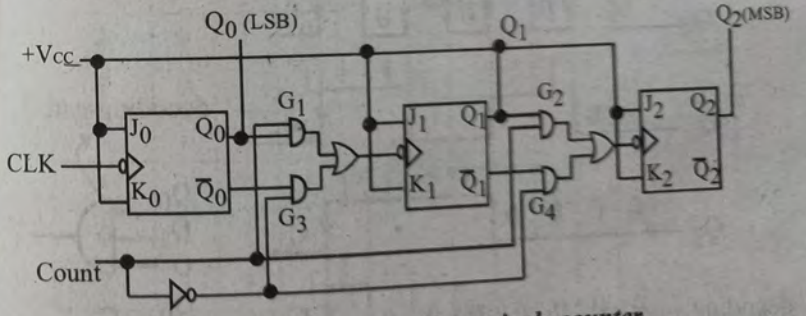


Fig.: 3-bit up/down ripple counter

Here 'Count' controls whether to count up or down.

When Count = 1, gates G_1 & G_2 are enabled, so it performs up count.

When Count = 0, gates G_3 & G_4 are enabled, so it performs down count.

8.3 Decoding Gates

A decoding gate can be connected to the outputs of a counter in such a way that the output of the gate will be high or low only when the counter contents are equal to a given state.

For example, the decoding gates connected to the 3-bit ripple counter in figure below will decode state 7 ($Q_2Q_1Q_0$ =111). The gate will be high when $Q_0 = Q_1 = Q_2 = 1$.

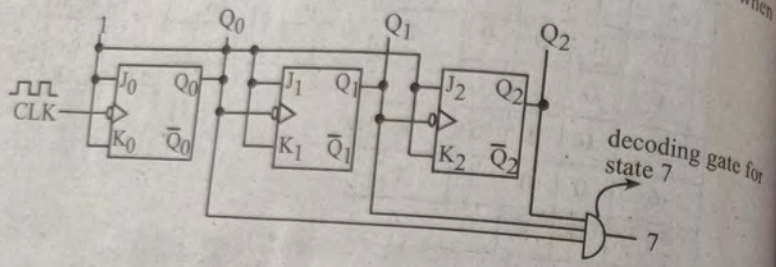


Fig.: Decoding gate for state 7

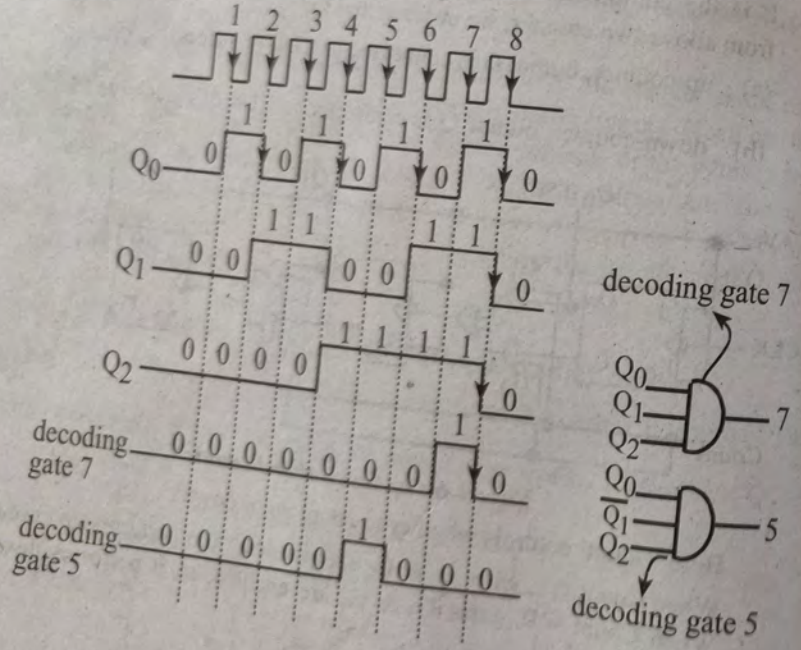


Fig.: Waveforms

8.4 Synchronous Counter (Parallel Counter)

3-bit Synchronous Up Counter (or MOD-8 Synchronous Up Counter)

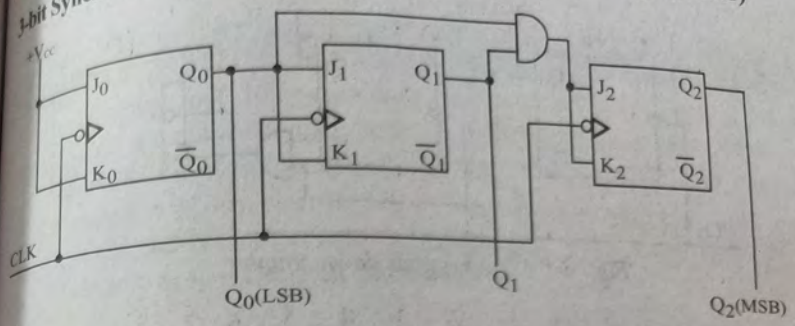


Fig.: MOD-8 synchrnonous up counter

Truth table

Clock (NT)	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

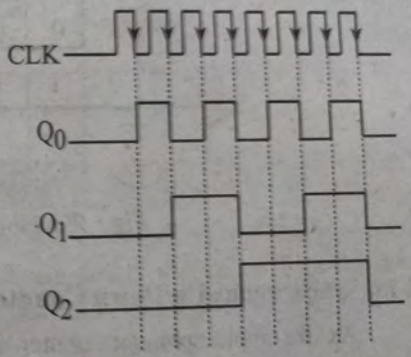


Fig.: Timing diagram

Initially, $Q_0 = Q_1 = Q_2 = 0$. When first clock pulse is applied, Q_0 toggles to 1. J_1 is still 0 for first clock pulse, so $Q_1 = 0$; similarly, $Q_2 = 0$. For second clock pulse, Q_0 toggles to 0, $J_1 = 1$, so Q_1 toggles to 1. Q_2 remains 0 since for second clock pulse, $Q_1 = 0$. For third clock pulse, Q_0 toggles to 1, $J_1 = 0$, so $Q_1 = 1$ and $Q_2 = 0$. For the fourth clock pulse, Q_0 toggles to 0, $J_1 = 1$, so Q_1 toggles to 0, $J_2 = Q_0Q_1 = 1$, so Q_3 toggles to 1, and so on.

3-bit Synchronous Down Counter (MOD-8 Synchronous Down Counter)

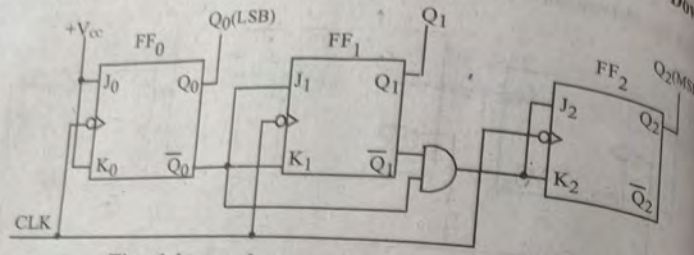


Fig.: 3-bit synchronous down counter

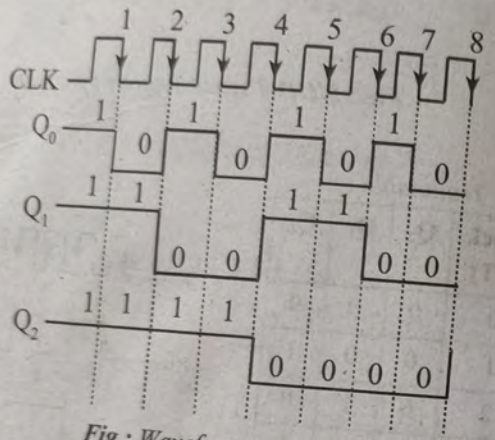


Fig.: Waveforms

3-bit Synchronous Up/Down Counter

As like ripple up/down counter, it is also the combination of up and down synchronous counter.

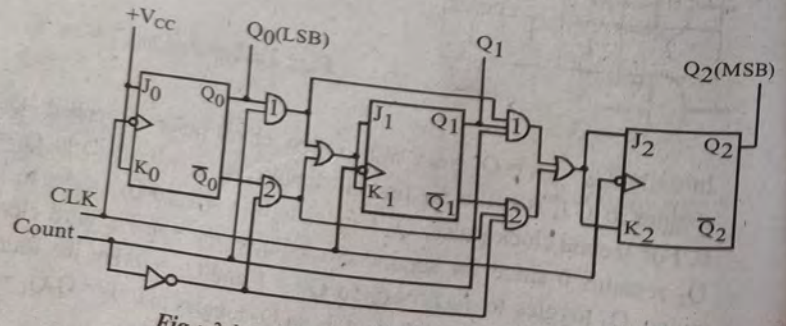


Fig.: 3-bit synchronous up/down counter

When Count = 1, AND gate 1 is ON, so Q_0 is transferred to J_1 and K_1, and so it starts counting up. When Count = 0, AND gate 2 is ON, so $\overline{Q}_0$ is transferred to J_1 and K_1, and so it starts counting down.

8.5 Decade Counter (MOD-10 Counter or BCD Counter)

It is also known as mod-10 counter or BCD counter. It counts from 0 to 9 in decimal. Thus it requires 10 clock pulse for resetting. It counts from 0000 to 1001 and skips rest of the states and this is possible because at 10^{th} clock pulse, it generates its own clear signal from decoding gate and resets to 0000. That is, when the counter counts $Q_3Q_2Q_1Q_0 = 1010$, then the output of NAND gate is low, so it clears all the flipflops.

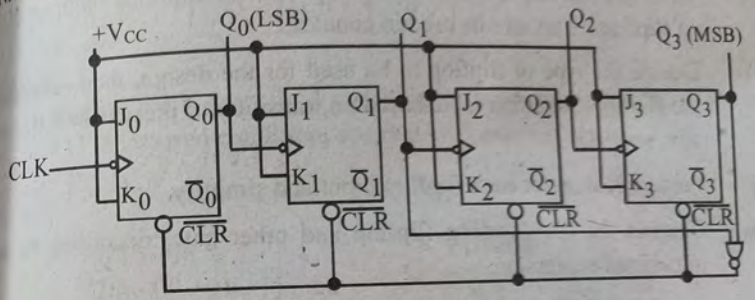


Fig.: Asynchronous decade counter

8.6 Presettable Counters

The presettable counter is one in which the counter starts counting not from zero but from any numbers. The figure below shows the 4-bit resettable counters, it has a special 'Load' terminal and counting begins with P_3, P_2, P_1, P_0 which can be any numbers between 0000 and 1111.

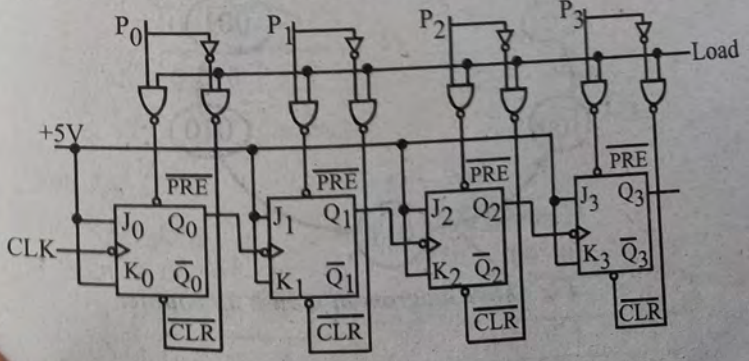


Fig.: 4-bit presettable counter

When Load is LOW, all NAND gates have high outputs, so all PRESET and CLEAR are inactive. Therefore, it counts in normal way. When Load is HIGH, the inputs P_3, P_2, P_1, P_0 and their complements $\overline{P_3}$, $\overline{P_2}$, $\overline{P_1}$, $\overline{P_0}$ passes through NAND gates. This action presets the counter to $P_3P_2P_1P_0$ states in the initial. When Load reverts to LOW again, then the counting starts and successive clock pulse produces the remaining states till to maximum.

8.7 Counter Design as a Synthesis Problem

Following steps are to be followed for designing synchronous counters:

1) Draw the state diagram from the given word description problem.
2) Draw the next state table and find number of flipflops required. No. of flipflops = no. of bits used in counter.
3) Decide the type of flipflop to be used for the design, then determine the flipflop excitation table based on transition of present state (Q_n) to next state (Q_{n+1}).
4) Prepare K-map for each flipflop input and simplify.
5) Connect the circuit using flipflop and other gates according to the minimized expression.

Example:

1. Design a MOD–6 up counter (i.e. 6–state counter)

⇒

Step 1: Draw the state diagram

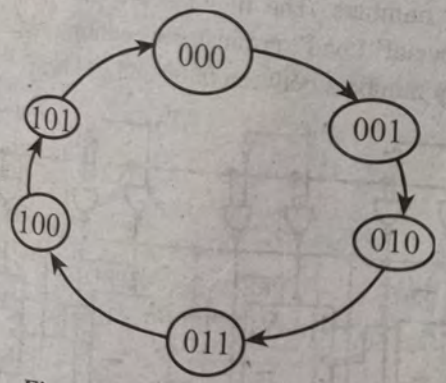


Fig.: State diagram of mod-6 up counter

Step 2 and step 3: Next state table and excitation table

Present State (Q_n)			Next State (Q_{n+1})			FF Excitation					
Q_{2n}	Q_{1n}	Q_{0n}	Q_{2n+1}	Q_{1n+1}	Q_{0n+1}	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	0	0	0	×	1	0	×	×	1

Step 4:

Prepare the K-map for J and K input from transition table. 110 and 111 are unused condition, so are don't care conditions.

For J_2,

Q_{2n} \ $Q_{1n}Q_{0n}$	00	01	11	10
0	0	0	1	0
1	×	×	×	×

$J_2 = Q_{1n}Q_{0n}$

For K_2,

Q_{2n} \ $Q_{1n}Q_{0n}$	00	01	11	10
0	×	×	×	×
1	0	1	×	×

$K_2 = Q_{0n}$

For J_1,

Q_{2n} \ $Q_{1n}Q_{0n}$	00	01	11	10
0	0	1	×	×
1	0	0	×	×

$J_1 = \overline{Q}_{2n}Q_{0n}$

For K_1,

Q_{2n} \ $Q_{1n}Q_{0n}$	00	01	11	10
0	×	×	1	0
1	×	×	×	×

$K_1 = Q_{0n}$

For J_0,

Q_{2n} \ $Q_{1n}Q_{0n}$	00	01	11	10
0	1	×	×	1
1	1	×	×	×

$J_0 = 1$

For K_1,

Q_{2n} \ $Q_{1n}Q_{0n}$	00	01	11	10
0	×	1	1	×
1	×	1	×	×

$K_0 = 1$

Step 5:

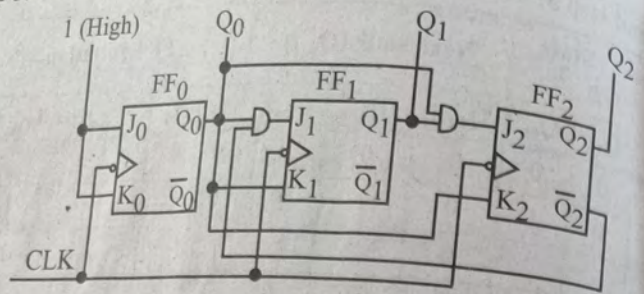


Fig.: Mod-6 synchronous up counter

2. Design a 2-bit up/down counter using T flipflops.

⇒

Step 1: Let Y = 1 (up counter), Y = 0 (down counter)

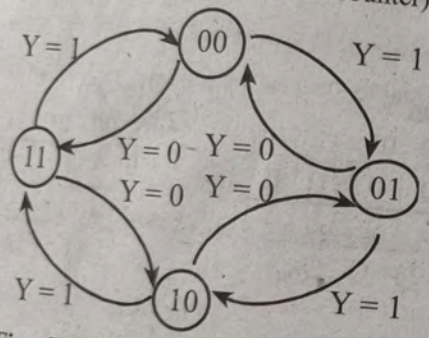


Fig.: State diagram of 2-bit up/down counter

Step 2 and Step 3:

The no. of bits are 2, so no. of flipflops = 2

Present State (Q_n)		Next State (Q_{n+1})			
		Y = 0 (down)		Y = 1 (up)	
Q_{1n}	Q_{0n}	Q_{1n+1}	Q_{0n+1}	Q_{1n+1}	Q_{0n+1}
0	0	1	1	0	1
0	1	0	0	1	0
1	0	0	1	1	1
1	1	1	0	0	0

Present State Q_n		Control Input	Next State (Q_{n+1})		Flipflop Excitation	
Q_{1n}	Q_{0n}	Y	Q_{1n+1}	Q_{0n+1}	T_1	T_0
0	0	0	1	1	1	1
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	1	1	1	0	1
1	1	0	1	0	0	1
1	1	1	0	0	1	1

Step 4: K-mapping for T_1 and T_0 (inputs are Q_{1n}, Q_{0n}, Y)

For T_1,

Q_{1n} \ Q_{0n}Y	00	01	11	10
0	1	0	1	0
1	1	0	1	0

$T_1 = \overline{Q}_{01}\overline{Y} + Q_{01}Y$

$= \overline{(Q_{01} \oplus Y)}$

For T_0,

Q_{1n} \ Q_{0n}Y	00	01	11	10
0	1	1	1	1
1	1	1	1	1

$T_0 = 1$

Step 5:

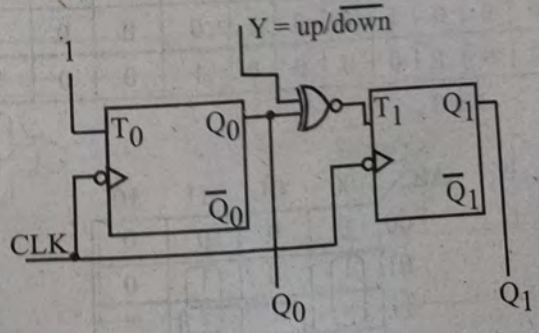
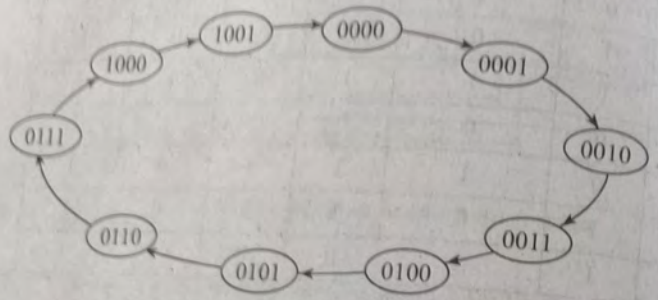


Fig.: 2-bit synchronous up/down counter.

MORE WORKED OUT EXAMPLES:

1. **Design the synchronous decade counter and also show the timing diagram.** [2069 Chaitra]

⇒



State diagram for synchronous decade counter

Present state				Next state				Flip-flop			
A	B	C	D	P	Q	R	S	T_A	T_B	T_C	T_D
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	1	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	1	0	0	0	1

For T_A

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	×	×	×	×
10	0	1	×	×

$T_A = AD + BC'D' + BCD$

For T_B

AB \ CD	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	×	×	×	×
10	0	0	×	×

$T_B = RD$

For T_C

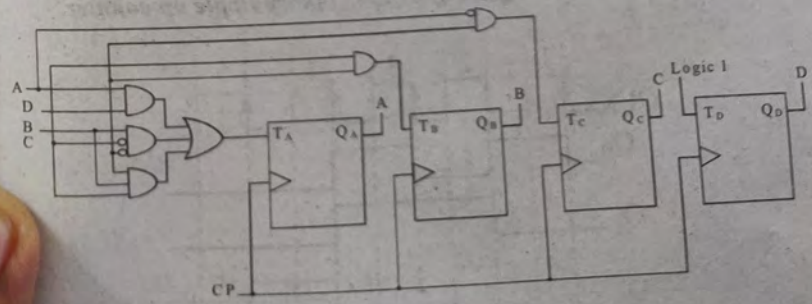
AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	×	×	×	×
10	0	0	×	×

$T_C = A'D$

For T_D

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	×	×	×	×
10	1	1	×	×

$T_D = 1$



2. **Design a MOD–5 binary ripple up counter. What are the advantages of a synchronous counter over a ripple counter? [2066 Bhadra]**

⇒

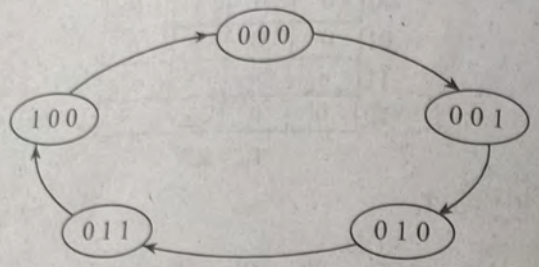


Fig.: State diagram of MOD-5-binary ripple up counter

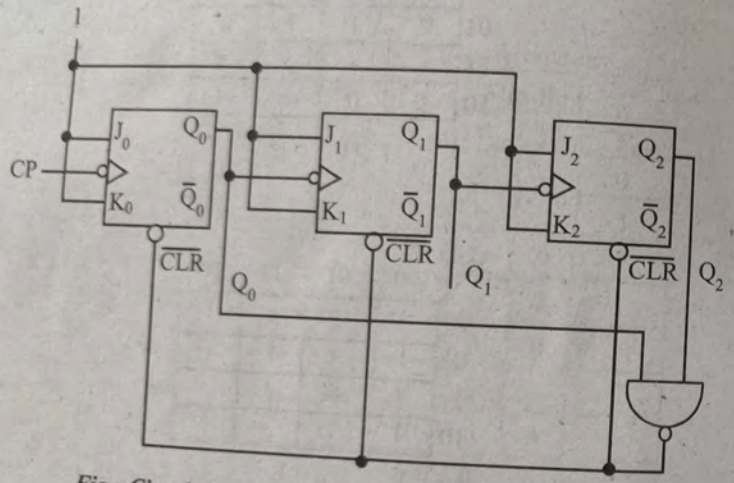


Fig.: Circuit diagram of MOD-5 binary ripple up counter.

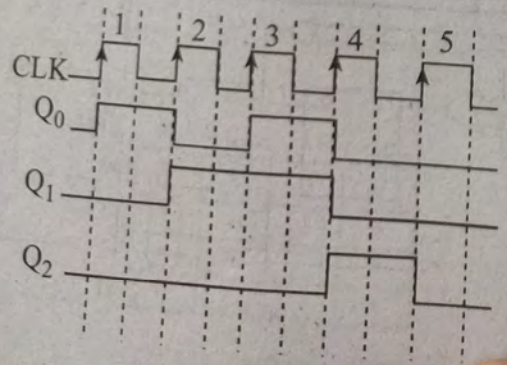


Fig.: Timing diagram

Advantages of a synchronous counter over a ripple counter

1. Synchronous counter is free from glitches.
2. Faster in operation if high frequency is applied.

3. **Design a MOD–5 binary synchronous up counter using the excitation table and draw its timing diagram. [2066 Bhadra]**

⇒

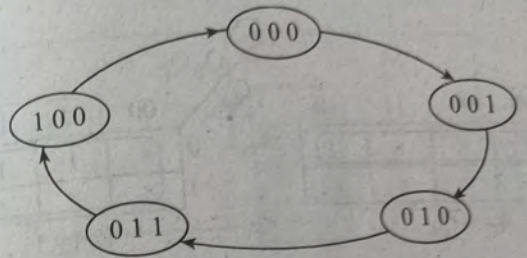


Fig.: Static diagram of Mod-5 binary up counter

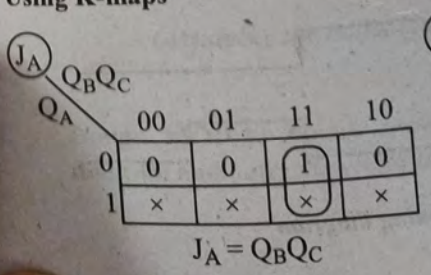
Excitation table of JK flip–flop

Q_t	Q_{t+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

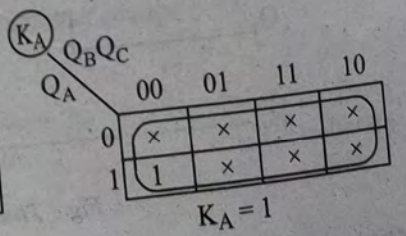
State excitation table:

Present state			Next state			J_A	K_A	J_B	K_B	J_C	K_C
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}						
0	0	0	0	0	1	0	×	×	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	0	0	0	×	1	0	×	0	×

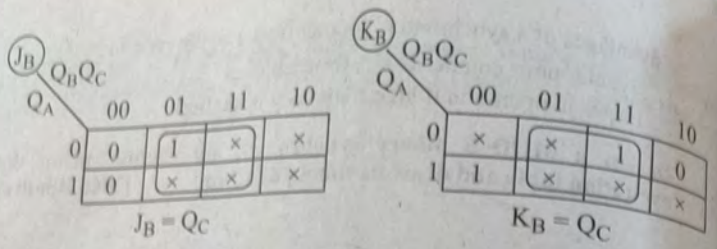
Using K-maps

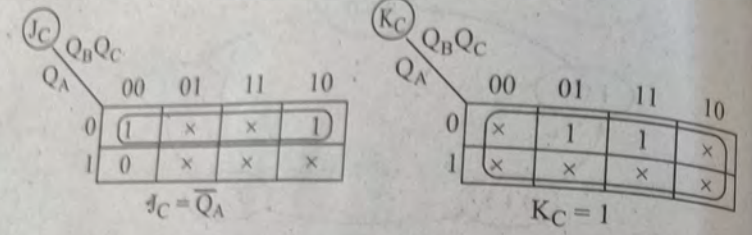


$J_A = Q_BQ_C$



$K_A = 1$





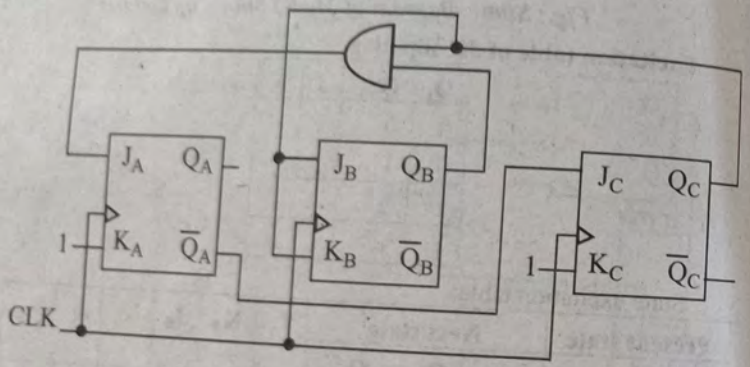


Fig.: 3-bit binary synchronous up counter

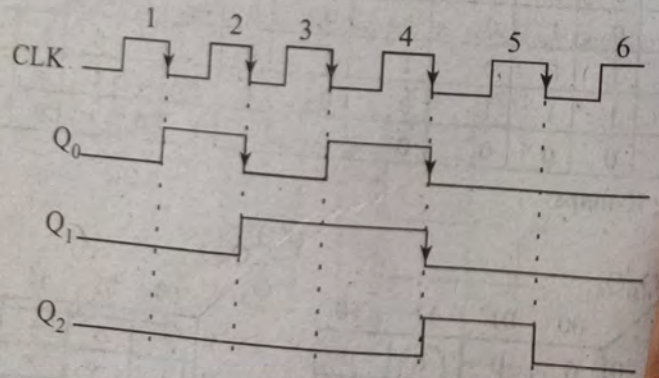


Fig.: Timing diagram

4. **Design a 3-bit down counter, using master–slave JK flipflops and show the timing diagram for 3 clock cycles, assuming that the initial reading of the counter is 0.** **[2065 Shrawan]**

⇒

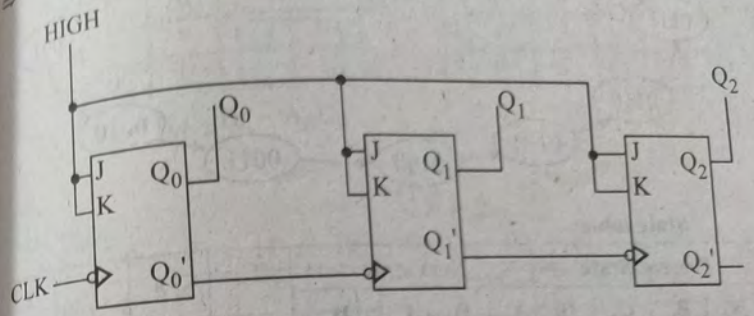


Figure above is the layout of 3-bit down counter. It resembles as the synchronous countdown binary counter goes through the binary states in reverse order, from 1111 down to 0000 so termed as down counter. For typically design purpose here we use master slave flip-flop, using three JK flip-flops counting three bits binary number on reverse order.

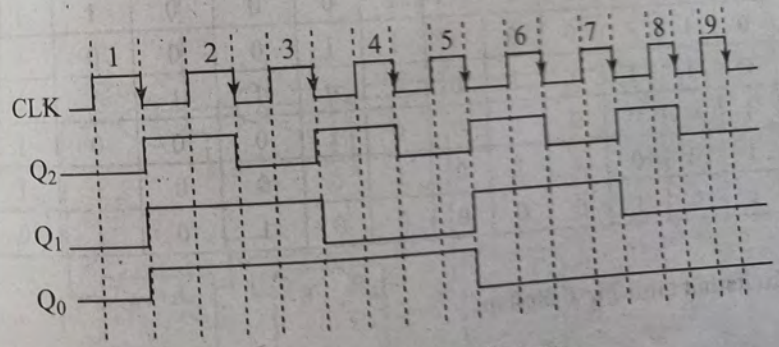
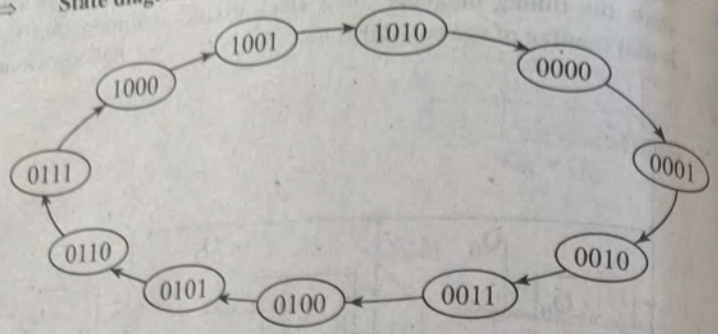


Fig.: Timing diagram

(Assuming the initial reading of counter is 0.)

5. **Design MOD-11 binary counter by showing its state, circuit diagram and output waveforms.** **[2064 Jestha]**

⇒ **State diagram:**

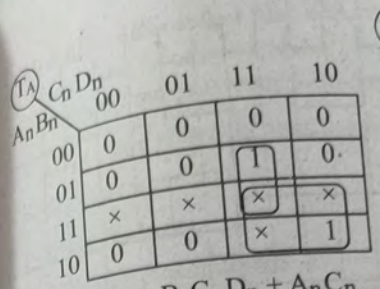


State table:

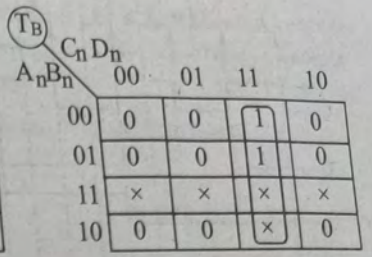
Present State				Next state				T_A	T_B	T_C	T_D
A_n	B_n	C_n	D_n	A_{n+1}	B_{n+1}	C_{n+1}	D_{n+1}				
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	0	0	0	0	1	0	1	0

Excitation table for T flipflop:

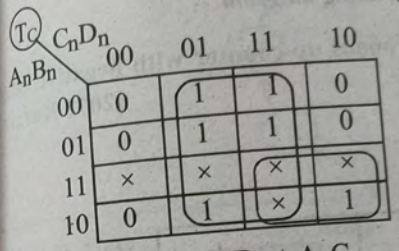
Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0



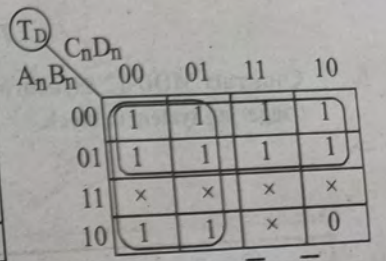
$T_A = B_nC_nD_n + A_nC_n$



$T_B = C_nD_n$



$T_C = D_n + A_nC_n$



$T_D = \overline{C}_n + \overline{A}_n$

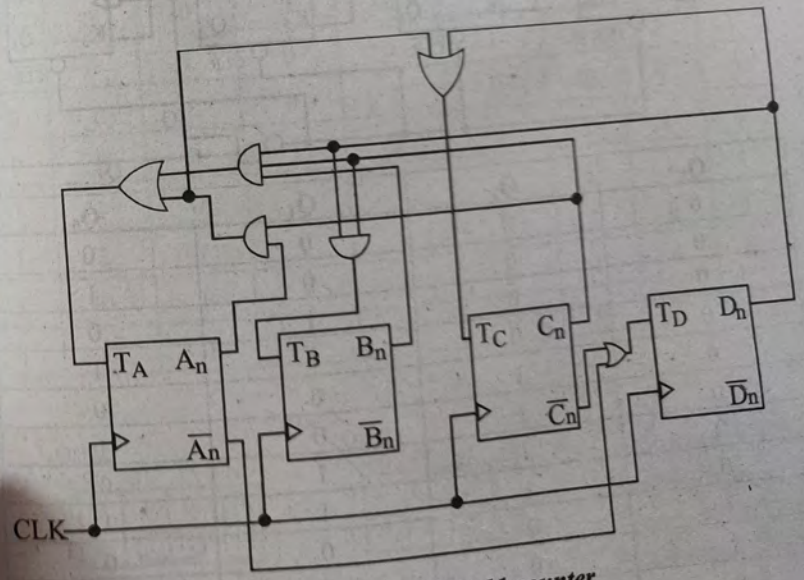


Fig.: MOD 11 counter

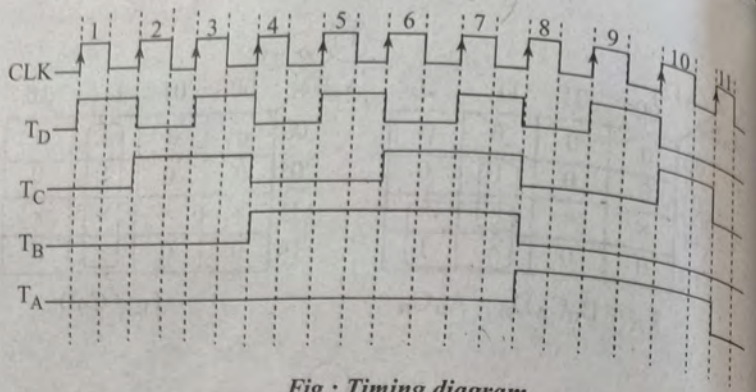
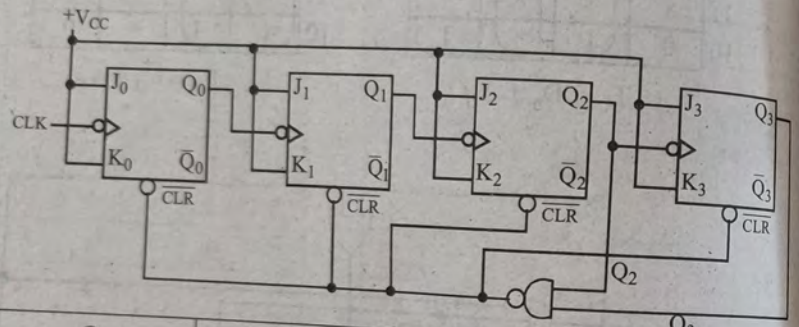


Fig.: Timing diagram

6. Construct MOD-12 asynchronous up-counter with negative edge triggering system in clock. [2072 Kartik]

⇒



Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
0	0	0	0

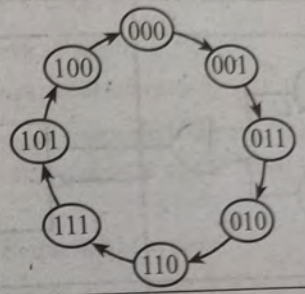
7. List the advantages and disadvantages of a synchronous counter over asynchronous counter. Design a 3-bit synchronous counter which follow gray code sequence. [2074 Ashwin]

⇒ **Advantage of synchronous counter over asynchronous counter:**

Fast operation

Disadvantages:

Complex circuit design



Present State			Next State			Flip flop		
Q_{An}	Q_{Bn}	Q_{Cn}	Q_{n+1}	Q_{Cn+1}	Q_{Cn+1}	D_A	D_B	D_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	1	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	0	1
1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	0	0

For D_A

Q_{An} \ Q_BQ_{cn}	00	01	11	10
0	0	0	0	1
1	0	1	1	1

$D_A = Q_{An}Q_{Cn} + Q_{Bn}\overline{Q}_{Cn}$

For D_B

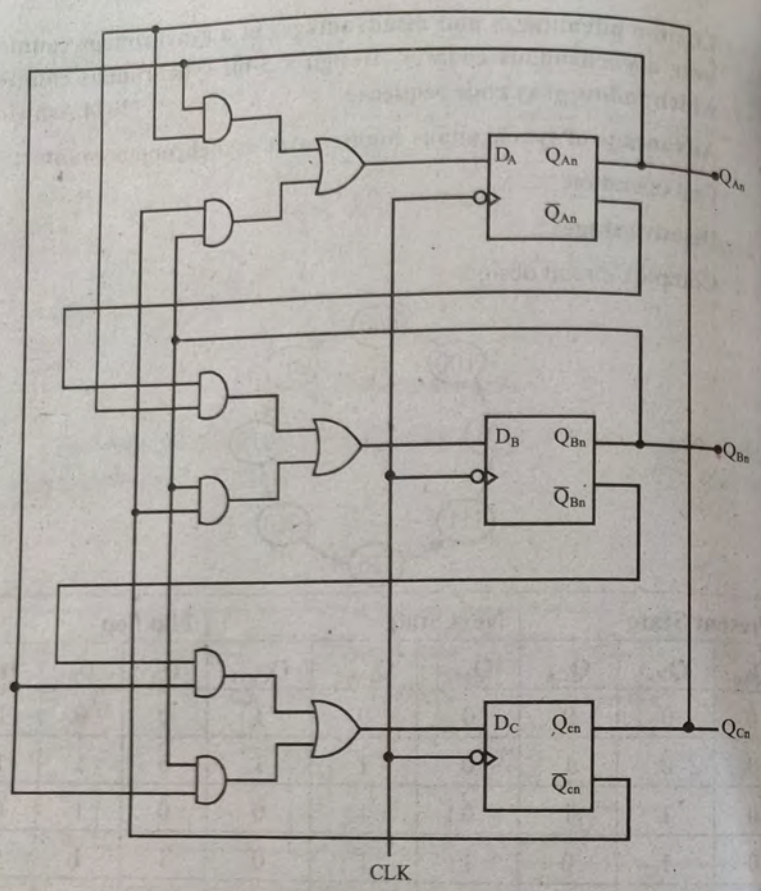
Q_{An} \ $Q_{Bn}Q_{cn}$	00	01	11	10
0	0	1	1	1
1	0	0	0	1

$D_B = \overline{Q}_{An}Q_{Cn} + Q_{Bn}\overline{Q}_{Cn}$

For D_C

Q_{An} \ $Q_{Bn}Q_{cn}$	00	01	11	10
0	1	1	0	0
1	0	0	1	1

$D_C = Q_{An}\overline{Q}_{Bn} + Q_{An}Q_{Bn}$



Chapter – 9
SEQUENTIAL MACHINES

9.1 Introduction

Sequential circuits may be classified into following two categories:

- Synchronous sequential circuits
- Asynchronous sequential circuits

1. **Synchronous Sequential Circuits**

In the synchronous sequential circuits, the contents of memory element can be changed only at the rising or falling edge of clock signals.

2. **Asynchronous Sequential Circuits**

In the asynchronous sequential circuits, the contents of memory elements can be changed at any instant of time.

	Synchronous sequential circuit		Asynchronous sequential circuit
1.	These circuits are easy to design.	1.	Difficult to design.
2.	A clocked flipflop acts as a memory element.	2.	Unclocked flipflop or time delay element is used as memory element.
3.	These circuits are slower because the delays correspond to those of the memory element.	3.	Faster as the clock is not present.
4.	The status of memory element is affected only at active edge of clock if the input is changed.	4.	The status of memory element will change any time as soon as the input is changed.

9.2 Design of Synchronous Sequential Circuit

There are two distinct models by which asynchronous sequential logic circuit can be designed:

- Moore Model
- Mealy Model

	Moore Model		Mealy Model
i.	The output depends only on present state and not on input.	i.	The output depends on present state as well as input.
ii.	It requires more no. of states and thereby more hardware.	ii.	It requires less no. of states and thereby less hardware to solve any problem.
iii.	Output is generated one clock cycle later than Mealy.	iii.	Output is generated one clock cycle earlier than Moore.
iv.	The output remains stable over entire clock period and changes only when there occurs a state change.	iv.	The output does not remain stable over entire clock period and changes over the same state.

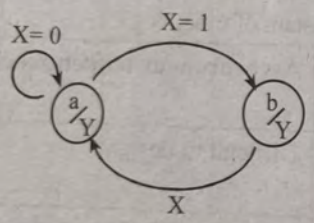


Fig.: State diagram representation

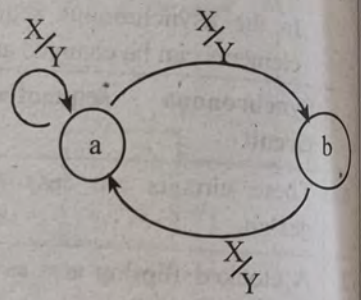


Fig.: State diagram representation

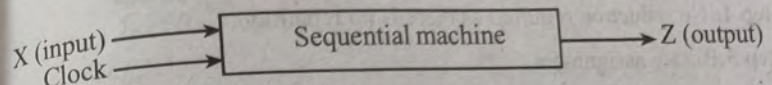
The steps to be followed for designing are:

1) Choose the model to be implemented (Moore or Mealy).
2) Draw the state transition diagram from word description problem.
3) Draw the next state table from the given information.
4) The number of states may be reduced by state reduction method if necessary.
5) Assign the states with binary value.
6) Choose the type of flipflop to be used, determine number of flipflops to be used based on states, and derive the excitation table based on present state (Q_n) to next state (Q_{n+1}) transition and output table.
7) Using K-map, simplify each input of flipflop and circuit output.
8) Draw the logic diagram.

EXAMPLES:

1. **A synchronous machine has one bit serial input 'X'. The output 'Z' of a machine is to be set high when the input contains the message '100'. Draw the state diagram, derive the transition table (state table), excitation table, and design the circuit diagram.**

⇒ **Using Moore Model**



Step 1: We choose Moore model

Step 2: State diagram

Circuit is initialized with state a.

If X = 1, then it moves to state b otherwise (if X = 0), then it remains in same state a.

At state b, if X = 1, it remains in same state b because state b is the state which have detected 1. But if X = 0, the first two pattern is detected so moves to state c.

At state c, if X = 0, then the input stream is 100 and circuit goes to state d with output = 1. But if X = 1, it moves to state b because state b has detected 1.

At state d, if the circuit continues sequence detection job, receiving X = 1, it goes to state b. If X= 0, the circuit goes to initial state a signifying not a single bit has been detected.

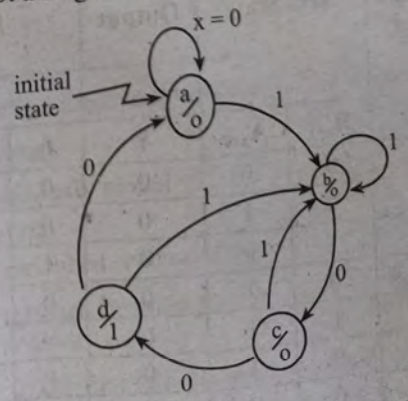


Fig.: State transition diagram of sequence detector.

Step 3: Next state table (transition table)

Previous State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	b	0	0
c	d	b	0	0
d	a	b	1	1

Step 4: No reduction required as there is no repetition.

Step 5: Binary assignment

Let a = 00, b = 01, c = 10, d = 11

Step 6: No. of flipflops = 2 because only 4 states are used.

Let's design with two JK- flipflops i.e., J_A K_A and J_B K_B.

Present State		Next State				Output Y		Excitation table							
		X=0		X=1		X= 0	X = 1	X = 0				X = 1			
B_n	A_n	B_{n+1}	A_{n+1}	B_{n+1}	A_{n+1}			J_B	K_B	J_A	K_A	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	0	0	×	0	×	0	×	1	×
0	1	1	0	0	1	0	0	1	×	×	1	0	×	×	0
1	0	1	1	0	1	0	0	×	0	1	×	×	1	1	×
1	1	0	0	0	1	1	1	×	1	×	1	×	1	×	0

OR

Present State		Present Input	Next State		Output	FF Excitation ($B_n \to B_{n+1}$) ($A_n \to A_{n+1}$)			
B_n	A_n	X	B_{n+1}	A_{n+1}	Y	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	0	×	1	×
0	1	0	1	0	0	1	×	×	1
0	1	1	0	1	0	0	×	×	0
1	0	0	1	1	0	×	0	1	×
1	0	1	0	1	0	×	1	1	×
1	1	0	0	0	1	×	1	×	1
1	1	1	0	1	1	×	1	×	0

Step 7: K-mapping for each input of flipflop (i.e., J_B, K_B, J_A, K_A) respective to present state and present input (i.e., B_n, A_n, X). Also, we plot K-map for output Y.

For J_B

B_n \ A_nX	00	01	11	10
0	0	0	0	1
1	×	×	×	×

$J_B = A_n\overline{X}$

For K_B

B_n \ A_nX	00	01	11	10
0	×	×	×	×
1	0	1	1	1

$K_B = X + A_n$

For J_A

B_n \ A_nX	00	01	11	10
0	0	1	×	×
1	1	1	×	×

$J_A = B_n + X$

For K_A

B_n \ A_nX	00	01	11	10
0	×	×	×	1
1	×	×	0	1

$K_A = \overline{X}$

Step 8: Logic diagram

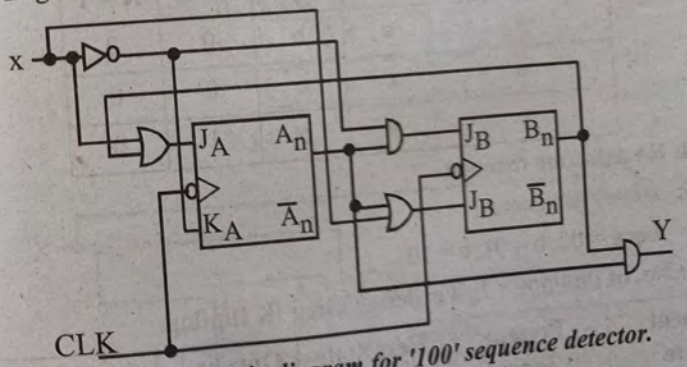


Fig.: Logic diagram for '100' sequence detector.

Using Mealy Model

Step 1: We choose Mealy model

Step 2: State diagram

Circuit is initialized with state a.

If X = 0, then it remains in same state a. If X = 1, then it moves to state b (because first bit is detected.)

At state b, if X = 0, the first two pattern is detected, so moves to state c. But if X = 1, it remains in same state b because state b is the state which have detected 1.

At state c, if X = 0, the input stream is 100, so the output is high and it goes to state a since it has to detect another 1 from starting bit of sequence 100. But if X = 1, it moves to state b because state b has detected 1.

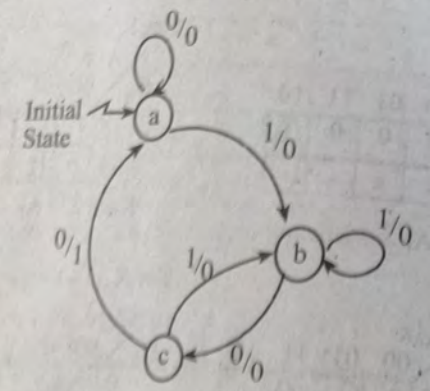


Fig.: State transition diagram

Step 3: Next state table (transition table)

Previous State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	b	0	0
c	a	b	1	0

Step 4: No reduction required.

Step 5: Binary assignment

Let a = 00, b = 01, c = 10

Step 6: No. of flipflops = 2. We design using JK flipflop.

Present State		Present Input	Next State		Output	Flipflop Excitation			
B_n	A_n	X	B_{n+1}	A_{n+1}	X	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	0	×	1	×
0	1	0	1	0	0	1	×	×	1
0	1	1	0	1	0	0	×	×	0
1	0	0	0	0	1	×	1	0	×
1	0	1	0	1	0	×	1	1	×

Here, 11 is the unused table, so values corresponding to it are don't care conditions.

Step 7:

For J_A

B_n \ A_n	00	01	11	10
0	0	0	0	1
1	×	×	×	×

$J_B = A_n\overline{X}$

For K_B

B_n \ A_n	00	01	11	10
0	×	×	×	×
1	1	1	×	×

$K_B = 1$

For J_A

B_n \ An x	00	01	11	10
0	0	1	×	×
1	0	1	×	×

$J_A = X$

For Y_1

B_n \ An x	00	01	11	10
0	0	0	0	0
1	0	1	×	×

$Y = BnX$

For K_A

B_n \ An x	00	01	11	10
0	×	×	0	1
1	×	×	×	×

$K_A = \overline{X}$

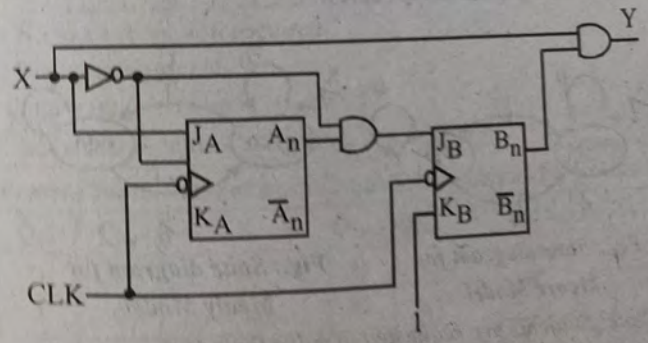


Fig.: Logic diagram of '100' sequence detector

2. Design a sequential detector that receives binary data stream at its input 'X' and signals when a combination '011' arrives at the input by making its output 'Y' high which otherwise remains low. Consider data is coming from left i.e., the first bit to be identified is 1, second 1, and third 0 from the input sequence.

⇒ The state diagram is shown below.

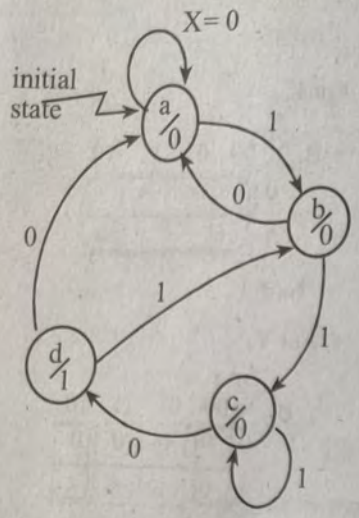


Fig.: State diagram of 011 sequence detector using Moore model

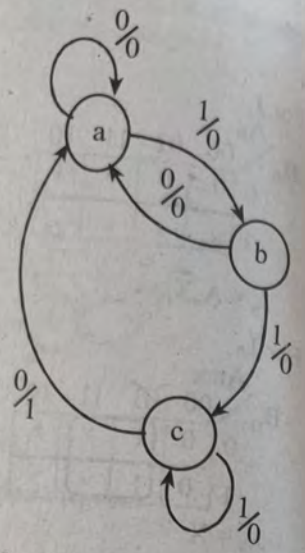


Fig.: State diagram of 011 sequence detector using Mealy model

Note: Students are requested to solve their own.

3. **Design a sequential machine which has single output Y such that it is set high if the input sequence 'X' contains odd no. of 1's otherwise low.**

⇒ The state diagram is shown below.

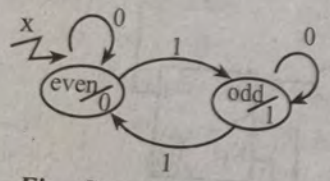


Fig.: State diagram for Moore Model

Fig.: State diagram for Mealy Model

Note: Students are requested to solve their own.

4. **Consider a machine that has a single input 'X' and the clock, and two outputs A and B. On consecutive rising edge of the clock, the code on A and B changes from 00 to 01 to 10 to 11 and repeats itself when 'X' is ENABLED. If any time 'X' is DISABLED, this machine is supposed to hold its present state. Design the sequential machine.**

⇒ The state diagrams using both models are shown below.

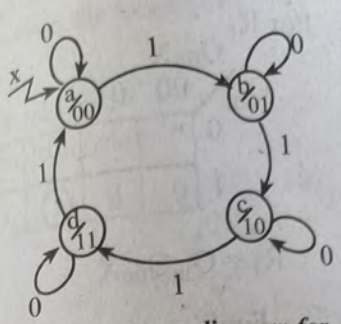


Fig.: State diagram for Moore Model

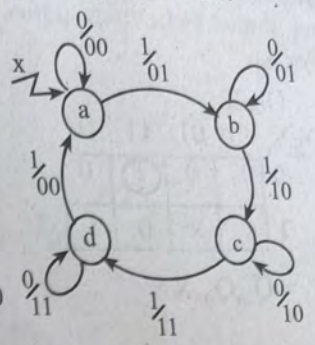


Fig.: State diagram for Mealy Model

Let's design the circuit from Mealy model.

Step 3: State table (transition table)

Present State	Next State		Output			
	X = 0	X = 1	X = 0		X = 1	
			A	B	A	B
a	a	b	0	0	0	1
b	b	c	0	1	1	0
c	c	d	1	0	1	1
d	d	a	1	1	0	0

Step 4: No more reduction required.

Step 5: Binary assignment

Let a = 00, b = 01, c = 10, d = 11

Step 6: Let's design with RS–flipflop, No. of flipflops = 2

Present State		Present input	Next State		Output		FF Excitation			
Q_{1n}	Q_{0n}	X	Q_{1n+1}	Q_{0n+1}	A	B	S_1	R_1	S_0	R_0
0	0	0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	1	0	×	1	0
0	1	0	0	1	0	1	0	×	×	0
0	1	1	1	0	1	0	1	0	0	1
1	0	0	1	0	1	0	×	0	0	×
1	0	1	1	1	1	1	×	0	1	0
1	1	0	1	1	1	1	×	0	×	0
1	1	1	0	0	0	0	0	1	0	1

Step 7:

For S_1

Q_{1n} \ $Q_{0n}X$	00	01	11	10
0	0	0	1	0
1	×	×	0	×

$S_1 = \overline{Q}_{1n} Q_{0n} X$

For R_1

Q_{1n} \ $Q_{0n}X$	00	01	11	10
0	×	×	0	×
1	0	0	1	0

$R_1 = Q_{1n} Q_{0n} X$

For A(o/p)

Q_{1n} \ $Q_{0n}X$	00	01	11	10
0	0	0	1	0
1	1	1	0	1

$A = Q_{1n}\overline{Q}_{0n} + Q_{1n}\overline{X} + \overline{Q}_{1n} Q_{0n} X$

For S_0

Q_{1n} \ $Q_{0n}X$	00	01	11	10
0	0	1	0	×
1	0	1	0	×

$S_0 = \overline{Q}_{0n} X$

For R_0

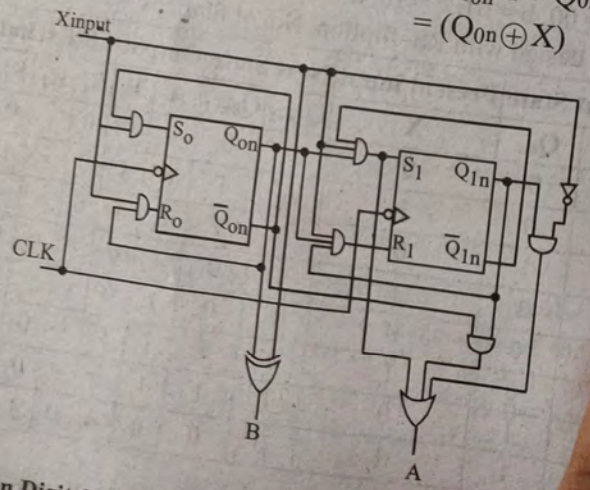
Q_{1n} \ $Q_{0n}X$	00	01	11	10
0	×	0	1	0
1	×	0	1	0

$R_0 = Q_{0n} X$

For B (o/p)

Q_{1n} \ $Q_{0n}X$	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$B = \overline{Q}_{0n} X + Q_{0n} \overline{X}$
$= (Q_{0n} \oplus X)$



5. A machine has two serial inputs X_1 and X_2 and one output Z. The machine is required to give an output Z = 1 when both X_1 and X_2 contain the message 011.

⇒

Step 1: Mealy model

Step 2: State diagram

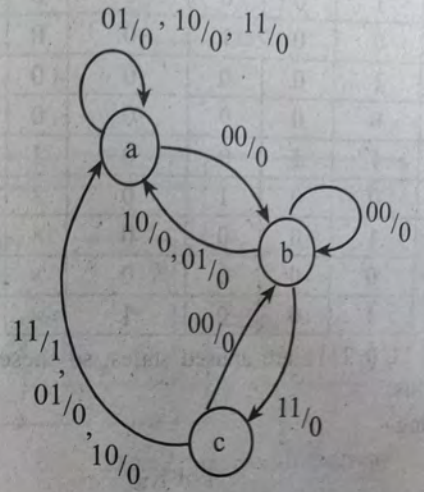


Fig.: State transition diagram for Mealy model

Step 3: Next state table

Present State	Next State				Present Output (Z)			
	X_1X_2 00	X_1X_2 01	X_1X_2 10	X_1X_2 11	X_1X_2 00	X_1X_2 01	X_1X_2 10	X_1X_2 11
a	b	a	a	a	0	0	0	0
b	b	a	a	c	0	0	0	0
c	b	a	a	a	0	0	0	1

Step 5: Binary assignment for states

a = 00, b = 01, c = 10

Step 6: No. of flipflops = 2

Let's design the machine from JK-flipflop.

Let A be o/p of FF J_A K_A &B be o/p of J_B K_B

Present State		Present Input		Next State		Present output	Flipflop Excitation			
B_n	A_n	X_1	X_2	B_{n+1}	A_{n+1}	Z	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	0	×	1	×
0	0	0	1	0	0	0	0	×	0	×
0	0	1	0	0	0	0	0	×	0	×
0	0	1	1	0	0	0	0	×	0	×
0	1	0	0	0	1	0	0	×	×	0
0	1	0	1	0	0	0	0	×	×	1
0	1	1	0	0	0	0	0	×	×	1
0	1	1	1	1	0	0	1	×	×	1
1	0	0	0	0	1	0	×	1	1	×
1	0	0	1	0	0	0	×	1	0	×
1	0	1	0	0	0	0	×	1	0	×
1	0	1	1	0	0	1	×	1	0	×

Here, 1100, 1101 1110, 1111 are unused states, so these are considered as don't care conditions.

Step 7: K-mapping

For J_B

B_nA_n \ X_1X_2	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	×	×	×	×
10	×	×	×	×

$J_B = A_n X_1 X_2$

For K_B

B_nA_n \ X_1X_2	00	01	11	10
00	×	×	×	×
01	×	×	×	×
11	×	×	×	×
10	1	1	1	1

$K_B = 1$

For o/p Z

B_nA_n \ X_1X_2	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	×	×	×	×
10	0	0	1	0

$Z = B_n X_1 X_2$

For J_A

B_nA_n \ X_1X_2	00	01	11	10
00	1	0	0	0
01	×	×	×	×
11	×	×	×	×
10	1	0	0	0

$J_A = \overline{X}_1 \cdot \overline{X}_2$

For K_A

B_nA_n \ X_1X_2	00	01	11	10
00	×	×	×	×
01	0	1	1	1
11	×	×	×	×
10	×	×	×	×

$K_A = X_2 + X_1$

Step 8:

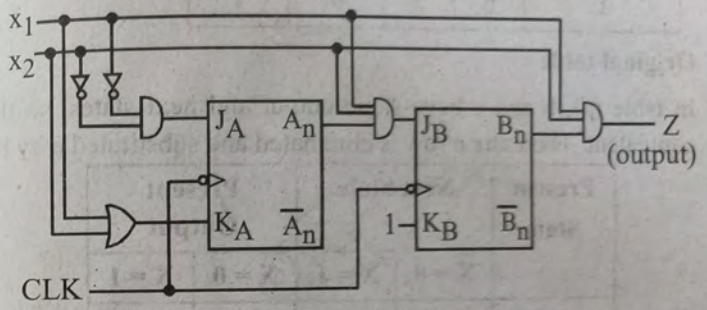


Fig.: Logic circuit diagram

State Reduction Technique

Redundant States

Two states are equivalent if they have same next state and output conditions. If they are equivalent, one of them is redundant i.e. can be eliminated or replaced. The necessary condition for equivalent is "output must be same".

Row elimination method

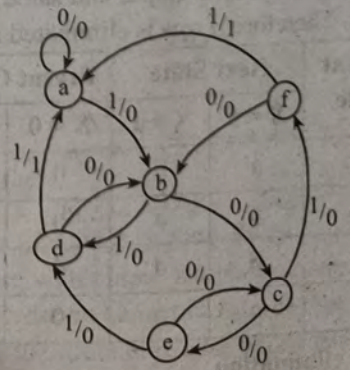


Fig.: State diagram to be reduced

Let's suppose the sequential machine state diagram as follows:

Present State	Next State		Present Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	e	f	0	0
d	b	a	0	1
e	c	d	0	0
f	b	a	0	1

a. **Original table**

In table (a), b and e have same output and next states, so they are equivalent. Therefore e row is eliminated and substituted e by b.

Present State	Next State		Present Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	b	f	0	0
d	b	a	0	1
f	b	a	0	1

b. **After one row elimination**

d and f states also have same output and same next states, so they are also equivalent. Therefore, f row is eliminated and substituted f by d.

Present State	Next State		Presnt Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	b	d	0	0
d	b	a	0	1

c. **After two row elimination**

Here, for X = 0, b → c and c → b, the states c and d are equiv

Present State	Next State		Present O/P	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	b	d	0	0
d	b	a	0	1

d. **Final reduced table after three row elimination**

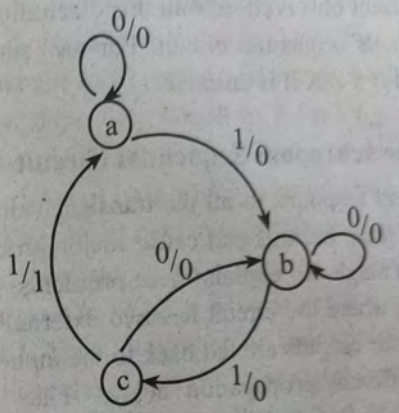


Fig.: Reduced state diagram

9.3 Asynchronous Sequential Circuit Design

Asynchronous sequential circuit, also called "event driven circuit" does not have any clock to trigger change of state.

Example:

AND Gate

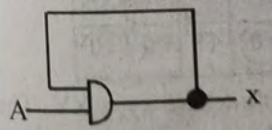


Fig.: AND gate

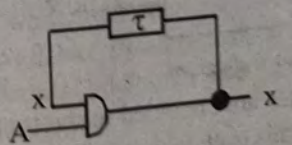


Fig.: AND gate with propagation delay

x \ A	0	1
0	0	0
1	0	1

X = x.A

The two input AND gate with output fed back as one input is shown in **fig.** (a). The circuit can be redrawn shown in **fig.** (b) that includes propagation delay of gate where τ is the finite time after which gate reacts to its input. In the truth table shown above, encircled states indicate the stable condition of the circuit.

For example, if A = 0 and X = 0, then, X = x.A = 0.0 = 0 after time T = τ, x takes the value of X i.e., 0 and because of that output X does not change. Thus, X = 0, A = 0 represents a stable state and is encircled. Similarly x = 0, A = 1 position and x = 1, A = 1 positions are also stable as in each of these cases X = x and no change in output is necessary. But at x = 1, A = 0, X becomes 0 and after time τ, x becomes 0 i.e., we move up by one position in Karnaugh map to x = 0, A = 0 position.

We make an important observation from this discussion which is universally true for asynchronous sequential circuit. For any state, if x = X then the circuit is stable and if x ≠ X it is unstable.

Problems with Asynchronous Sequential Circuit

Asynchronous circuit responds to all the transient values and problems like oscillation, critical race, hazards can cause major problems unless they are addressed at design stage. To explain these problems, we take help of truth table shown below, where the circuit has two external inputs A, B and two outputs X, Y. Both the outputs are fed back to the input side in the form of x and y but with different propagation delays. Thus x, y cannot change simultaneously but with time delays τ_1, and τ_2 respectively and we can write

$x = X(t-\tau_1)$

$y = Y(t-\tau_2)$

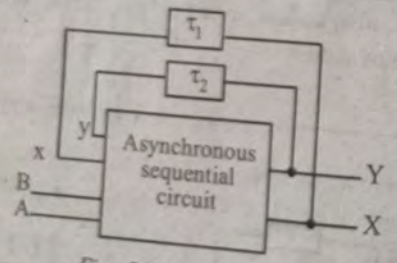


Fig.: Block diagram

xy \ AB	00	01	11	10
00	(00)	11	(00)	01
01	00	(01)	00	00
11	(11)	(11)	00	(11)
10	(10)	11	00	(10)

The stable states are encircled in the truth table where xy = XY. The major problems with this truth table are discussed below.

i) Oscillation

Consider the stable state xyAB = 0000, where x = X and y = Y. If input AB changes from 00 to 10, the circuit goes to xyAB = 0010 state and then output XY = 01. This is a transient state because xy ≠ XY. After time τ_2, y takes the value of Y = 1 and the circuit goes to xyAB = 0110 where output XY = 00. This is again a transient state and after another propagation delay of τ_2, the circuit goes to xyAB = 0010. Thus, the circuit oscillates between state 0010 and 0110, and the output Y oscillates between 0 and 1 with a gap τ_2. In asynchronous sequential circuits for any given input, transitions between two unstable states like these are to be avoided to remove oscillation.

ii) Critical race

This occurs when an input tries to modify more than one output. In the truth table shown above, consider the stable state xyAB = 0000. Now, if AB changes to 01, the circuit moves to xyAB = 0001 where XY = 11. Now, depending which of τ_1 and τ_2 is lower, xy moves from 00 to either 01 or 10. If τ_1 is lower, x changes earlier and the circuit goes to xyAB = 1001 which is an unstable state with XY = 11 and xy ≠ XY. The circuit next moves to state xyAB = 1101 which is a stable state and final output XY = 11. If τ_2 is lower, y changes earlier and the circuit goes to xyAB = 0101 which is a stable state and final output XY = 01. Thus, depending on propagation delays in feedback path, the circuit settles at two different states generating two different set of output. Such situation is called critical race condition and is to be avoided in asynchronous sequential circuit.

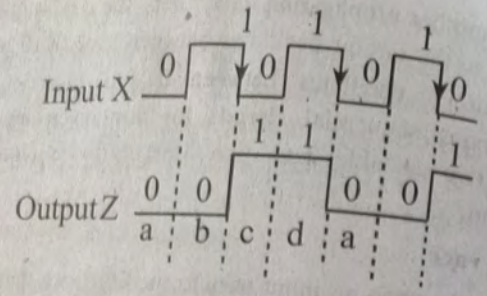
iii) Hazards:

Static and dynamic hazards cause malfunctioning of asynchronous sequential circuit. Situations like Y = A+A or Y = A.A are to be avoided for any input output combination with the help of hazards covers in truth table. In circuit with feedback, even when those hazards are adequately covered there can be another problem called "essential hazard". This occurs when change in input doesn't reach one part of the circuit while from other part one output fed back to the input side becomes available. Essential hazard is avoided by adding delay, may be in the form of additional gates that does not change the logic level in the feedback path.

EXAMPLES

1. Design frequency divide by 2 counter from asynchronous machine.

⇒ The waveform for the divide 2 counter is



State diagram:

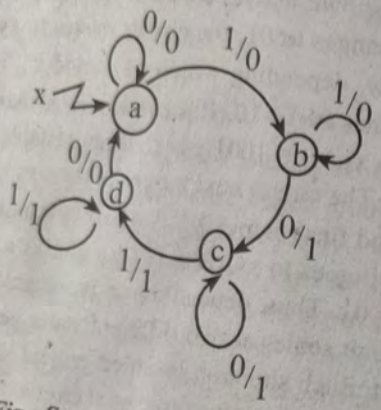


Fig.: State diagram using Mealy model

Primitive flow table (or simply flow table):

Present State	Present Input (X)	Next state	Output
a	0	a	0
a	1	b	0
b	0	c	1
b	1	b	0
c	0	c	1
c	1	d	1
d	0	a	0
d	1	d	1

Binary assignment:

To avoid uncertainty change of secondary variable, we assign binary value so that only one bit changes.

i.e., a = 00, b = 01, c = 11, d = 10

Excitation table:

Present State		Present Input	Next state		Output
y_1	y_2	X	Y_1	Y_2	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	0	1

For asynchronous machine design 'Next State' is the excitation table. So, plot K-map for Y_1 and Y_2.

For Y_1

y_1 \ y_2X	00	01	11	10
0	0	0	0	1
1	0	1	1	1

$$y_1 = y_1x + y_1\bar{y}_2 + y_2\bar{x}$$

For Y_2

y_1 \ y_2X	00	01	11	10
0	0	1	1	1
1	0	0	0	1

$$y_2 = \bar{y}_1x + \bar{y}_1y_2 + y_2\bar{x}$$

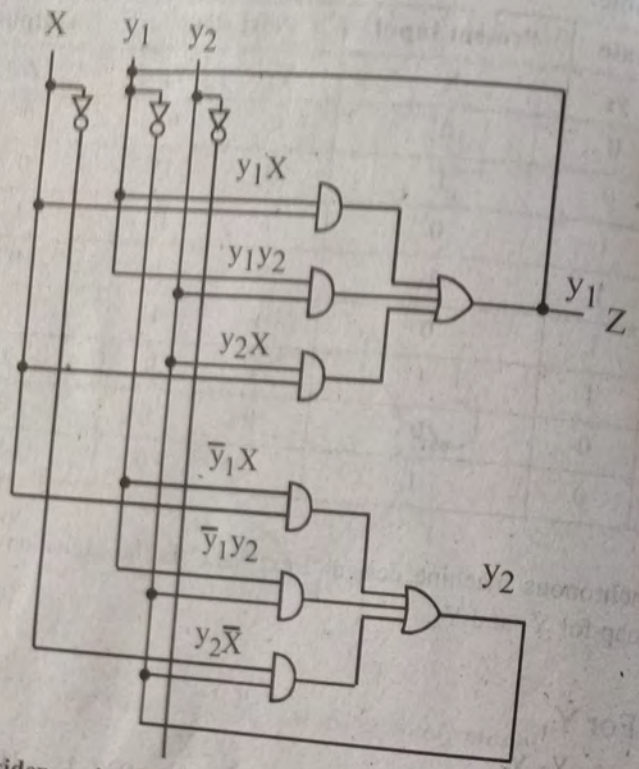
For Z

y_1 \ y_2X	00	01	11	10
0	0	0	0	1
1	0	1	1	1

→ to avoid hazard

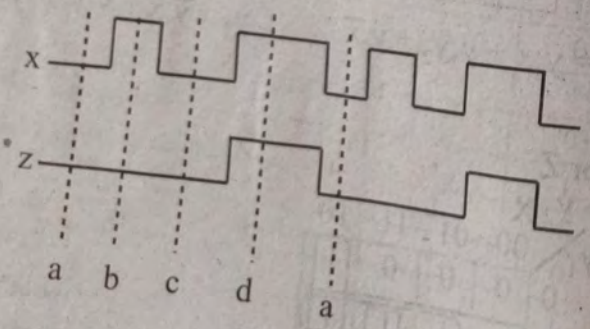
$$Z = y_1 \times y_1y_2 + y_2\bar{x} = y_1$$

Logic diagram:



2. Consider a circuit with one input (X) and one output (Z). The output should suppress every alternative pulse on the input starting with the first.

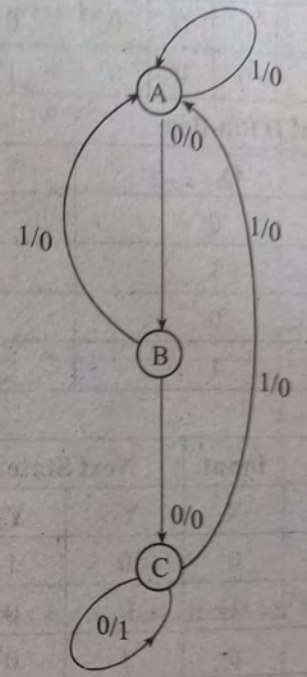
⇒



MORE WORKED OUT EXAMPLES:

1. Design a sequential machine that detects three consecutive zeroes from an input data stream X by making output Y =1.[2069 Chaitra]

⇨



Transition table

Present state	Next state		Output	
	At X = 0	At X = 1		
X_1 X_2	Y_1	Y_2	At X = 0	At X = 1
A	B	A	0	0
B	C	A	0	0
C	C	A	1	0

Assigning the binary values

A → 00

B → 01

C → 10

Present state		Next state (Y_1Y_2)				Output	
X_1	X_2	At $X=0$		At $X=1$		At $X=0$	At $X=1$
0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0
1	0	1	0	0	0	1	0

Excitation table of D flip-flop

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Present state		Input	Next State		D_A	D_B
X_1	X_2	X	Y_1	Y_2		
0	0	0	0	1	0	1
0	0	1	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	1	0	1	0
1	0	1	0	0	0	0
1	1	0	×	×	×	×
1	1	1	×	×	×	×

For D_A

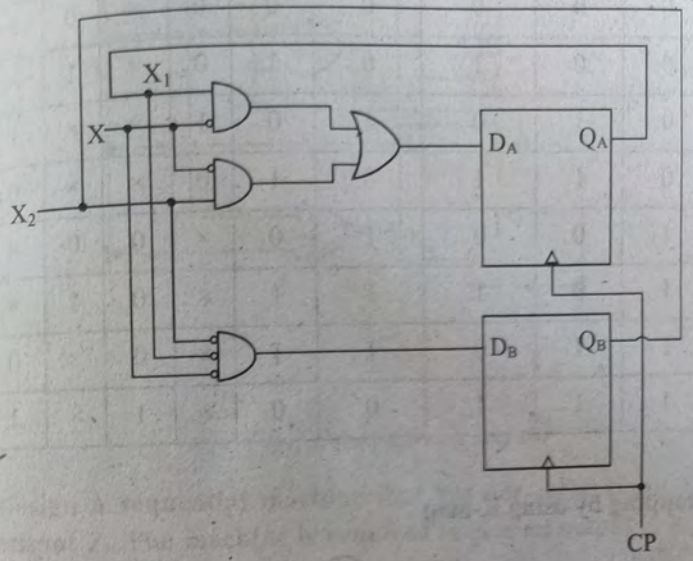
X_1 \ X_2X	00	01	11	10
0	0	0	0	1
1	1	0	×	×

$D_A = X_1X_2' + X_2'X'$

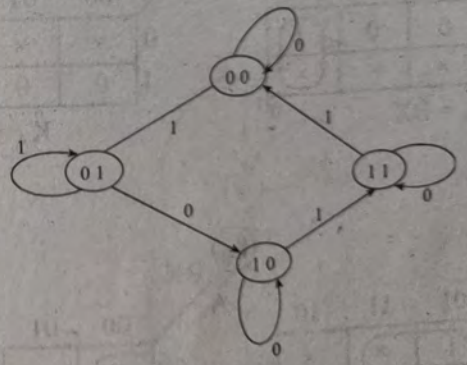
For D_B

X_1 \ X_2X	00	01	11	10
0	1	0	0	0
1	0	0	×	×

$D_B = X_1'X_2'X'$



2. **Design synchronous sequential circuit for the given state diagram using JK flip-flop.** [2069 Ashadh]



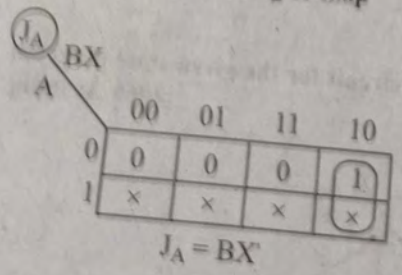
⇒ **Excitation table for JK flip-flop**

Q_t	Q_{t-1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

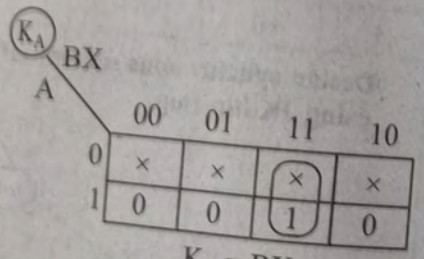
State table:

Present state		Input control	Next state		J_A	K_A	J_B	K_B
A	B	X	C	D				
0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	×	1	×
0	1	0	1	0	1	×	×	1
0	1	1	0	1	0	×	×	0
1	0	0	1	0	×	0	0	×
1	0	1	1	1	×	0	1	×
1	1	0	1	1	×	0	×	0
1	1	1	0	0	×	1	×	1

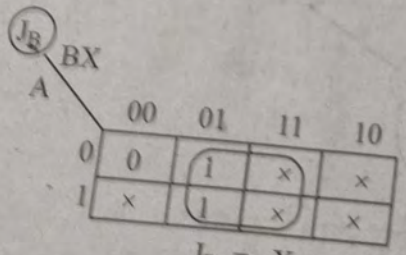
Now, mapping by using K-map



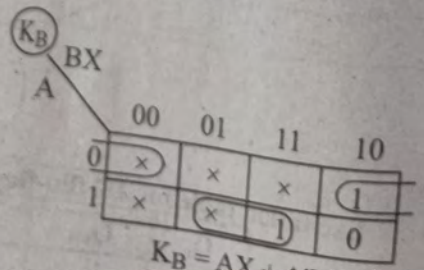
$J_A = BX'$



$K_A = BX$



$J_B = X$



$K_B = AX + A'X'$

$= A \odot X$

Circuit diagram

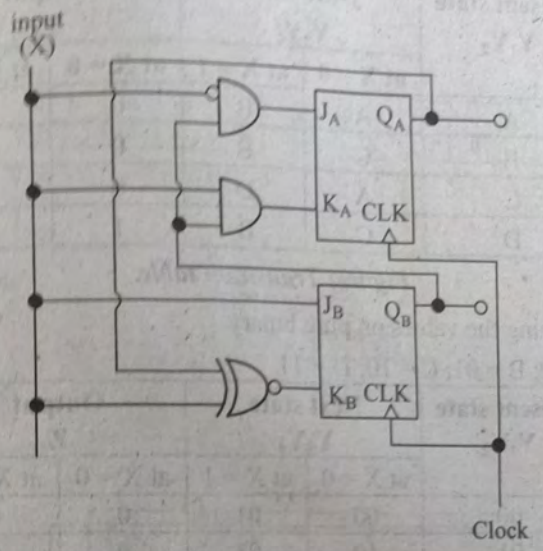


Fig.: Detail Circuit Layout

3. **Design a sequential machine that has one serial input and one output Z. The machine is required to give an output Z = 1 when the input X contains the message 1010.** **[2068 Chaitra]**

⇒ Here input X = 1010, Output Z = 0001

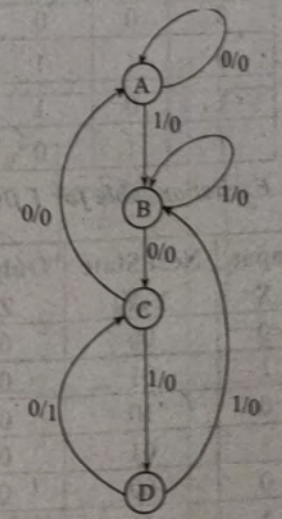


Fig.: State diagram

Present state Y_1Y_2	Next state Y_1Y_2		Output Z	
	at X = 0	at X = 1	at X = 0	at X = 1
A	A	B	0	0
B	C	B	0	0
C	A	D	0	0
D	C	B	1	0

Figure: Transition table.

Assigning the values on pure binary

A = 00; B = 01; C = 10; D = 11

Present state Y_1Y_2	Next state Y_3Y_4		Output Z	
	at X = 0	at X = 1	at X = 0	at X = 1
00	00	01	0	0
01	10	01	0	0
10	00	11	0	0
11	10	01	1	0

Fig.: Pure binary assignment

Now, designing by using T flip–flop

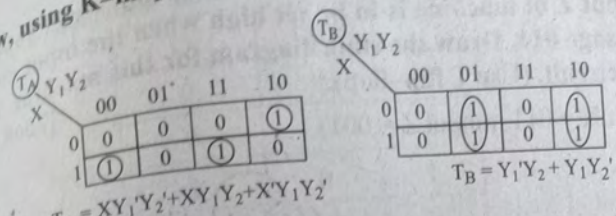
Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

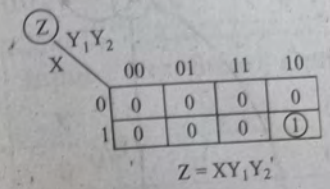
Fig.: Excitation table for T flip-flop

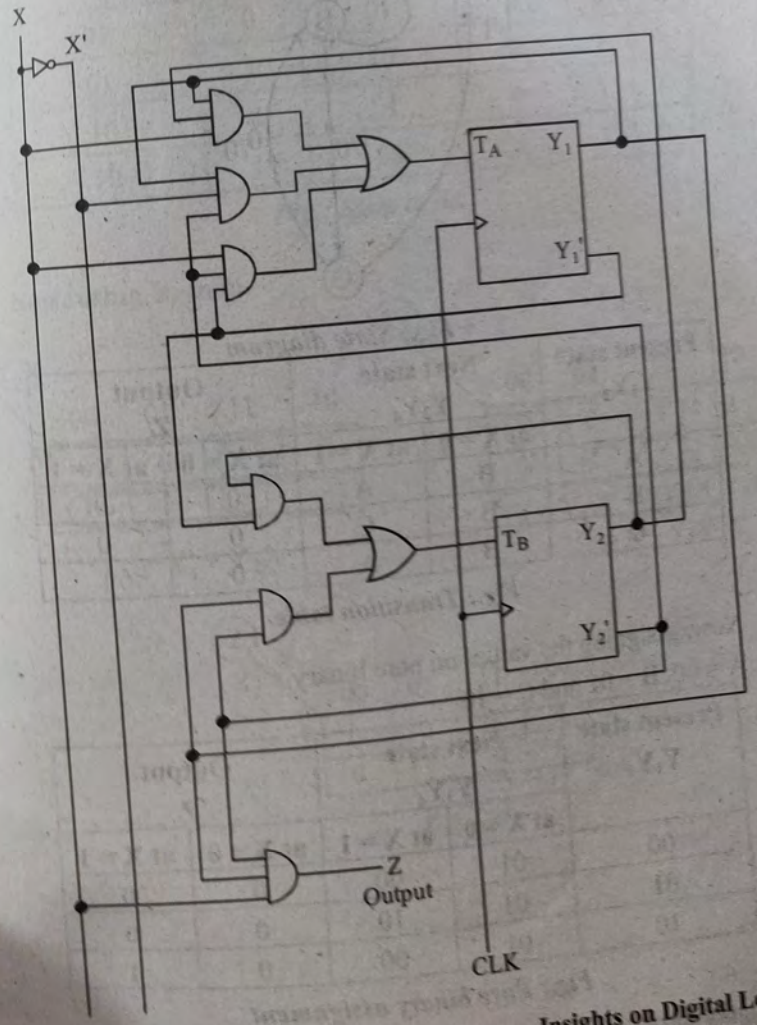
Present State Y_1Y_2	Input X	Next State Y_3Y_4	Output Z	T_A	T_B
00	0	00	0	0	0
00	1	01	0	0	1
01	0	10	0	1	1
01	1	01	0	0	0
10	0	00	0	1	0
10	1	11	0	0	1
11	0	10	1	0	1
11	1	01	0	1	0

Figure: State table

Now, using K–map







4. **A synchronous state machine has one bit serial input X. The output Z of machine is to be set high when the input contains the message 011. Draw the state diagram for this machine and design the circuit. (Use T flip–flop).** **[2068 Shrawan]**

⇒ Input X = 011, output Z = 001

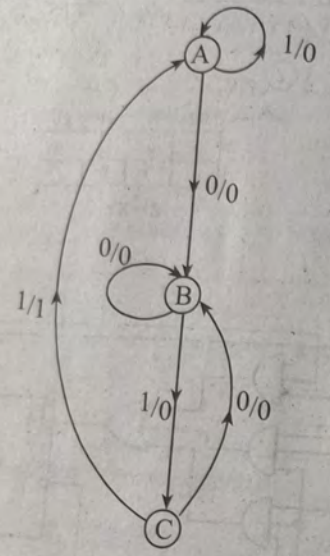


Fig.: State diagram

Present state Y_1Y_2	Next state Y_3Y_4		Output Z	
	at X = 0	at X = 1	at X = 0	at X = 1
A	B	A	0	0
B	B	C	0	0
C	B	A	0	1

Fig.: Transition table

Now, assigning the values on pure binary
A = 00, B = 01 and C = 10

Present state Y_1Y_2	Next state Y_3Y_4		Output Z	
	at X = 0	at X = 1	at X = 0	at X = 1
00	01	00	0	0
01	01	10	0	0
10	01	00	0	1

Fig.: Pure binary assignment

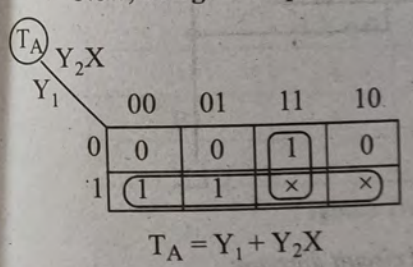
We know, transition table for T flip–flop

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

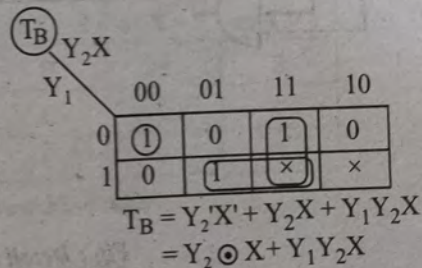
Present State Y_1Y_2	Input X	Next State Y_3Y_4	Output Z	T_A	T_B
00	0	01	0	0	1
00	1	00	0	0	0
01	0	01	0	0	0
01	1	10	0	1	1
10	0	01	0	1	1
10	1	00	1	1	0

Fig.: State table.

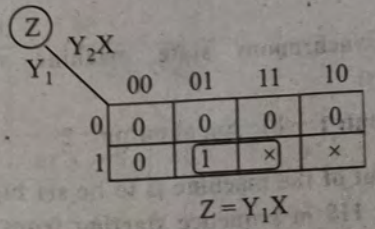
Now, using K–map.



$T_A = Y_1 + Y_2X$



$T_B = Y_2'X' + Y_2X + Y_1Y_2X$
$= Y_2 \odot X + Y_1Y_2X$



$Z = Y_1X$

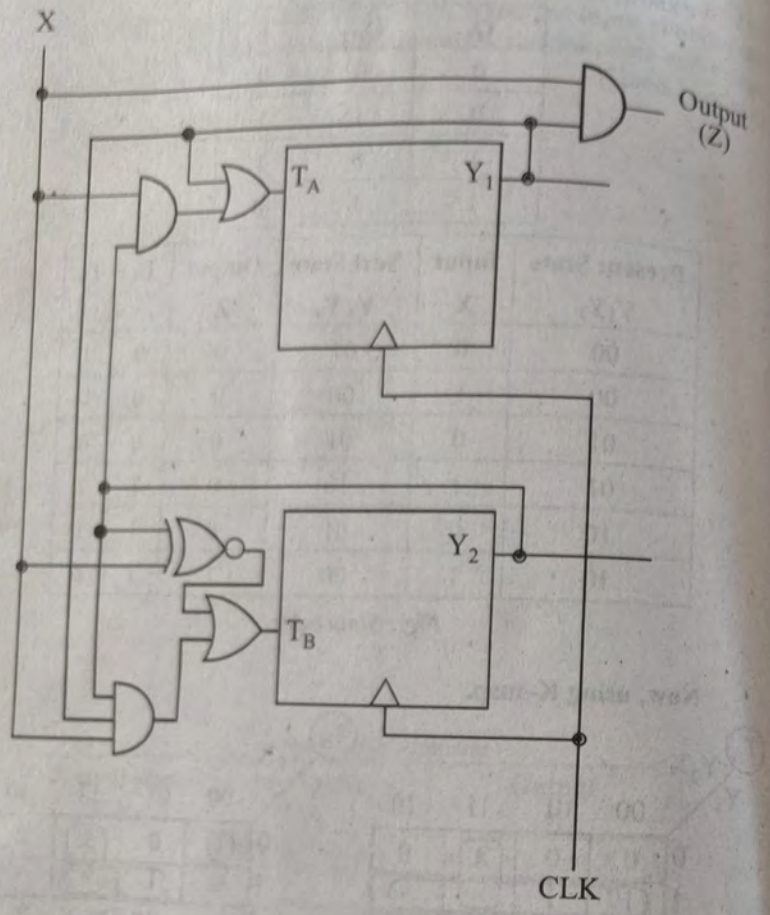


Fig.: Detail circuit diagram

5. Design a synchronous state machine with the following specifications:

a) No. of input: 1 b) No. of output: 2

c) The output of the machine is to be set high when the data in the input 110 in sequence starting from MSB (use SR flip-flop).

[2068 Baishakh]

⇒ *Input X = 110*

Output Z = 001

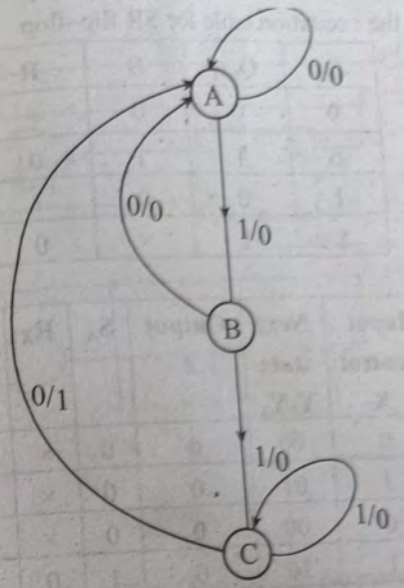


Fig.: State diagram from given parameters

Present state Y_1Y_2	Next state Y_3Y_4		Output Z	
	at X = 0	at X = 1	at X = 0	at X = 1
A	A	B	0	0
B	A	C	0	0
C	A	C	1	0

Figure: Transition table

Now, assigning the values on pure binary

A = 00; B = 01; C = 10

Present state Y_1Y_2	Next state Y_3Y_4		Output Z	
	at X = 0	at X = 1	at X = 0	at X = 1
00	00	01	0	0
01	00	10	0	0
10	00	10	1	0

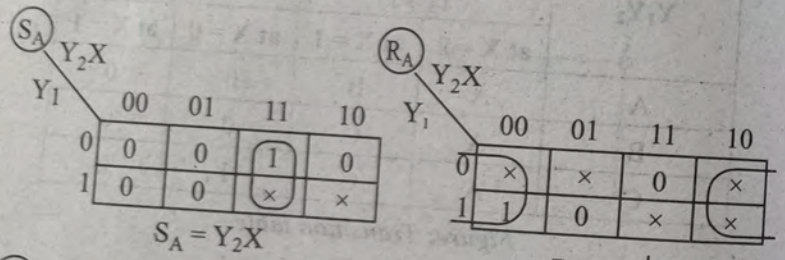
Figure: Pure binary assignment

We know the excitation table for SR flip–flop

Q_t	Q_{t+1}	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

Present state $Y_1 Y_2$	Input control X	Next state $Y_3 Y_4$	Output Z	S_A	R_A	S_B	R_B
00	0	00	0	0	×	0	×
00	1	01	0	0	×	1	0
01	0	00	0	0	×	0	1
01	1	10	0	1	0	0	1
10	0	00	1	0	1	0	×
10	1	10	0	×	0	0	×

Fig.: State table



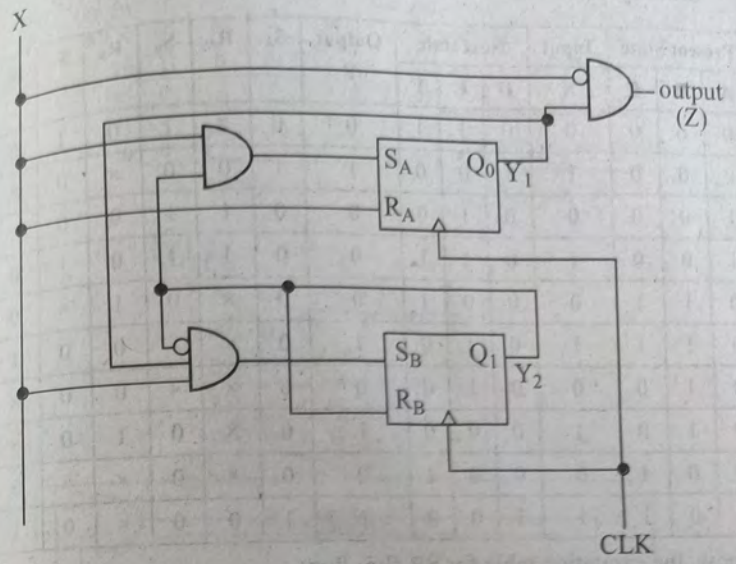
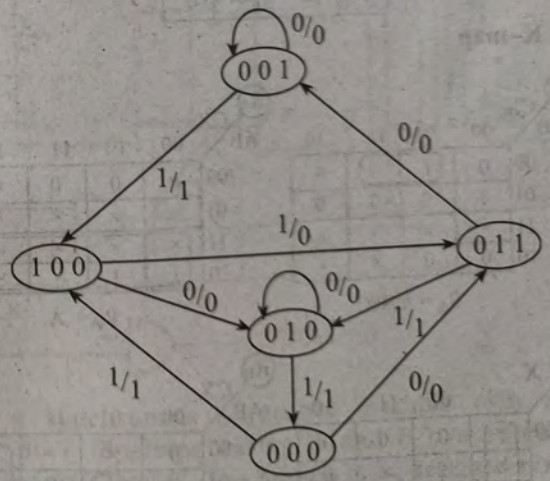


Fig.: Detail logic diagram

6. **A sequential circuit has one input and one output. The state diagram is shown in figure. Design the sequential circuit with RS flip-flops.** **[2067 Ashadh]**



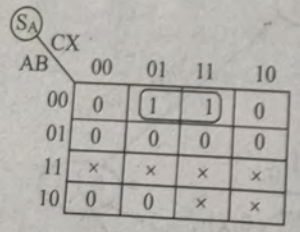
⇒ **State synthesis table:**

Present State			Input	Next state			Output	S_A	R_A	S_B	R_B	S_C	R_C
A	B	C	X	D	E	F	Y						
0	0	0	0	0	1	1	0	0	×	1	0	1	0
0	0	0	1	1	0	0	1	1	0	0	×	0	×
1	0	0	0	0	1	0	0	0	1	1	0	0	×
1	0	0	1	0	1	1	0	0	1	1	0	1	0
0	1	1	0	0	0	1	0	0	×	0	1	×	0
0	1	1	1	0	1	0	1	0	×	×	0	0	1
0	1	0	0	0	1	0	0	0	×	×	0	0	×
0	1	0	1	0	0	0	1	0	×	0	1	0	×
0	0	1	0	0	0	1	0	0	×	0	×	×	0
0	0	1	1	1	0	0	1	1	0	0	×	0	1

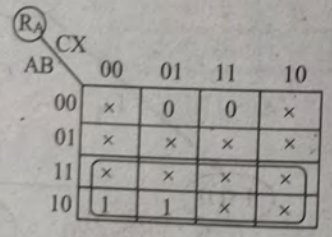
We know, the excitation table for SR flip–flop:

Q	Q_{t+1}	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

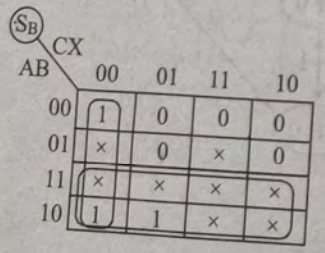
Using K–map



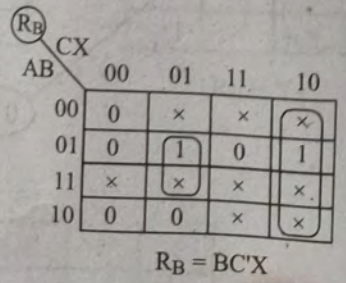
$S_A = A'B'X$



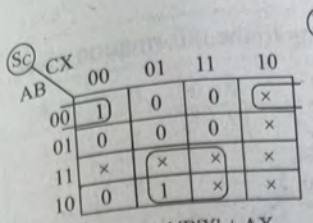
$R_A = A$



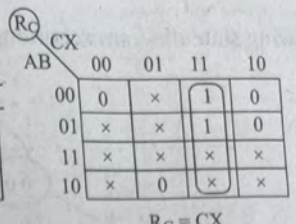
$S_B = A + C'X'$



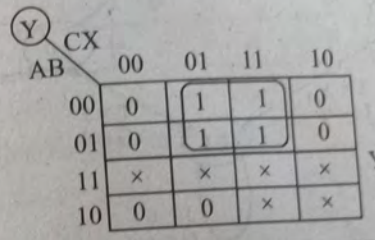
$R_B = BC'X$



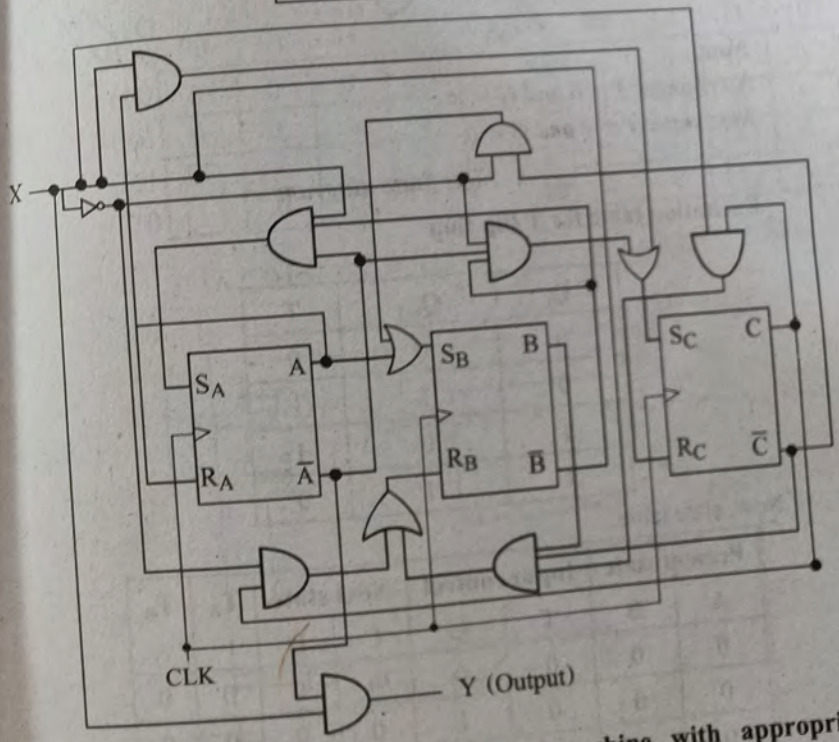
$S_C = A'B'X' + AX$



$R_C = CX$

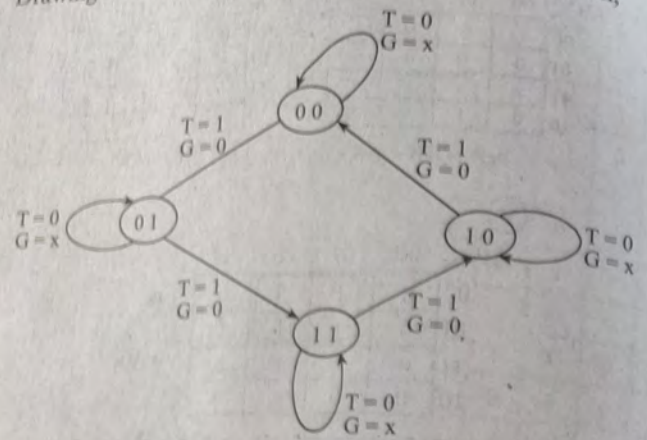


$Y = A'X$



7. **Design a synchronous sequential machine with appropriate number of T flip–flops having A and B two outputs and two control inputs T and G. The System to be designed remains in the same state (i.e., no change) while T = 0 and G = X (don't care). The state machine goes through the state transition from 00 to 10 to 11 to 01 back to 00 and repeats when given control inputs become T = 1 and G = 0.** **[2064 Falgun]**

⇒ Drawing state diagram according to the information given,



Note:
No change: T = 0 and G = ×
Next state: T = 1 and G = 0

Fig.: State diagram

Excitation table for T flip flop

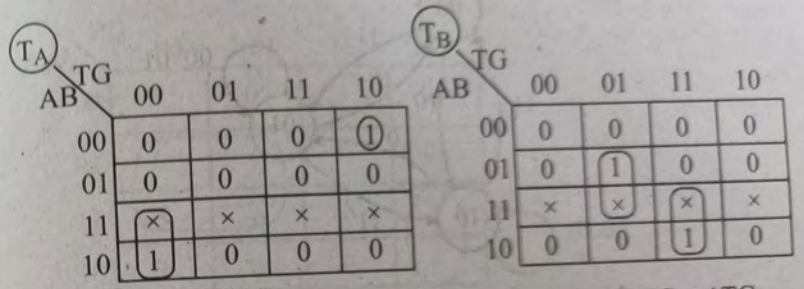
Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Now, state table:

Present state		Input control		Next state		T_A	T_B
A	B	T	G	C	D		
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	1	0	1
1	1	0	0	1	1	0	

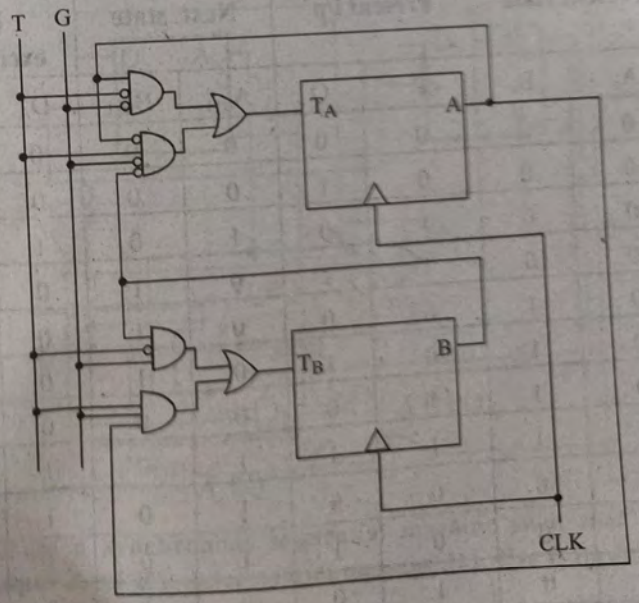
Present state		Input control		Next state		T_A	T_B
A	B	T	G	C	D		
1	1	0	1	1	1	0	0
1	1	1	0	0	1	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	1

Now, Using K–map for T_A and T_B



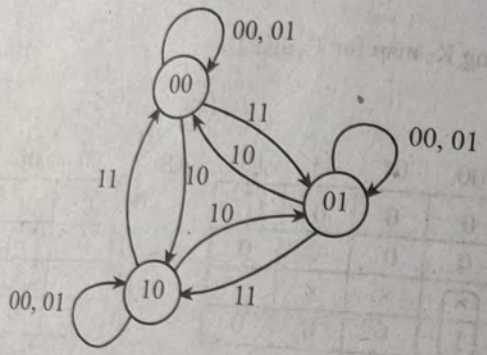
$T_A = AT'G' + AB'TG'$

$T_A = BT'G + ATG$



8. **Design a sequential circuit with two D flip-flops and two inputs, P and Q. If P = 0, the circuit remains in the same state regardless of the value of Q. When P = 1 and Q = 1, the circuit goes through the state transitions from 00 to 01 to 10 back to 00, and repeats. When P = 1 and Q = 0, the circuit goes through the state transitions from 00 to 10 to 01 back to 00, and repeats. The circuit is to be designed by treating the unused state(s) as don't care condition(s).** **[2071 Chaitra]**

⇒



State diagram

Present state		Present i/p		Next state		FF excitation	
A_n	B_n	P	Q	A_{n+1}	B_{n+1}	D_A	D_B
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	1
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	1
0	1	1	0	0	0	0	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	0	0	0

For D_A

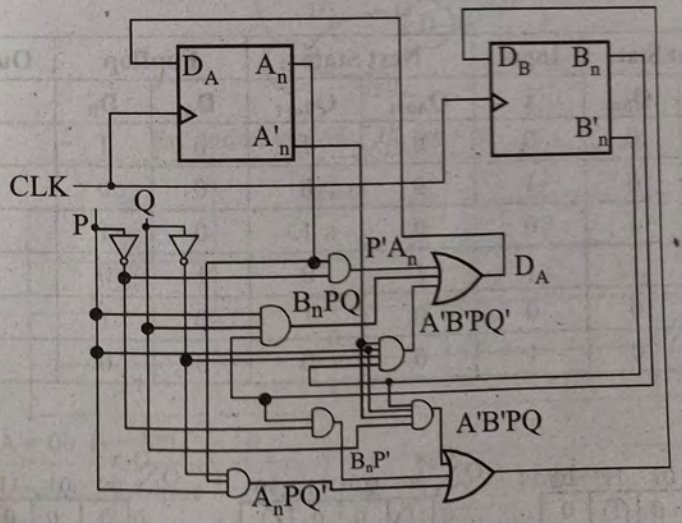
A_nB_n \ PQ	00	01	11	10
00	0	0	0	(1)
01	0	0	1	0
11	X	X	X	X
10	1	1	0	0

$\therefore \quad D_A = A_nP' + B_nPQ + A'_nB'_nPQ'$

For D_B

A_nB_n \ PQ	00	01	11	10
00	0	0	(1)	0
01	1	1	0	0
11	X	X	X	(X)
10	0	0	0	1

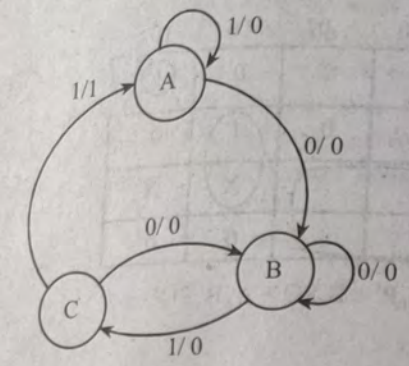
$D_B = B_nP' + A_nPQ' + A'_nB_n'PQ$



Design a synchronous sequential machine such that it gives output Z = 1 if it detects input message 011. Use D flipflop.

[2073 Chaitra]

⇒



Excitation table for D flipflop

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

Let A = 00, B= 01, C = 10

Present State		Input	Next State		Flipflop		Output
Q_{An}	Q_{Bn}	x	Q_{An+1}	Q_{Bn+1}	D_A	D_B	Z
0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	0
0	1	0	0	1	0	1	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	1	0
1	0	1	0	0	0	0	1

For D_A

Q_A \ Q_BX	00	01	11	10
0	0	0	1	0
1	0	0	x	x

$D_A = Q_BX$

For D_B

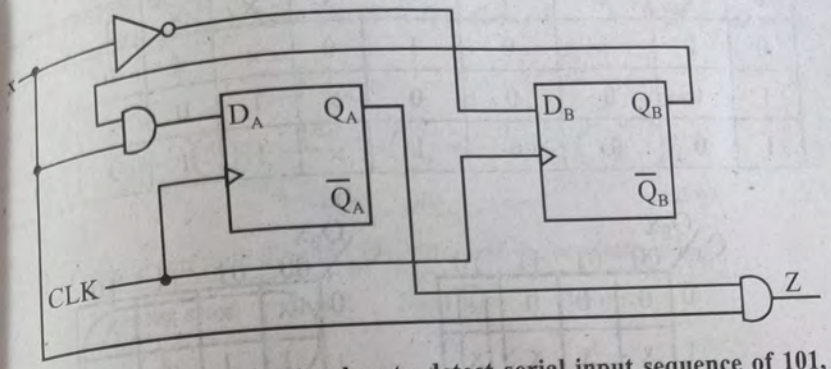
Q_A \ Q_BX	00	01	11	10
0	1	0	0	1
1	1	0	x	x

$D_B = \bar{x}$

For Z

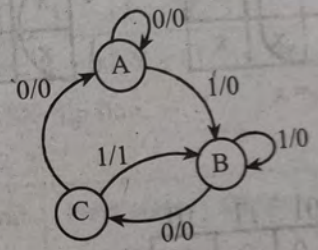
Q_A \ Q_BX	00	01	11	10
0	0	0	0	0
1	0	1	x	x

$Z = Q_AX$



10. **A sequential machine has to detect serial input sequence of 101, for which the machine output will be high. The machine contains two JK flipflops, A and B. Assume: single input, X and single output, Y.** **[2073 Shrawan]**

⇒



Excitation table for JK flip flop

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

Let A = 00, B = 01, C = 10

Present State		Input	Next State		Flipflop				Output
Q_{An}	Q_{Bn}	x	Q_{An+1}	Q_{Bn+1}	J_A	K_A	J_B	K_B	Y
0	0	0	0	0	0	×	0	×	0
0	0	1	0	1	0	×	1	×	0

0	1	0	1	0	1	×	×	1	0
0	1	1	0	1	0	×	×	0	0
1	0	0	0	0	×	1	0	×	0
1	0	1	0	1	×	1	1	×	1

For J_A

Q_A \ Q_BX	00	01	11	10
0	0	0	0	1
1	x	x	x	x

$\therefore J_A = Q_B\bar{x}$

For K_A

Q_A \ Q_BX	00	01	11	10
0	x	x	x	x
1	1	1	x	x

$\therefore K_A = 1$

For J_B

Q_A \ Q_BX	00	01	11	10
0	0	1	x	x
1	0	1	x	x

$J_B = x$

For K_B

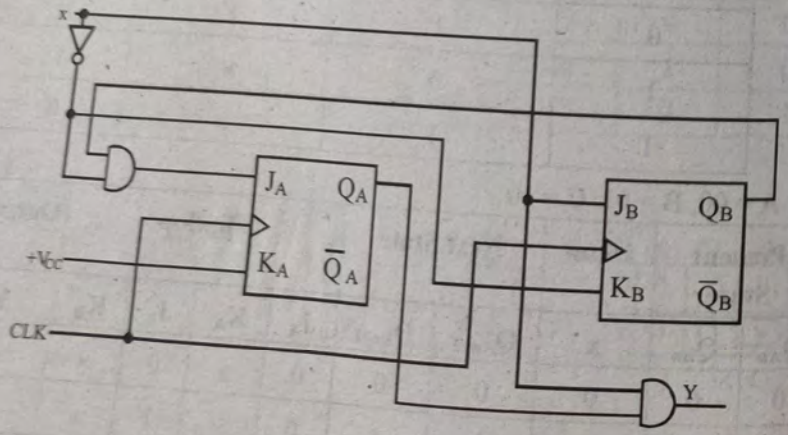
Q_A \ Q_BX	00	01	11	10
0	x	x	0	1
1	x	x	x	x

$K_B = \bar{x}$

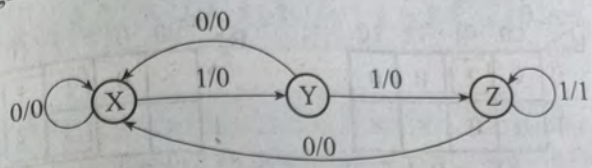
For Y

Q_A \ Q_BX	00	01	11	10
0	0	0	0	0
1	0	1	x	x

$Y = Q_Ax$



11. **Design a synchronous sequenstial machine from the state diagram given below. Use S–R flip–flop.** **[2075 Ashwin]**



⇒ From given state diagram, we will draw the next state table,

Present state	Input	Next state	Output
X	0	X	0
X	1	Y	0
Y	0	X	0
Y	1	Z	0
Z	0	X	0
Z	1	Z	1

Then we will assign X = 00, Y = 01, Z = 10

Let's design with SR-flip flop.

No. of flip-flops = 2

Present state		Input	Next state		Output	Excitation table			
B_n	A_n	C	B_{n+1}	A_{n+1}	D	S_B	R_B	S_A	R_A
0	0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	0	×	1	0
0	1	0	0	0	0	0	×	0	1
0	1	1	1	0	0	1	0	0	1
1	0	0	0	0	0	0	1	0	×
1	0	1	1	0	1	×	0	0	×

For S_B

B_n \ A_nC	00	01	11	10
0	0	0	1	0
1	0	×	×	×

$S_B = A_nC$

For R_B

B_n \ A_nC	00	01	11	10
0	×	×	0	×
1	1	0	×	×

$R_B = \bar{C}$

For S_A

B_n \ A_nC	00	01	11	10
0	0	1	0	0
1	0	0	×	×

$S_A = \bar{B}_n\bar{A}_nC$

For R_A

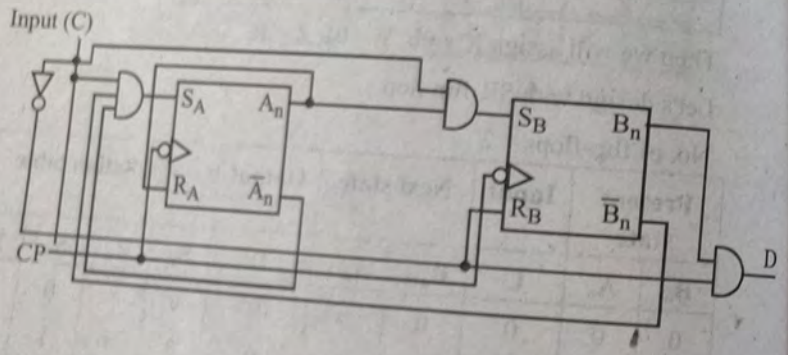
B_n \ A_nC	00	01	11	10
0	×	0	1	1
1	×	×	×	×

$R_A = A_n$

For output (D)

B_n \ A_nC	00	01	11	10
0	0	0	0	0
1	0	1	×	×

$D = B_nC$



Chapter – 10
DIGITAL INTEGRATED CIRCUITS

10.1 Introduction

A digital IC is constructed by an interconnection of resistors, transistors, and perhaps small capacitors, all of which have been formed on the surface of a silicon wafer. The entire circuit resides on a tiny piece of semiconductor material called a chip.

Different types of IC families are listed below:

1. Resist transistor logic (RTL)
2. Diode Transistor logic (DTC)
3. Integrated Injection logic (I^2L)
4. Transistor Transistor logic (TTL)
5. Schottky TTL
6. NMOS
7. CMOS

Characteristics of digital ICs include:

- **Propagation delay (speed of operation)**

  The output of logic gate does not change its state instantaneously in response to the change in state of its input. There is a time delay between these two events, which is called as propagation delay.

- **Power dissipation**

  As a result of applied voltage and current flowing through the logical ICs, some power will be dissipated in it, in the form of heat.

- **Current and voltage parameters**

  These parameters include $V_{IL,\,max}$, $V_{IH,\,min}$, $V_{OH,\,min}$, $V_{OL,\,max}$, I_{IL}, I_{IH}, I_{OH}, I_{OL}.

Noise immunity

It is the measure of how much stray noise voltage the device can handle without giving any error at the output level.

- **Fan-in and Fan-out**

  Fan-in is defined as the number of inputs a gate has. Fan-out is defined as the maximum number of inputs of the same IC family that a gate can drive without falling outside the specified output voltage limits.

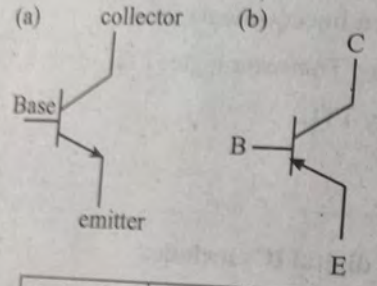
- **Operating temperature**

  The temperature range acceptable for the consumer and industrial applications is 0°C to 70°C and that for the military applications is –55°C to 125°C.

10.2 Switching Circuits

1) Bipolar Junction Transistor (BJT)

Bipolar junction transistor is available in two polarities.



Input (B)	npn	pnp
0V	OFF	ON
5V	ON	OFF

2) Metal Oxide Semiconductor Field Effect Transistor (MOSFET)

There are two polarities of MOSFETs: n-channel MOSFET (NMOS) and p-channel MOSFET (PMOS). They operate either in enhancement mode or depletion mode. The switch is activated by applying a voltage between gate and source.

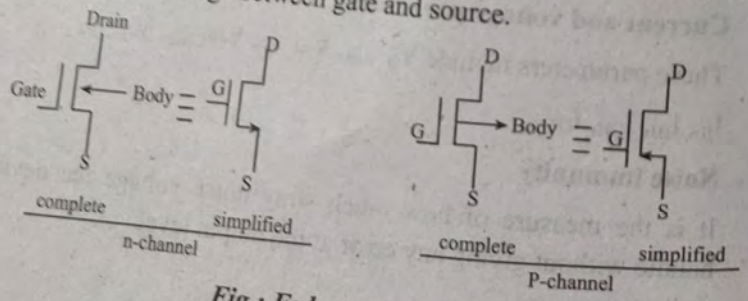


Fig.: Enhancement MOSFET

3) Complementary Metal Oxide Semiconductor (CMOS) Field Effect Transistor

ICs constructed entirely from n-channel MOSFET are called NMOS ICs. ICs constructed entirely from p-channel MOSFET are called PMOS ICs. ICs constructed using both PMOS and NMOS are called CMOS ICs.

10.3 External Drive for TTL Loads

Some of the ways to drive a TTL load are explained as follows:

1. Switch drive

Figure below shows the preferred method for driving a TTL input from a switch. With the switch open, the input is pulled up to +5 V. When the switch is closed, the input is pulled down to ground.

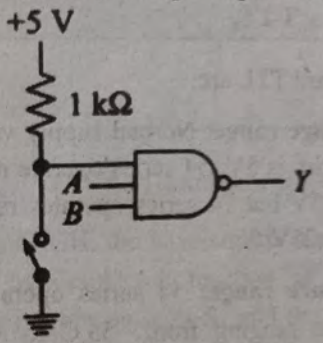


Fig.: Switch drive for TTL input

2. Size of pull-up resistor

The smaller the pull-up resistance, the larger the current drain. On the other hand, too large a pull-up resistance causes the time constant (RC) to be larger which in turn causes speed problems. Pull-up resistances between 1 and 10 KΩ are typical.

3. Transistor drive

In this case, we use a transistor switch to control the state of the TTL input. When V_i is low, the transistor is off and is equivalent to open switch. Then, the TTL input is pulled up to +5 V through a resistance of 1 KΩ. When V_i is high, the transistor is on and is equivalent to a closed switch. In this case, it easily sinks the 1.6 mA of input current.

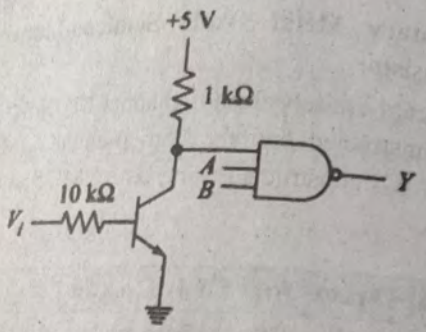


Fig.: Transistor drive for TTL input.

4. **Operational amplifier drive**
5. **Comparator drive**

10.4 74XX Series TTL

Characteristic of standard TTL are:

i) **High voltage range**: Normal supply voltage (V_{CC}) for both 54 and 74 series is 5V. 54 series operate reliably over the range of 4.5V to 5.5V but 74 series operates reliably over the range of 4.75V to 5.25V.

ii) **Temperature range:** 54 series operates properly in ambient temperature ranging from –55°C to +125°C while 74 series operates only in the range from 0°C to 70°C.

iii) **Power dissipation:** A standard TTL gate takes an average power of 10 miliwatts.

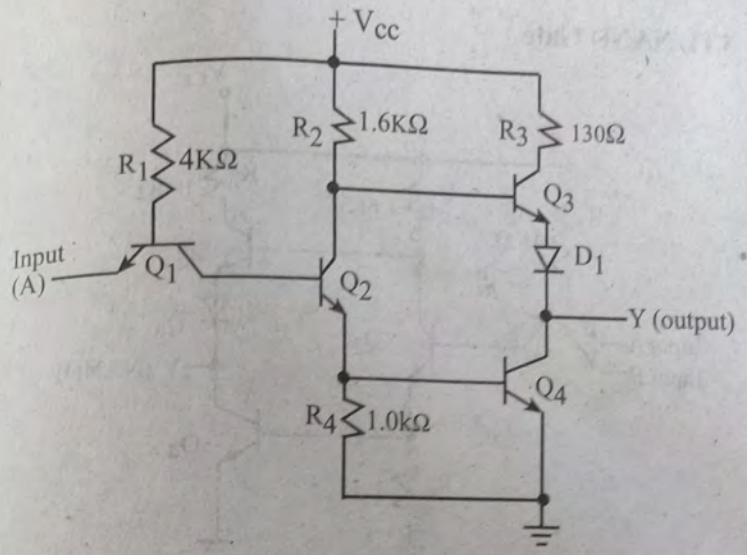
iv) **Propagation delay:** It is approximately 10 nanosecond.

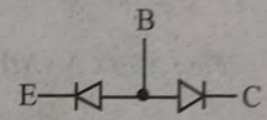
v) **Fan-out**: For standard TTL, fan-out is 10.

vi) **Noise margin:** The noise margin for the TTL family is 0.4V.

1. TTL Inverter (NOT Gate)

Totem-pole arrangement: Here, the combination of Q_3 and Q_4 transistors forms the output circuit often referred to as totem-pole arrangement.





npn transistor (Q_1)

When the input is HIGH, the base-emitter junction of Q_1 is reversed biased and the base-collector junction is forward biased. This condition permits current through R_1 and the base-collector junction of Q_1 into the base of Q_2, thus driving Q_2 into saturation (Q_2 'ON'). As a result, Q_4 is turned ON by Q_2, and its collector voltage, which is the output, is at near ground potential. Therefore, the output is LOW for HIGH input. At the same time, the collector of Q_2 is at LOW (ground) which keeps Q_3 OFF.

When the input is LOW, the base-emitter junction is forward biased and the base-collector junction is reversed biased. There is current through R_1 and the base-emitter junction of Q_1 to the LOW input. A LOW input provides path to ground for current. There is no current on the base of Q_2, so it is OFF. Thus, collector of Q_2 is HIGH, causing transistor Q_3 turned ON, meanwhile transistor Q_4 is turned OFF. Hence, the output is HIGH. Diode 'D_1' provides an additional V_{BE} (0.7 V) equivalent drop in series with base-emitter junction of Q_3 to ensure it is turned OFF when Q_2 is ON.

2. TTL NAND Gate

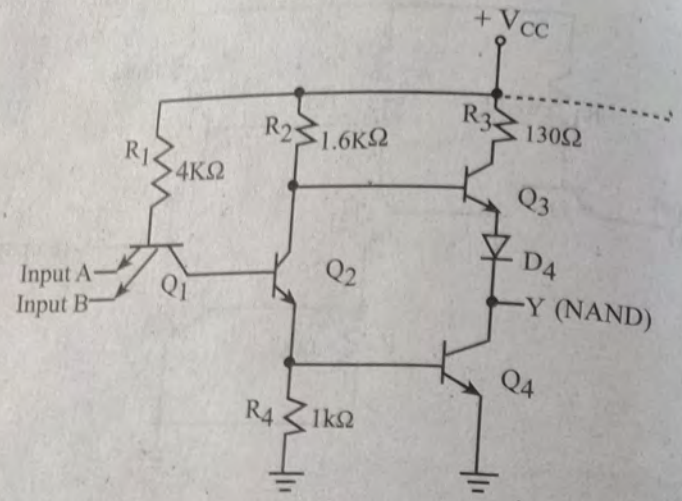


Fig.: A TTL NAND gate circuit

1. A LOW input on either input A or input B or both A and B forward biases the base-emitter junction of Q_1 and reverse biases the base-collector junction of Q_1 i.e., transistor Q_1 is saturated. This causes Q_2 to turn OFF, thereby causing Q_4 to turn OFF. Meanwhile, Q_3 is turned ON and the output is HIGH.
2. A HIGH on both input reverse biases the base-emitter junction of Q_1 and forward biases the base collector junction of Q_1. This causes Q_2 to turn ON which in turn makes transistor Q_4 ON and the output is LOW.

Inputs		State			Output
A	B	Q_2	Q_3	Q_4	Y
0	0	OFF	ON	OFF	1
0	1	OFF	ON	OFF	1
1	0	OFF	ON	OFF	1
1	1	ON	OFF	ON	0

3. TTL NOR Gate

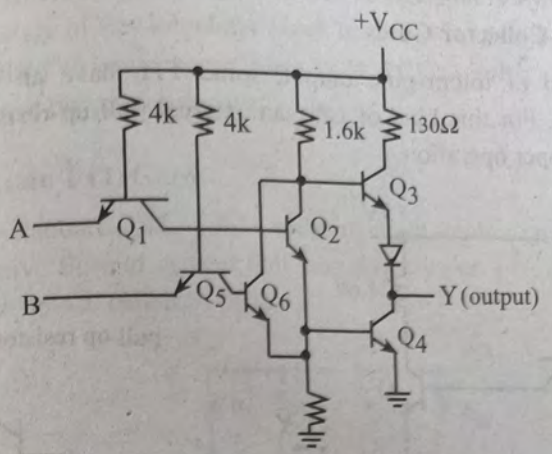


Fig.: TTL NOR gate circuit

Here, Q_5 and Q_6 have been added to basic NAND gate design. Since Q_6 and Q_2 are in parallel we get OR function which is followed by inversion to get NOR function.

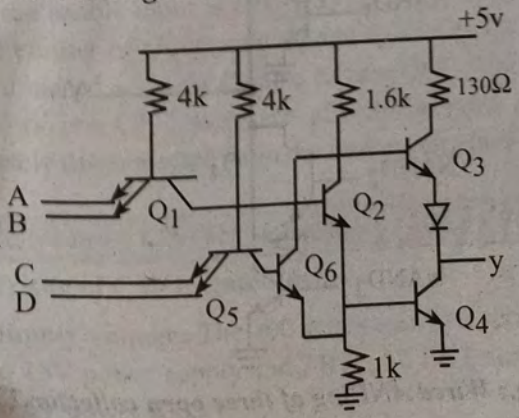
When A and B are LOW, the bases of Q_1 and Q_5 are pulled LOW, thus cuts off Q_2 and Q_6. Then, Q_3 is turned ON and Q_4 is OFF. Thus, output is HIGH.

When A or B is HIGH or both are HIGH, Q_1 and Q_5 are cut off respectively forcing Q_2 or Q_6 or both turned ON. This causes Q_4 to ON and pulling the output voltage to LOW.

4. TTL AND and OR Gate

To produce the AND function, another inverting stage is inserted to basic NAND. Similarly, to produce OR function, another inverting stage is inserted to basic NOR.

5. AND-OR-INVERT gate:



Here, Q_1 & Q_5 acts as 2 input AND gate, Q_2 & Q_6 produce ORing of output of Q_1 and Q_5, and Q_4 performs inversion.

6. **Open-Collector Gates**

Instead of totem–pole output, some TTL have an open collector output. For this kind of gate, an external pull-up resistor is required for proper operation.

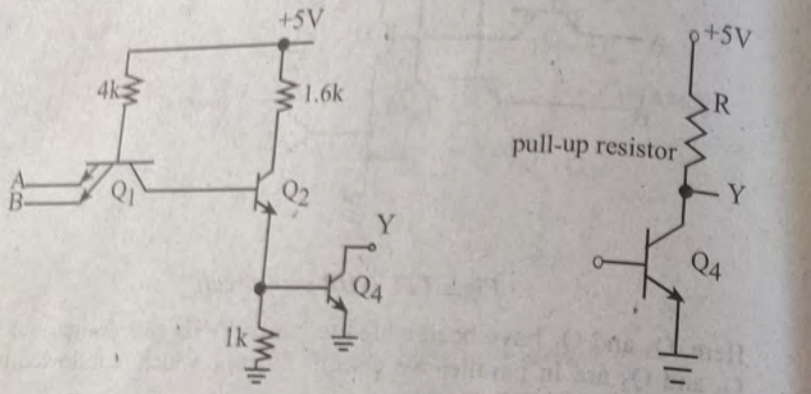


Fig.: (a) Open-collector NAND TTL *(b)pull-up resistor*

Advantages:

It has the advantages of combining three devices without using a final OR gate (or AND gate). This is done through direct connecting of the '3' outputs to the lower end of common pull–up resistor.

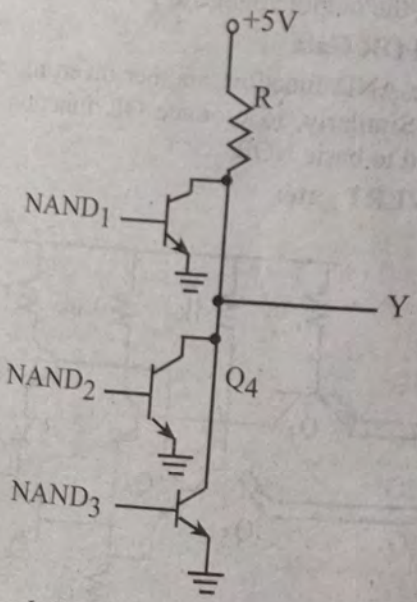


Fig.: Wired ANDing of three open collector TTL gates.

Disadvantages:

Open collector gate is slow in switching speed because the pull-up resistor are of few kiloohms which results in long time constant when multiplied to stray capacitance (τ = RC). Another disadvantage is increased power dissipation.

7. **Tri-State TTL Gates**

For a standard TTL, while we wire more gates together there is an excessive flow of current that may destroy our TTL device. This led to new device called tri-state TTL.

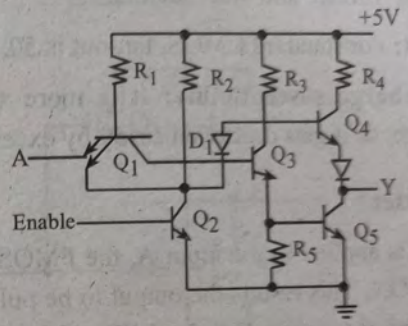
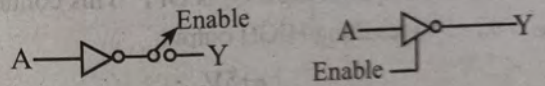


Fig.: Basic tri-state inverter



(a) Equivalent ckt (b) Logic symbol

When the enable input is LOW, Q_2 is OFF and the output circuit operates as a normal totem-pole configuration in which the output state depends on the input state.

When the enable input is HIGH, Q_2 is ON. There is thus low on the ground emitter of Q_1 causing Q_3 and Q_5 to turn OFF and diode D_1 is forward biased, causing Q_4 also to turn OFF. When both totem-pole transistors are OFF, they are effectively open and the output is completely disconnected from the internal circuitry.

0.5 74XX Series CMOS Gates

The characteristic of CMOS gates include:

i) **Supply voltage:** The 400 series and 74C series operate with 3V to 15V power supply and 74HC, 74 HCT series operate with 2V to 6V power supply.

ii) **Temperature range:** 74C series operates properly in ambient temperature ranging from –40°C to +85°C and this is adequate for most of the commercial applications. But 54C series can be operated in temperature ranging from –55°C to +125°C.

iii) **Power dissipation:** CMOS logic circuit dissipates 2.5nW power per gate (extremely low power). This is the reason why CMOS is used so widely.

iv) **Propagation delay:** A standard CMOS gate has a propagation delay time approximately 25 ns to 100 ns depending on the operating voltage and other factors.

v) **Fan-out:** For standard CMOS, fan-out is 50.

vi) **Static charge susceptibility:** It is more susceptible to static discharge, so it gets destroyed easily by excessive gate voltage.

1. CMOS Inverter

When HIGH is applied to the input A, the PMOS Q_1 is OFF and the NMOS Q_2 is ON. This results the output to be pulled down to ground voltage causing the output Y to be LOW. When LOW is applied to the input, transistor Q_1 is ON and Q_2 is OFF. This connects the output to $+V_{DD}$ (i.e., +5V) resulting HIGH output.

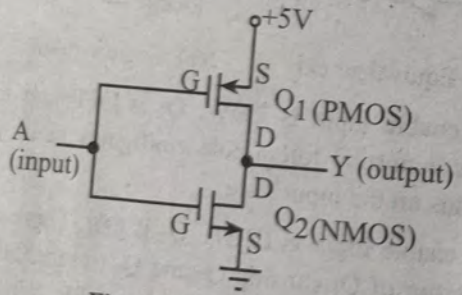


Fig.: CMOS inverter

Input			Output
A	Q_1	Q_2	Y
0	ON	OFF	1
1	OFF	ON	0

2. CMOS AND Gate

Input						Output
A	B	Q_1	Q_2	Q_3	Q_4	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

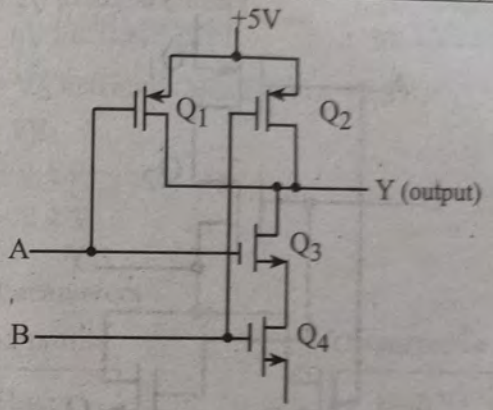


Fig.: CMOS NAND gate

When both inputs are LOW, Q_1 & Q_2 are ON, and Q_3 & Q_4 are OFF. The output is pulled HIGH through the ON resistance of Q_1 & Q_2 in parallel.

When A is LOW, B is HIGH, Q_1 & Q_4 are ON, and Q_2 & Q_3 are OFF. The output is pulled HIGH through the on resistance of Q_1.

When A is HIGH, B is LOW, Q_1 & Q_4 are OFF, and Q_2 & Q_3 are ON. The output is pulled HIGH through the low on resistance of Q_2.

When both inputs are HIGH, Q_1 & Q_2 are cut off, and Q_3 & Q_4 are ON. In this case, the output is pulled low through the on resistance of Q_3 & Q_4 in series to ground.

3. CMOS NOR Gate

Input						Output
A	B	Q_1	Q_2	Q_3	Q_4	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

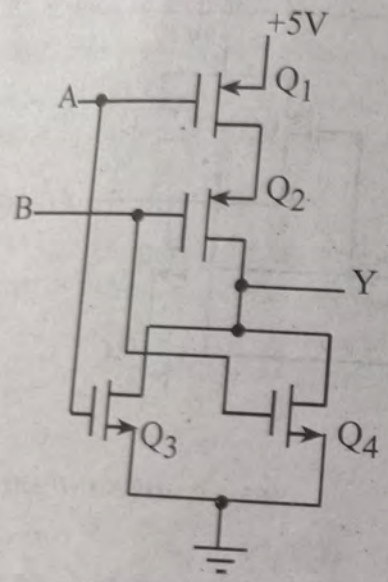


Fig.: CMOS NOR gate

When both inputs are LOW, Q_1 & Q_2 are ON, and Q_3 & Q_4 are OFF. As a result, the output is HIGH through Q_1 and Q_2 in series.

When A is LOW, B is HIGH, Q_1 & Q_4 are ON, and Q_2 & Q_3 are OFF. The output is LOW through Q_4 to ground.

When A is HIGH, B is LOW, Q_1 & Q_4 are OFF, and Q_2 & Q_3 are ON. The output is LOW through Q_3 to ground.

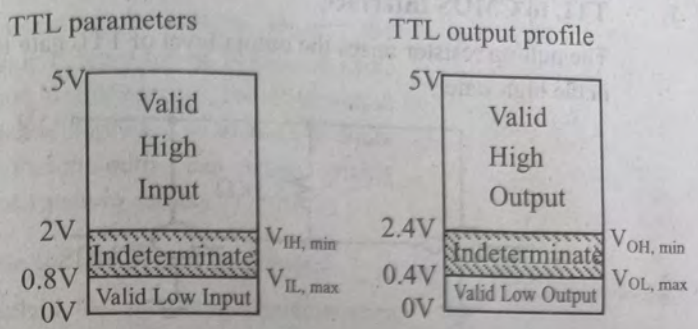
When both inputs are HIGH, Q_1 & Q_2 are OFF, and Q_3 & Q_4 are ON. The output is LOW through Q_3 & Q_4 in parallel.

4. **CMOS AND and OR gates**

These can be formed by just inserting the inverter to their

10.6 TTL and CMOS Parameters

1) **TTL Parameters**



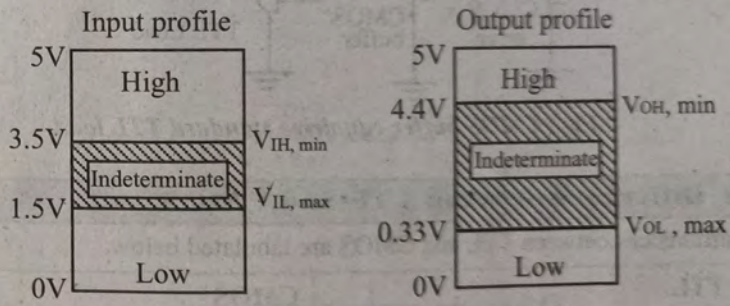
$V_{IL, max} = V_{in}$ maximum which is low = 0.8V

$V_{IH, min} = 2V$

$V_{OL, max} = 0.4$ V

$V_{OH, min} = 2.4$ V

2) **CMOS Parameters**



10.7 Interface between TTL and CMOS

The different voltage level requirements of TTL and CMOS technology presents problem when these two types of gates exist in the same system. Although CMOS and TTL gates are operating on the same 5V power supply, e output voltage level of TTL is not compatible to input voltage level of MOS and vice-versa.

Example:

For instance, if the TTL gate outputs a high signal with 3V (outputs HIGH between 2.4V and 5V) is fed to CMOS input, it might not be properly interpreted by CMOS gates input as HIGH (HIGH is between 3.5V to 5V).

This results the high output of TTL to uncertain region of CMOS output and may be interpreted as LOW.

1. **TTL to CMOS Interface**

The pull-up resistor raises the output level of TTL gate to about +5 V, in the high state.

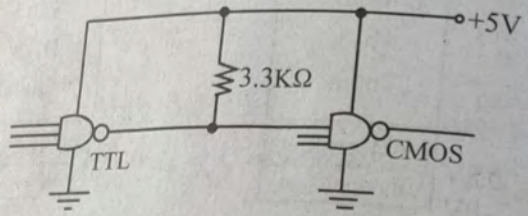


Fig.: TTL drive and CMOS load

2. **CMOS to TTL Interface**

Buffer increases the current sourcing capacity of CMOS gate.

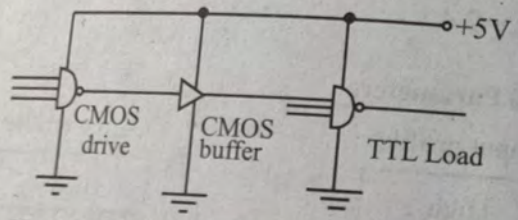


Fig.: CMOS buffer can drive standard TTL load.

10.8 Difference between TTL and CMOS

The differences between TTL and CMOS are tabulated below:

	TTL		CMOS
1)	TTL circuits utilize BJTs (Bipolar Junction Transistors).	1)	CMOS circuits utilize FETs (Field Effect Transistors).
2)	Less density of logic function can be fabricated in a single chip.	2)	It allows much higher density of logic function in a single chip compared to TTL.
3)	TTL chips are less susceptible to static discharge compared to CMOS.	3)	More susceptible to static discharge compared to TTL.
4)	Consumes more power compared to CMOS.	4)	Consumes less power.

MORE WORKED OUT EXAMPLES:

1. **What do you mean by totem-pole output?**

The totem pole output means the transistor T_1 sits a top of T_2 so as to gsive low output impedence. The low output impedance implies a short time constant RC so that the output can change quickly from one state to another.

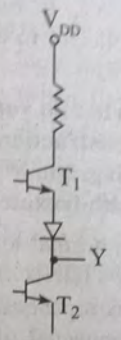


Figure: Tottem-pole output

2. **Explain the use of BJT as a switch.**

⇒ To explain the switching characteristics of BJT let us take an npn transistor of common emitter circuit.

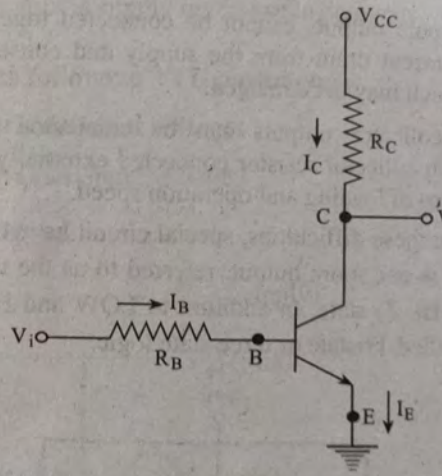


Fig.: Common emitter BJT

The current marked I_C flows through resister R_C and the collector of the transistor current I_B flows through resistor R_B and the base of the transistor. The emitter is connected to ground and $I_E = I_B + I_C$. The supply voltage is between V_{CC} and ground. The input is between V_i and ground and the output is between V_0 and ground.

Case I

If V_{BE} is than 0.6V ≈ 0V, the transistor is cut off with $I_B = 0$. The collector to emitters circuit when behaves like an open circuit with $I_C = 0$ and drop R_C is 0 and $V_0 = V_{CC}$.

Case II

If V_{BE} is from 0.8V to $V_{CC} \approx V_{CC}$, the transistor is biased with collector to emitter voltage $V_{CE} = 0$ this leads $I_C = V_{CC}/R_C$ So, $V_0 =$ 0V.

3. **What do you understand by the term 'common bus'? Explain the construction of common bus using Tristate buffer with a clear diagram. What issues should be considered when designing a bus with tristate buffer?**

⇒ In normal logic circuits, there are two states of output, either 'LOW' or 'HIGH'. But in complex systems like microcomputers and microprocessors, a number of gate outputs may be required to be connected to common line which is referred to as a "bus" which, inturn, may be required to drive a number of gate inputs. When a number of gate output are connected to the bus, some difficulties are encountered these are:

i. Totem-pole outputs cannot be connected together due to very large current drain from the supply and consequent heating of IC's which may get damaged.

ii. Open collector outputs can be connected together with a common collector resistor connected externally. This causes the problems of loading and operation speed.

To overcome these difficulties, special circuit have been developed in which there is one more output, referred to as the third state or high impedance (Hi–Z) state. In addition to LOW and HIGH states, such circuit are called Tristate or three state logic.

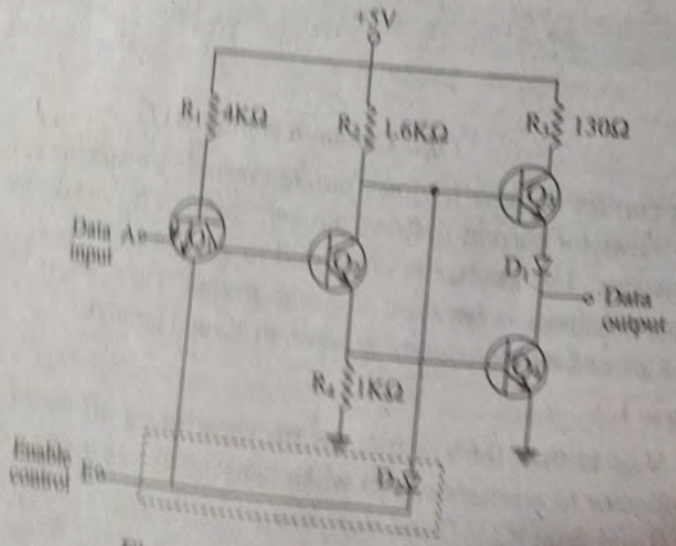


Fig.: Tristate inverter (circuit d

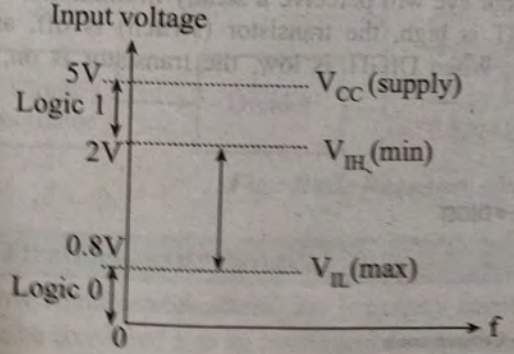
The tristate operation is achieved by modifying the basic totem–pole circuit. Figure above shows the circuit for a Tristate inverter where the portion enclosed in a dotted rectangle has been added to a basic circuit. The circuit has two inputs: A is the normal logic input and E is enable input capable of producing Hi–Z state. When E = 0 the circuit goes into Hi–Z state regardless of the state of logic input A. The LOW E forward biases the emitter –base junction of Q_1 and shunt the resistor R_1 current away from Q_2 and that Q_2 turns off, which turns Q_2 off. The LOW at E also forward biases D_2 to shunt current away from base of Q_3 so that Q_3 also turns off with both totem–pole transistors in the off state, the output terminal is essentially open circuit.

When E = 1, the circuit operates at normal inverter because HIGH input at E has no effect on transistor Q_1 and D_2. In this enabled condition output is simply inverse of logic inputs.

4. **Discuss the following TTL parameters:**

i. Worst-case input voltages

⇒ Worst case input voltage:



Ideally logic 0 is the input voltage level of OV and logic 1 is the input voltage level of +5V. But, practically the exact matching of voltage level is not obtained. Thus, the maximum input voltage which is considered as logic 0 is V_{IL}(max) and the minimum input voltage which is considered as logic 1 is V_{IH}(min) as shown in graph. These are the worst case input voltage levels.

Chapter – 11

APPLICATIONS

In this chapter, we will try to tie together many of the fundamental ideas presented previously by considering some of the more common digital circuit design encountered in industry.

11.1 Multiplexing Displays

The decimal outputs of digital instruments such as digital voltmeters and frequency counters are often displayed using seven-segment indicators. Such indicators are constructed by using a fluorescent bar, a liquid crystal bar, or a LED bar for each segment. LED-type indicators are convenient because they are directly compatible with TTL circuits, do not require the higher voltages used with fluorescents, and are generally brighter than liquid crystals. On the other hand, LEDs do generally require more power than either of the other two types, and *multiplexing* is a technique used to reduce indicator power requirements.

Basically, multiplexing is accomplished by applying current to each display digit in short, repeated pulses rather than continuously. If the pulse repetition rate is sufficiently high, our eye will perceive a steady illumination without any flicker. When DIGIT is high, the transistor (switch) is off, and the indicator current is zero. When DIGIT is low, the transistor is on, and a number is displayed.

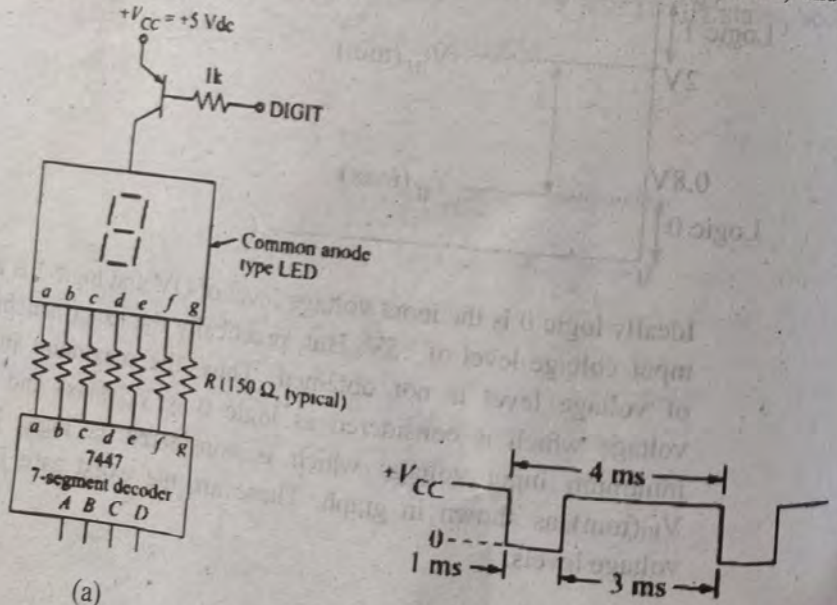


Fig.: (a) Multiplexed display (b) DIGIT waveform

11.2 Frequency Counter

A frequency counter is a digital instrument that can be used to measure the frequency of any periodic waveform. The fundamental concepts involved are illustrated in the block diagram shown below. A GATE ENABLE signal that has a known period t is generated with a clock oscillator and a divider circuit and is applied to leg of an AND gate. The unknown signal is applied to the other leg of the AND gate and output of AND gate acts as the clock for the counter. The counter will advance one count for each transition of the unknown signal, and at the end of the known time period, the contents of the counter will be equal to the number of periods of the unknown signal that have occurred during t. In other words, the counter contents will be proportional to the frequency of unknown signal.

For instance, suppose that the gate signal is exactly 1s and unknown input signal is a 750-Hz square wave. At the end of 1 s, the counter will have counted up to 750, which is exactly the frequency of the input signal.

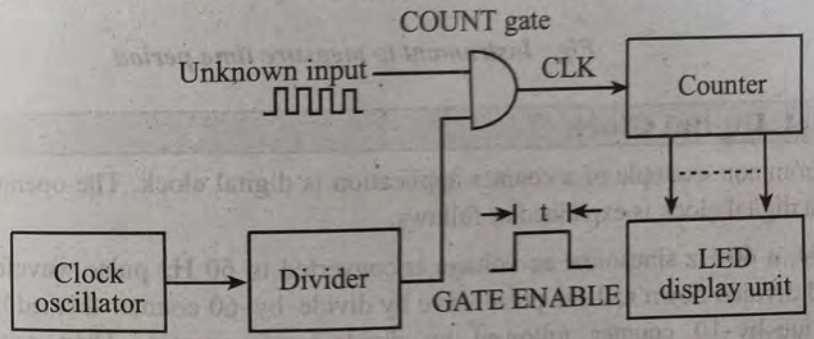


Fig.: Basic frequency counter

11.3 Time Measurement

With only slight modifications, the frequency counter we have studied just now can be converted into an instrument for measuring time. The logic block diagram shown below illustrates the fundamental ideas used to construct an instrument that can be used to measure the period of any periodic waveform. The unknown voltage is passed through a conditioning amplifier to produce a periodic waveform that is compatible with TTL circuits and is then applied to a JK flip-flop. The output of this flip-flop is used as the ENABLE-gate signal, since it is high for a time t that is exactly equal to the time period of the unknown input voltage. The oscillator and divider provides a series of pulses that are passed through the count gate and serve as the clock for the counter. The contents of the counter and display unit will then be proportional to the time period of the unknown input signal.

For instance, if the unknown input signal is a 5-kHz sine wave and the clock pulses from the divider are 0.1 μs in width and are spaced every 1.0 μs, the counter and display will read 200. Clearly, this means 200 μs, since 200 of these 0.1-μs pulses will pass through the COUNT gate during the 200 μs that ENABLE-gate signal is high. Naturally, the counter and the display have an accuracy of plus or minus one count.

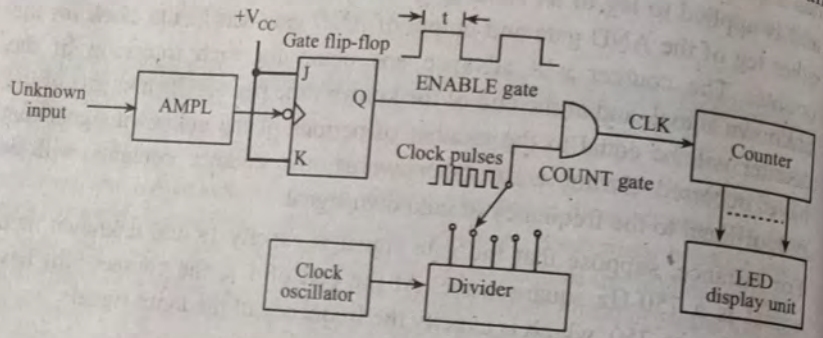


Fig.: Instrument to measure time period

11.4 Digital Clock

A common example of a counter application is digital clock. The operation of a digital clock is explained as follows.

First, a 60 Hz sinusoidal ac voltage is converted to 60 Hz pulse waveform and divided down to 1 Hz pulse wave by divide–by–60 counter formed by a divide–by–10 counter followed by divide–by–6 counter. Then second divide–by–60 counter produces value for 'seconds' which counts from 00 to 59 and then recycle to 00 at each second. The third divide–by–60 counter changes state once each minute and counts from 00 to 59 minutes. The last counter changes state once on each 60 minutes (once each hour). Thus it is a divide–by–12 counter, it will count from 1 to 12 states that can be decoded to provide signals to display the correct hour. The values of counter are displayed in seven segment display unit using BCD to seven segment decoder.

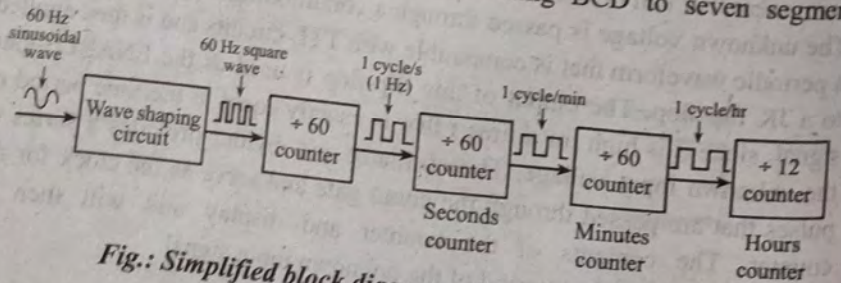


Fig.: Simplified block diagram of diagram of a digital clock

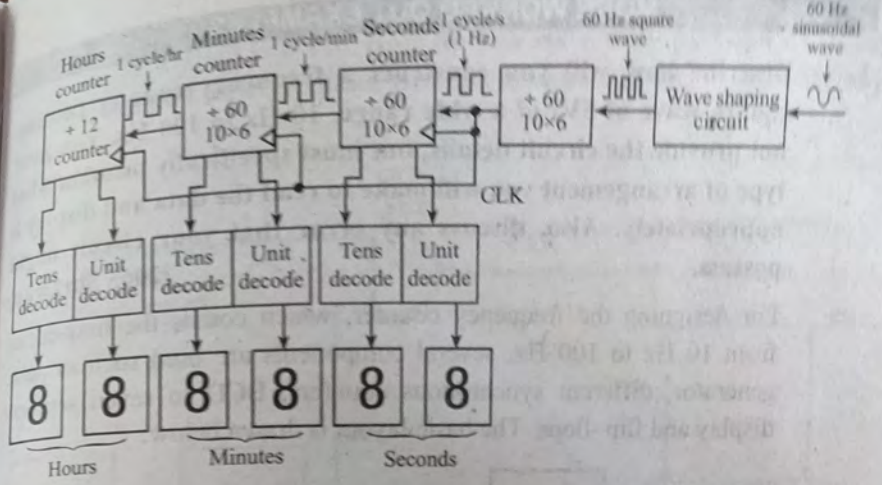
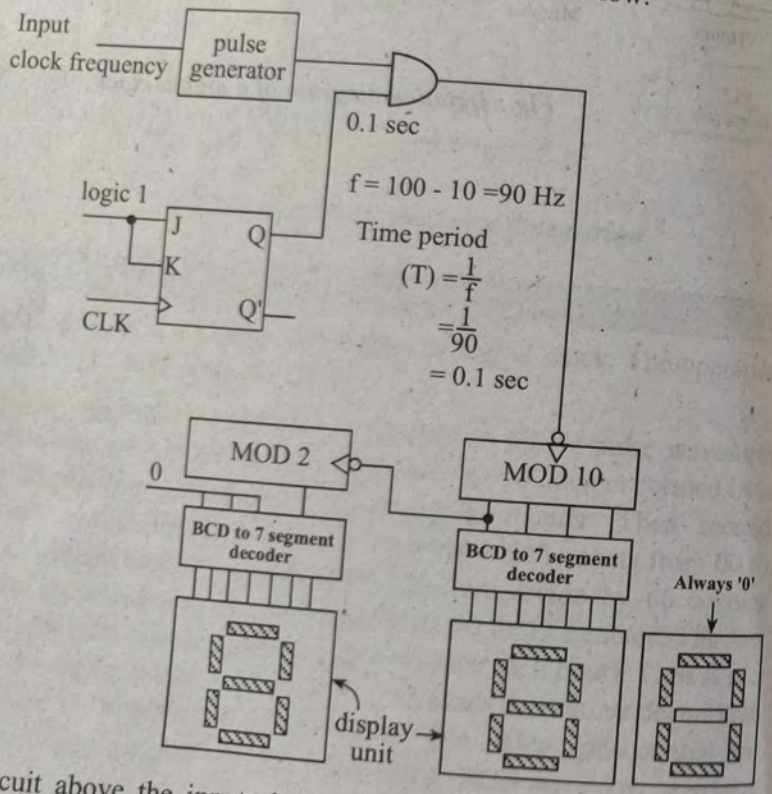


Fig.: Detailed diagram of a digital clock

MORE WORKED OUT EXAMPLES:

1. **Describe how will you construct a frequency counter (assume square wave of 5V) of a wide range, 10 Hz to 100 Hz. You need not provide the circuit details, but must specifically mention what type of arrangement you will make to read the data and display it appropriately. Also, discuss any error that your circuit might possess.** **[2065 Shrawan]**

⇒ For designing the frequency counter, which counts the frequencies from 10 Hz to 100 Hz, several components are used such as pulse generator, different synchronous counters, BCD to seven segment display and flip–flops. The basic layout is drawn below:



In circuit above the input clock frequency is generated to a clock pulses by pulse generator and ANDed with 0.1 sec clock pulses. Three seven segment displays are taken and rightmost display is taken to always '0' and respective onward displays are driven by MOD 10 and MOD 2 synchronous counters.

Limitation: This circuit only counts the signals which are only a multiple of 10. The other frequency signals, this circuit can't works.

2. **A digital system requires 3 Hz clock, but the clock coming to this digital system is 6 Hz. How do you solve such problem by using sequential logic? Explain with necessary diagram.** **[2064 Jestha]**

⇒

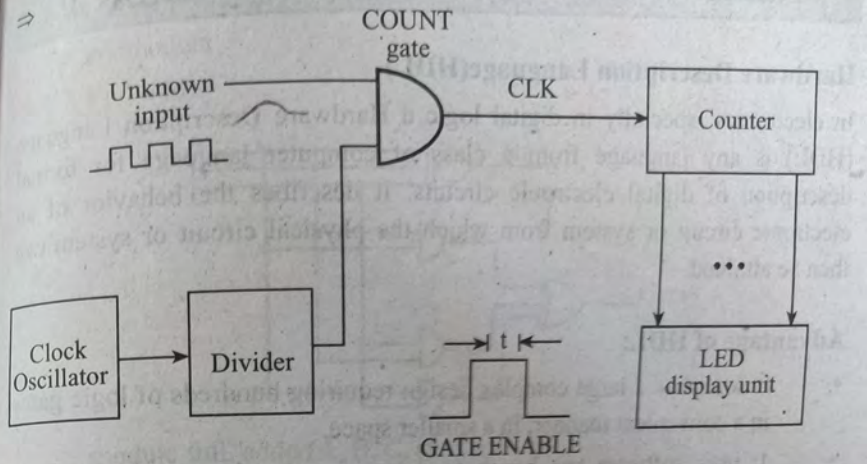


Figure above shows the digital system that counts the frequency which of 3 Hz clock signals as the input to a counter. Initially applied 6 Hz clock is fed to clock oscillator and according to system requirement this clock signal is divided into half by divider circuit. Now the 3 Hz clock signal is divided into half by divider circuit. Frequency signal of periodic nature is applied to the counting purpose.

APPENDIX

HARDWARE DESCRIPTION LANGUAGE

Hardware Description Language(HDL)

In electronics, specially in digital logic a Hardware Description Language (HDL) is any language from a class of computer language for formal description of digital electronic circuits. It describes the behavior of an electronic circuit or system from which the physical circuit or system can then be attained.

Advantage of HDL:

- It describes a large complex design requiring hundreds of logic gates in a convenient manner, in a smaller space.
- It uses software test–bench to detect functional error, if any, and correct it (called simulation).
- Finally, get hardware implementation details (called synthesis).

HDL types:

Currently there are two widely used HDLs.

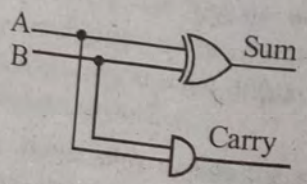
1. Verilog HDL
2. VHDL (Very high speed integrated circuit HDL)

Verilog is the first introduced in 1985 and is considered to be simpler than VHDL and more popular. However both of this two share lot of common features and it is not difficult to switch from one to the other. Here, we will be discussing about Verilog only.

Examples using Gate Modeling or Structural Model:

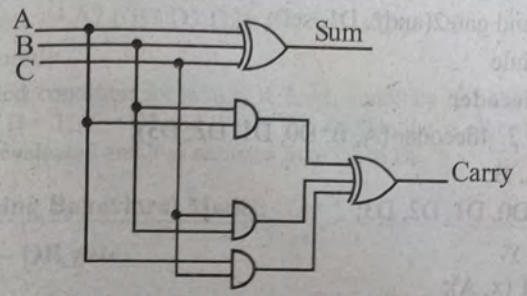
1) **Half adder**

```
module half_adder (A,B, sum, carry);
```




```
input A, B;
output sum, carry;
xor sum_1(sum, A, B);        // sum = A xor B
and carry_1(carry, A, B);    // carry = A and B
endmodule
```

2) **Full adder**




```
module full_adder(A, B, C, sum, carry);
input A, B, C;
output sum, carry;
wire and1, and2, and3;        // wire shows gate level inter connection
  and U_and1(and1, A, B);
  and U_and2(and2, B, C);
  and U_and3(and3, A, C);
  or U_or(carry, and1, and2, and3);    // carry
  xor U_sum(sum, A, B, C);             //sum
endmodule
```

3) **Full subtractor**

```
module full_subtractor (A, B, C, difference, borrow);
input A, B, C;
output difference, borrow;
wire inv_A, and1, and2, and3;
      not i1 (inv_A, A);
      and U_and1(and1, inv_A, B);
      and U_and2(and2, inv_A, C);
      and U_and3(and3, B, C);
      xor U_difference (difference, A, B, C);
      or U_borrow (borrow, and1, and2, and3);
endmodule
```

4) **2:1 MUX**

```
module 2_1MUX (D0, D1, sel, y);
input D0, D1, sel;
output y;
wire inv_sel, and1, and2;
      not i1(inv_sel, sel);
      and gate1(and1, D0, inv_sel);
      and gate2(and2, D1, sel);
endmodule
```

5) **2 to 4 decoder**

```
module 2_4decoder (A, B, D0, D1, D2, D3);
input A, B;
output D0, D1, D2, D3;
wire x, y;
 not i1 (x, A);
 not i2 (y, B);
 and and1(D0, x, y);
 and and2(D1, x, B);
 and and3(D2, A, y);
 and and4(D3, A, B);
endmodule
```

6) **4 to 2 encoder**

```
module encoder4_2(D0, D1, D2, D3, y);
input D0, D1, D2, D3;
output [1,0]y;        //y in array form i.e. two output y[0] and y[1]
 or output1(y[0], D1, D3);    // y[0] = D1 +D3
 or output2(y[1], D2, D3);    // y[1] = D2 + D3
endmodule
```

Examples using Data Flow Model:

1) **2 to 1 MUX**

```
module 2_1MUX(y, D0, D1, S0);
input D0, D1, S0;
output y;
 assign y = (~ S0& D0) | (D1& S0);
endmodule
```

OR

```
module 2_1MUX (y, D0, D1);
input D0, D1, S0;
output y;
 assign y = S0? D1:D0   // if S0 = 1, y = D1; if S0 = 0, y = D0
endmodule
```

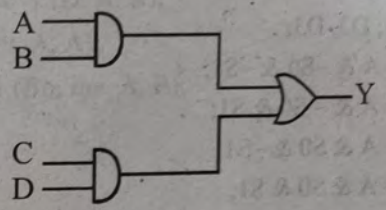
2) **4 to 1 MUX**

```
module 4_1MUX(A, B, D0, D1, D2, D3, y);
input A, B, D0, D1, D2, D3;
output y;
 assign y= A? ((B? D3:D2): (B? D1:D0));
endmodule
```

/* nested condition for assign, if A =1, condition (B? D3:D2) is evaluated, then if B = 1, Y = D3 and B = 0, Y = D2. Similarly, other combination of A and B evaluated and Y is assigned from D2 to D0. */

Examples using Behavioral Model:

1) **AND – OR gate**




```
module figure (A, B, C, D, Y);
input A, B, C, D;
output Y;
reg Y;
always @ (A or B or C or D)
if ((A = = 1) && (B = = 1))
  Y = 1;
else if ((C = = 1) && (D = = 1))
  Y = 1;
else
  Y = 0;
endmodule
```

2) **4 to 1 MUX**

```
module 4_1MUX (D0, D1, D2, D3, sel, y);
input D0, D1, D2, D3;
input [1,0]sel;    //select as array sel[0] and sel[1]
output y;
reg y;
```

```
always @ (D0 or D1 or D2 or D3 or sel)
case (sel)
0: y = D0;   // sel [0] = 0 and sel [1] = 0
1: y = D1;   // sel [0] = 0 and sel [1] = 1
2: y = D2;   // sel [0] = 1 and sel [1] = 0
3: y = D3;   // sel [0] = 1 and sel [1] = 1
endcase
endmodule
```

MORE WORKED OUT EXAMPLES:

1. Implement 1:4 DEMUX using VHDL **[2069 Ashadh]**

⇒ *HDL for 1 to 4 DEMUX*

```
module 1to4DMUX (A, S0, S1, D0, D1, D2, D3);
input A, S0, S1;
output D0, D1, D2, D3;
 assign D0 = A & ~S0 & ~S1;
 assign D1 = A & ~ S0 & S1;
 assign D2 = A & S0 & ~S1;
 assign D3 = A & S0 & S1;
endmodule
```

2. Design full adder circuit using HDL. **[2069 Ashadh]**

⇒ *HDL for full adder*

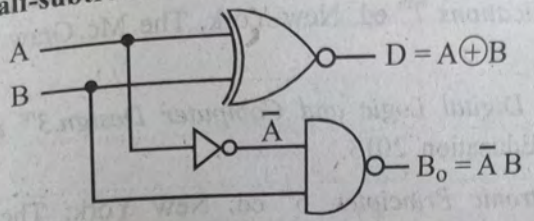
```
module fulladder (A, B, C Sm, Cr);
input A, B, C;
output Sm, Cr;
 assign Sm = A^B^C;
 assign Cr = (A & B) | (B & C) | (C & A);
endmodule
```

3. Design 2 to 4 line decoder circuit using HDL. **[2068 Baishakh]**

⇒ *HDL for 2 to 4 decoder*

```
module 2to4decoder (A,B,D0,D1,D2,D3);
input A, B;
output D0, D1, D2, D3;
 assign D0 = ~A & ~B;
 assign D1 = ~A&B;
 assign D2 = A&~B;
 assign D3 = A&B;
endmodule
```

4. Design half-subtractor using HDL. **[2075 Ashwin]**




```
Module half–subtractor (A, B, D, Bo);
input A, B;
output D, Bo;
wire inv_A;
x or difference_1 (D, A, B);
not not_A (inv_A, A);
and borrow_1 (Bo, inv_A, B);
endmodule
```

BIBLIOGRAPHY

Donald P. Leach, Albert Paul Malvino, Goutam Saha. *Digital Principles and Applications* 7th ed. New York, The Mc Graw-Hill Companies, 2011.

Morris Mano. *Digital Logic and Computer Design.3rd ed.* New Jersey: Pearson Education, 2013.

Malvino. *Electronic Principles.* 6th ed. New York: The Mc Graw-Hill Companies, 2006.

Thomas L. Floyd. *Electronic Devices.* 8th ed. New Jersey: Pearson Education, 2007.

Thomas L. Floyd. *Digital Fundamentals.* 10th ed. New Jersey: Pearson Education, 2013.

J.B. Gupta. *A Course in Electrical Technology.* 12th ed. New Delhi: S.K. Kataria & Sons, 2010.

Note: