# Computer Science & IT

# Digital Logic

Comprehensive Theory

*with* Solved Examples and Practice Questions

MADE EASY

MADE EASY

India's Best Institute for IES, GATE & PSUs

# Contents

## Digital Logic


Chapter 1

**Basics of Digital Logic ..... 1**

1.1. Digital Number Systems ..... 2
1.2 Codes ..... 7
1.3 Arithmetic Operations ..... 10
1.4 Signed Number Representation ..... 13
1.5 Over Flow Concept ..... 19
*Student Assignments* ..... 21

Chapter 2

**Boolean Algebra & Minimization Techniques ..... 22**

2.1 Logic Operations ..... 22
2.2 Laws of Boolean Algebra ..... 23
2.3 Boolean Algebric Theorems ..... 24
2.4 Minimization of Boolean Functions ..... 27
2.5 Representation of Boolean Functions ..... 27
2.6 Implicants, Prime Implicants and Essential Prime Implicants ..... 37
*Student Assignments* ..... 41

Chapter 3

**Logic Gates and Switching Circuits ..... 43**

3.1 Basic Gates ..... 43
3.2 Universal Gates ..... 49
3.3 Special Purpose Gate ..... 52
3.4 Realization of Logic Gates Using Universal Gates ..... 57
*Student Assignments* ..... 61

Chapter 4

**Combinational Logic Circuits ..... 63**

4.1 Design Procedure for Combinational Circuit ..... 63
4.2 Arithmetic Circuits ..... 64
4.3 Non-arithmetic Circuit ..... 77
4.4 Hazards ..... 108
*Student Assignments* ..... 111

Chapter 5

**Sequential Logic Circuits ..... 114**

5.1 Latches and Flip-Flops ..... 115
5.2 Race Around Condition ..... 126
5.3 Conversion of Flip-Flops ..... 129
5.4 Applications of Flip-Flops ..... 131
*Student Assignments* ..... 133

Chapter 6

**Registers ..... 135**

6.1 Shift Register ..... 135
*Student Assignments* ..... 144

Chapter 7

**Counters ..... 145**

7.1 Asynchronous/Ripple Counters ..... 147
7.2 Synchronous Counters ..... 152
7.3 Synchronous Counter Design ..... 159
7.4 State Diagram and State Table ..... 162
7.5 Finite State Model/Machine ..... 165
*Student Assignments* ..... 167


■■■

CHAPTER 01

# Basics of Digital Logic

## Introduction

Electronic systems are of two types:

(*i*) Analog systems (*ii*) Digital systems

Analog systems are those systems in which voltage and current variations are continuous through the given range and they can take any value within the given specified range, whereas a digital system is one in which the voltage level assumes finite number of distinct values. In all modern digital circuits there are just two discrete voltage level.

Digital circuits are often called switching circuits, because the voltage levels in a digital circuit are assumed to be switched from one value to another instantaneously. Digital circuits are also called logic circuits, because every digital circuit obeys a certain set of logical rules.

Digital systems are extensively used in control systems, communication and measurement, computation and data processing, digital audio and video equipments, etc.

### Advantages of Digital Systems

Digital systems have number of advantages over analog systems which are summarized below:

**1. Ease of Design**

The digital circuits having two voltage levels, OFF and ON or LOW and HIGH, are easier to design in comparison with analog circuits in which signals have numerical significance ; so their design is more complicated.

**2. Greater Accuracy and Precision**

Digital systems are more accurate and precise than analog systems because they can be easily expanded to handle more digits by adding more switching circuits.

**3. Information Storage is Easy**

There are different types of semiconductor memories having large capacity, which can store digital data.

**4. Digital Systems are More Versatile**

It is easy to design digital systems whose operation is controlled by a set of stored instructions called program. However in analog systems, the available options for programming is limited.

**5. Digital Systems are Less Affected by Noise**

The effect of noise in analog system is more. Since in analog systems the exact values of voltages are important. In digital system noise is not critical because only the range of values is important.

**6. Digital Systems are More Reliable**

As compared to analog systems, digital systems are more reliable.

### Limitations of Digital System

(*i*) The real world is mainly analog.

(*ii*) Human does not understand the digital data.

## 1.1. Digital Number Systems

Many number systems are used in digital technology. A number system is simply a way to count. The most commonly used number systems are:

- Decimal number system
- Binary number system
- Octal number system
- Hexadecimal number system

A number system with base or radix '*r*' will have *r* number of different digits from $0 \rightarrow (r-1)$ thus, number system is represented by $(N)_b$

where, $N$ = Number ; $b$ = Base or radix

In general a number with an integer part of '*n*' bits and a fraction part of '*m*' bits can be written as

$$(N)_b = \underbrace{b_{n-1}\, b_{n-2} \cdots b_1 b_0}_{\text{Integer part}} \underset{\text{Radix point}}{\overset{\cdot}{\uparrow}} \underbrace{b_{-1}\, b_{-2} \cdots b_{-m}}_{\text{Fraction part}}$$

### 1.1.1 Decimal Number System

- This system has **'base 10'**.
- It has 10 distinct symbols (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9).
- This is a positional value system in which the value of a digit depends on its position.

  ⇒ Let we have $(453)_{10}$ is a decimal number

  then,

4 5 3

$3 \times 10^0 = 3$

$5 \times 10^1 = 50$

$4 \times 10^2 = 400$

Finally we get, $(453)_{10}$

∴ We can say "3" is the least significant digit(LSD) and "4" is the most significant digit(MSD).

**Example - 1.1** A particular number system having base *B* is given as $(\sqrt{41})_B = 5_{10}$. The value of '*B*' is

(a) 5 (b) 6

(c) 7 (d) 8

***Solution: (b)***

Squaring both side, $(\sqrt{41})^2 = (5)^2$

$$(41)_B = (25)_{10}$$

$$(4B+1)_{10} = (25)_{10}$$

$$\Rightarrow \quad B = 6$$

### 1.1.2 Binary Number System

- It has base '2' i.e. it has two base numbers 0 and 1 and these base numbers are called "Bits".
- In this number system, group of "Four bits" is known as "Nibble" and group of "Eight bits" is known as "Byte".

i.e. 4 bits = 1 Nibble; 8 bits = 1 Byte

**Binary to Decimal Conversion**

A binary number is converted to decimal equivalent simply by summing together the weights of various positions in the binary number which contains '1'.

**Example-1.2** The decimal number represention of 101101.10101 is

***Solution:***

$$(101101.10101)_2 = 1\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} + 0\times 2^{-4} + 1\times 2^{-5}$$

$$= 32 + 0 + 8 + 4 + 0 + 1 + \frac{1}{2} + 0 + \frac{1}{8} + 0 + \frac{1}{32} = (45.65625)_{10}$$

**Decimal to Binary Conversion**

The integral decimal number is repeatedly divided by '2' and writing the remainders after each division until a quotient '0' is obtained.

**Example-1.3** Convert $(13)_{10}$ to binary.

***Solution:***

| | Quotient | Remainder | |
|---|---|---|---|
| 13 ÷ 2 | 6 | 1 | LSB |
| 6 ÷ 2 | 3 | 0 | |
| 3 ÷ 2 | 1 | 1 | |
| 1 ÷ 2 | 0 | 1 | MSB |

$\therefore$ $(13)_{10} \Rightarrow (1101)_2$

**Remember**

To convert Fractional decimal into binary, Multiply the number by '2'. After first multiplication integer digit of the product is the first digit after binary point. Later only fraction part of the first product is multiplied by 2. The integer digit of second multiplication is second digit after binary point, and so on. The multiplication by 2 only on the fraction will continue like this based on conversion accuracy or until fractional part becomes zero.

**Example-1.4** Convert $(0.65625)_{10}$ to an equivalent base-2 number.

***Solution:***

| | | | | |
|---|---|---|---|---|
| 0.65625 × 2 | 0.31250 × 2 | 0.62500 × 2 | 0.25000 × 2 | 0.50000 × 2 |
| 1.31250 | 0.62500 | 1.25000 | 0.50000 | 1.00000 |
| 1 | 0 | 1 | 0 | 1 |

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

### 1.1.3 Octal Number System

- It is very important in digital computer because by using the octal number system, the user can simplify the task of entering or reading computer instructions and thus save time.
- It has a base of '8' and it posses 8 distinct symbols (0,1...7).
- It is a method of grouping binary numbers in group of three bits.

**Octal to Decimal Conversion**

An octal number can be converted to decimal equivalent by multiplying each octal digit by its positional weightage.

**Example - 1.5** Convert $(6327.4051)_8$ into its equivalent decimal number.

***Solution:***

$$(6327.4051)_8 = 6\times 8^3+3\times 8^2+2\times 8^1+7\times 8^0+4\times 8^{-1}+0\times 8^{-2}+5\times 8^{-3}+1\times 8^{-4}$$

$$= 3072+192+16+7+\frac{4}{8}+0+\frac{5}{512}+\frac{1}{4096}$$

$$= (3287.5100098)_{10}$$

Thus, $(6327.4051)_8 = (3287.5100098)_{10}$

**Decimal to Octal Conversion**

- It is similar to decimal to binary conversion.
- For integral decimal, number is repeatedly divided by '8' and for fraction, number is multiplied by '8'.

**Example - 1.6** Convert $(3287.5100098)_{10}$ into octal.

***Solution:***

For integral part:

| | Quotient | Remainder |
|---|---|---|
| 3287 ÷ 8 | 410 | 7 |
| 410 ÷ 8 | 51 | 2 |
| 51 ÷ 8 | 6 | 3 |
| 6 ÷ 8 | 0 | 6 |

$\therefore$ $(3287)_{10} = (6327)_8$

Now for fractional part:

$$\begin{array}{r} 0.5100098 \\ \times\ 8 \\ \hline 4.0800784 \end{array} \rightarrow \begin{array}{r} 0.0800784 \\ \times\ 8 \\ \hline 0.6406272 \end{array} \rightarrow \begin{array}{r} 0.6406272 \\ \times\ 8 \\ \hline 5.1250176 \end{array} \rightarrow \begin{array}{r} 0.1250176 \\ \times\ 8 \\ \hline 1.0001408 \end{array}$$

$$\downarrow \qquad \downarrow \qquad \downarrow \qquad \downarrow$$

$$4 \qquad 0 \qquad 5 \qquad 1$$

$\therefore$ $(0.5100098)_{10} = (0.4051)_8$

Finally, $(3287.5100098)_{10} = (6327.4051)_8$

**Octal-to-Binary Conversion**

This conversion can be done by converting each octal digit into binary individually.

**Example-1.7** Convert $(472)_8$ into binary

***Solution:***

$$\therefore \quad (472)_8 = (\underset{\Downarrow}{100}\ \underset{\Downarrow}{111}\ \underset{\Downarrow}{010})_2$$

**Binary-to-Octal Conversion**

In this conversion the binary bit stream are grouped into groups of three bits starting at the LSB and then each group is converted into its octal equivalent. After decimal point grouping start from left.

**Example-1.8** Convert $(1011011110.11001010011)_2$ into octal.

***Solution:***

For left-side of the radix point, we grouped the bits from LSB: $\underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{011}_{3}\ \underbrace{110}_{6}$

Here two 0's at MSB are added to make a complete group of 3 bits.

For right-side of the radix point, we grouped the bits from MSB: $\underset{\text{radix point}}{\bullet}\ \underbrace{110}_{6}\ \underbrace{010}_{2}\ \underbrace{100}_{4}\ \underbrace{110}_{6}$

Here a '0' at LSB is added to make a complete group of 3 bits.

Finally, $(1011011110.11001010011)_2 = (1336.6246)_8$

## 1.1.4 Hexadecimal Number System

- The base for this system is "16", which requires 16 distinct symbols to represent the numbers.
- It is a method of grouping 4 bits.
- This number system contains numeric digits (0, 1, 2,....9) and alphabets (*A*, *B*, *C*, *D*, *E* and *F*) both, so this is an "ALPHANUMERIC NUMBER SYSTEM".
- Microprocessor deals with instructions and data that use hexadecimal number system for programming purposes.
- To signify a hexadecimal number, a subscript 16 or letter '*H*' is used i.e. $(A7)_{16}$ or $(A7)_H$.

**Table-1.1**

| Hexadecimal | Decimal | Binary |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

### Hexadecimal-to-Decimal Conversion

**Example - 1.9** Convert $(3A.2F)_{16}$ into decimal system.

***Solution:***

$$(3A.2F)_{16} = 3 \times 16^1 + 10 \times 16^0 + 2 \times 16^{-1} + 15 \times 16^{-2}$$

$$= 48 + 10 + \frac{2}{16} + \frac{15}{16^2} = (58.1836)_{10}$$

### Decimal-to-Hexadecimal Conversion

**Example - 1.10** Convert $(675.625)_{10}$ into Hexadecimal.

***Solution:***

For Integral Part:

| | Quotient | Remainder |
|---|---|---|
| 675 ÷ 16 | 42 | 3 |
| 42 ÷ 16 | 2 | 10 = A |
| 2 ÷ 16 | 0 | 2 |

$\therefore$ $(675)_{10} = (2A3)_{16}$

For Fractional Part:

$$625 \times 16 = 10 = A$$

$\therefore$ $(0.625)_{10} = (0.A)_{16}$

Finally, $(675.625)_{10} = (2A3.A)_{16}$

### Hexadecimal-to-Binary Conversion

For this conversion replace each hexadecimal digit by its 4 bit binary equivalent.

**Example - 1.11** Convert $(2F9A)_{16}$ to Binary System

***Solution:***

| 2 | F | 9 | A |
|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ |
| 0010 | 1111 | 1001 | 1010 |

$\therefore$ $(2F9A)_{16} = (0010\ 1111\ 1001\ 1010)_2$

### Binary-to-Hexadecimal Conversion

For this conversion the binary bit stream is grouped into pairs of four (starting from LSB) and hex number is written for its equivalent binary group.

**Example - 1.12** Convert $(10100110101111)_2$ to hexadecimal number system.

***Solution:***

$$\underbrace{00\,10}_{2}\ \underbrace{10\,01}_{9}\ \underbrace{10\,10}_{A}\ \underbrace{1111}_{F}$$

*Here two 0's at MSB are added to make a complete group of 4 bits.*

$\therefore$ $(10100110101111)_2 = (29AF)_{16}$

The number systems can also be classified as weighted binary number and unweighted binary number. Where weighted number system is a positional weighted system for example, Binary, Octal, Hexadecimal *BCD*, 2421 etc. The unweighted number systems are non-positional weightage system for example Gray code, Excess-3 code etc.

## 1.2 Codes

When numbers, letters or words are represented by a special group of symbols, we say that they are being encoded, and the group of symbols is called "CODE".

**CODES**

| Non-Binary | Binary | Alphanumeric |
|---|---|---|
| **Ex:** Morse code | **Ex:** Gray code | **Ex:** Boudot's code (5 bit code) |
| ⇒ In Morse code series of dots and dashes represents the letters of the alphabet. | BCD, Excess-3 Code, 8-4-2-1 code Error detecting and correcting codes | Internal code (6 bit) ASCII code (7 bit + 1bit parity) EBCDIC (8 bit) |

### 1.2.1 Binary Coded Decimal Code (BCD)

- In this code, each digit of a decimal number is represented by binary equivalent.
- It is a 4-bit binary code.
- It is also known as "8-4-2-1 code" or simply "BCD Code".
- It is very useful and convenient code for input and output operations in digital circuits.
- Also, it is a "weighted code system".

For example:

$(943)_{decimal} \longrightarrow (.....)_{BCD}$

$$\Rightarrow \quad \begin{array}{ccc} 9 & 4 & 3 \\ \downarrow & \downarrow & \downarrow \\ 1001 & 0100 & 0011 \end{array}$$

$$\therefore \quad (943)_{10} = (100101000011)_2$$

**Advantages of BCD Code**

- The main advantage of the BCD code is relative ease of converting to and from decimal.
- Only 4-bit code groups for the decimal digits "0 through 9" need to be remembered.
- This case of conversion is especially important from the hardware standpoint.

  ⇒ In 4-bit binary formats, total number of possible representation = $2^4 = 16$

  Then, Valid BCD codes = 10

  Invalid BCD codes = 6

  ⇒ In 8-bit binary formats,

  Valid BCD codes = 100

  Invalid BCD codes = 256 – 100 = 156

### 1.2.2 Excess-3 Code

- It is a 4-bit code.
- It can be derived from BCD code by adding "3" to each coded number.
- It is an "unweighted code".

- It is a "self-complimenting code" i.e. the 1's compliment of an excess-3 number is the excess-3 code for the 9's compliment of corresponding decimal number.
- This code is used in arithmetic circuits because of its property of self complimenting.

**Example - 1.13** Convert $(48)_{10}$ into Excess-3 code.

***Solution:***

$$\begin{array}{cc} 4 & 8 \\ +3 & +3 \\ \hline 7 & 11 \\ \downarrow & \downarrow \\ 0111 & 1011 \end{array}$$

$$\therefore \quad (48)_{10} = (\underbrace{01111011}_{\text{equivalent 4-bit binary}})$$

**Example - 1.14** Represent the decimal number 6248 in

(*i*) BCD code (*ii*) Excess-3 code (*iii*) 2421 code

***Solution:***

(*i*) BCD code

$$\begin{array}{cccc} 6 & 2 & 4 & 8 \\ 0110 - & 0010 - & 0100 - & 1000 \end{array}$$

(*ii*) Excess-3 = BCD + 3

= 1001 0101 0111 1011

(*iii*) 2421 code

| 2 | 4 | 2 | 1 | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 1 | 5 |
| 1 | 1 | 0 | 0 | 6 |
| 1 | 1 | 0 | 1 | 7 |
| 1 | 1 | 1 | 0 | 8 |
| 1 | 1 | 1 | 1 | 9 |

6248 = 1100 0010 0100 1110

**Example - 1.15** The state of a 12-bit register is 1000100101111. What is its content if it represents?

(*i*) Three decimal digits in BCD?

(*ii*) Three decimal digits in Excess-3 code?

***Solution:***

(*i*) In BCD ⇒ 1000 1001 0111 ; Decimal digits = 897

(*ii*) In Excess-3 ⇒ 1000 1001 0111; Decimal digits = 564

$$\underset{8-3=5}{\downarrow} \quad \underset{9-3=6}{\downarrow} \quad \underset{7-3=4}{\downarrow}$$

### 1.2.3 Gray Code

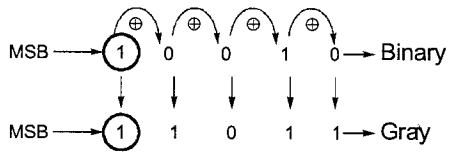
- It is a very useful code also called "minimum change codes" in which only one bit in the code group changes when going from one step to the next.
- It is also known as "Reflected code".
- It is an unweighted code, meaning that the bit positions in the code groups do not have any specific weight assigned to them.
- This code is not well suited for arithmetic operations but it finds application in input/output devices.
- These are used in instrumentation such as shaft encoders to measures angular displacement or in linear encoders for measurement of linear displacement.

**Binary-to-Gray Conversion**

- 'MSB' in the gray code is same as corresponding digit in binary number.
- Starting from "Left to Right", add each adjacent pair of binary bits to get next gray code bit. (Discard the carry if generated).

**Example - 1.16** Convert $(10010)_2$ to gray code.

***Solution:***



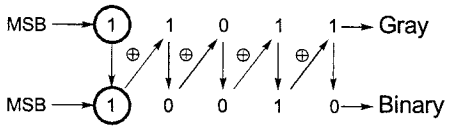

∴ $(10010)_2 = (11011)_{Gray}$

**Gray-to-Binary Conversion**

- "MSB" of Binary is same as that of gray code .
- Add each binary bit to the gray code bit of the next adjacent position (discard the carry if generated), to get next bit of the binary number.

**Example - 1.17** Convert $(11011)_{Gray}$ to Binary code.

***Solution:***




∴ $(11011)_{Gray} = (10010)_2$

**Various Binary Codes**

**Table-1.2** *Various binary codes*

| Decimal Number | Binary | | | | BCD | | | | Excess-3 | | | | Gray | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $B_3$ | $B_2$ | $B_1$ | $B_0$ | $D$ | $C$ | $B$ | $A$ | $E_3$ | $E_2$ | $E_1$ | $E_0$ | $G_3$ | $G_2$ | $G_1$ | $G_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 | | | | | | | | | 1 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 | | | | | | | | | 1 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | | | | | | | | | 1 | 0 | 1 | 0 |
| 13 | 1 | 1 | 0 | 1 | | | | | | | | | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | | | | | | | | | 1 | 0 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | | | | | | | | | 1 | 0 | 0 | 0 |

## 1.3 Arithmetic Operations

We are all familiar with the arithmetic operations like addition, subtraction, multiplication and division using decimal numbers. Such operations can also be performed on digital numbers.

### 1.3.1 Binary Addition

$$0 + 0 = 0; \quad 0 + 1 = 1$$
$$1 + 0 = 1; \quad 1 + 1 = 10$$

For example:

Add the binary numbers 110110 and 101101

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

Carry

Arrow indicates the carry operation

### 1.3.2 Binary Subtraction

$$0 - 0 = 0; \quad 10 - 1 = 1 \text{ (Borrow)}$$
$$1 - 0 = 1; \quad 1 - 1 = 0$$

While subtracting a large number from a smaller number, we can subtract the smaller from the larger and change the sign.

For example:

Subtract two binary numbers 11011 and 10110.

$\Rightarrow$

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

→ Represents borrow

### 1.3.3 Binary Multiplication

Multiply two binary numbers 1010 and 101

$\Rightarrow$

$$\begin{array}{r} 1010 \times 101 \\ \hline 1010 \\ 000\times \\ 1010\times \\ \hline 110010 \end{array}$$

**Example - 1.18** If $(10W1Z)_2 \times (15)_{10} = (Y01011001)_2$ then find the value of $W$, $Y$, $Z$.

***Solution:***

$$\begin{array}{r} 10W1Z \times 1111 \\ \hline 10W1Z \longrightarrow 1 \\ 10W1Z\times \\ 10W1Z\times \\ 10W1Z\times \\ \hline 101011001 \end{array}$$

$\ominus\ (15)_{10} = (111)_2$

Also,
$W = 1$

$\therefore W = Y = Z = 1$

**Example - 1.19** $(1111)_2 \times (1111)_2 = ?$

***Solution:***

$$\begin{array}{r} 1111 \times 1111 \\ \hline 1111 \\ 1111\times \\ 1111\times \\ 1111\times \\ \hline 11100001 \end{array}$$

### 1.3.4 Octal Addition

$0 + 0 = 0$

$0 + 2 = 2$

$1 + 6 = 7$

$1 + 7 = 0$ with carry = 1

Whenever the generated number is greater than 7 then, after decimal addition it should be converted into octal.

For example:

$7 + 2 = 9$

$$\begin{array}{r|l} 8 & 9 \\ \hline & 1 \to 1 = 11 \end{array}$$

$7 + 17 = 26$

$$\begin{array}{r|l} 8 & 14 \\ \hline & 1 \to 6 = 16 \end{array}$$

**Example - 1.20** Add two octal numbers 567 and 243.

***Solution:***

$$\Rightarrow \quad \begin{array}{r} {\scriptstyle 1\,1\phantom{0}} \\ 567 \\ +\,243 \\ \hline 1032 \end{array}$$

Here,

$7 + 3 = 10$

| 8 | 10 |
|---|---|
| | $1 \rightarrow 2$ |

$10 + 1 = 11$

| 8 | 11 |
|---|---|
| | $1 \rightarrow 3$ |

## 1.3.5 Octal Subtraction

Let, the two octal numbers to be subtracted are 723 and 564.

Then,

$$\Rightarrow \quad \begin{array}{r} {\scriptstyle 8\phantom{0}\,8} \\ 723 \\ -\,564 \\ \hline 137 \end{array}$$

Represents the borrow

## 1.3.6 Hexadecimal Addition

$1 + 1 = 2$

$1 + 9 = A$

$1 + 15 = 0$ with carry '1'

$A + A = 14$

| 16 | 20 |
|---|---|
| | $1 \rightarrow 4$ |

**Example - 1.21** Add two Hexadecimal numbers ADD + DAD = ?

***Solution:***

$$\Rightarrow \quad \begin{array}{r} {\scriptstyle 1\,1\phantom{0}} \\ ADD \\ +\,DAD \\ \hline 188A \end{array}$$

| 16 | 26 |
|---|---|
| | $1 \rightarrow A$ |

| 16 | 24 |
|---|---|
| | $1 \rightarrow 8$ |

| 16 | 24 |
|---|---|
| | $1 \rightarrow 8$ |

## 1.3.7 Hexadecimal Subtraction

Subtract 974B to 587C

$$\Rightarrow \quad \begin{array}{r} {\scriptstyle 16\,16\,16} \\ 974B \\ -\,587C \\ \hline 3ECF \end{array}$$

Represents borrow

## 1.3.8 BCD Addition

- Addition is the most important operation because subtraction, multiplication, and division can all be done by a series of additions or two's-compliment additions.
- The procedure for BCD addition is as follows:
  (*i*) Add the BCD numbers as regular true binary numbers.

(*ii*) If the sum is 9(1001) or less, it is a valid BCD answer; leave it as it is.

(*iii*) If the sum is greater than 9 or if there is a carry-out of the MSB, it is an invalid BCD number.

(*iv*) If it is invalid, add 6 (0110) to the result to make it valid. Any carry-out of the MSB is added to the next-more-significant BCD number.

(*v*) Repeat steps 1 to 4 for each group of BCD bits.

**Example-1.22** Convert the decimal number $(76)_{10}$ and $(94)_{10}$ in BCD and add them. Convert the result back to decimal to check the answer.

***Solution:***

$$\begin{array}{rcl} 76 = & 0111 & 0110 \\ +94 = & 1001 & 0100 \\ \hline & \boxed{10000} & \boxed{1010} \end{array}$$

From the rule we add (0110)

$$\begin{array}{rr} 10000 & 1010 \\ +0110 & 0110 \\ \hline 10111 & 0000 \end{array}$$

In BCD $\Rightarrow (1\,7\,0)_{10}$

Also, we have,

$$\begin{array}{r} (76)_{10} \\ +(94)_{10} \\ \hline (170)_{10} \end{array}$$

## 1.4 Signed Number Representation

So far, we have considered only positive numbers but the representation of negative number is also equally important. There are basically two ways of representing signed number as

- Sign magnitude representation.
- *r*'s compliment representation.

### 1.4.1 Sign Magnitude Representation

- In sign magnitude form, the most significant bit (MSB) is used to represent sign (0 for positive and 1 for negative number) and the remaining bits are used to represent the magnitude of the number. For example, a 4-bit binary number 0011 represents a positive number (3) and 1011 represents a negative number (–3).
- In general maximum positive number that can be represented using sign magnitude form is $(2^{n-1}-1)$ and the maximum negative number that can be represented is $-(2^{n-1}-1)$ ; where *n* is the number of bits.

### 1.4.2 r's Compliment Representation

In *r*'s compliment representation '*r*' represents the radix. It can be divided as

- $(r-1)$'s compliment.
- *r*'s compliment.

Depending upon the radix or base of different number systems the compliment representations are

- Binary number system: → 1's complement, → 2's complement
- Decimal number system: → 9's complement, → 10's complement

- Octal number system: → 7's complement, → 8's complement
- Hexadecimal number system: → F's complement, → 16's complement
- To determine the $(r-1)$'s compliment subtract the given number from the maximum possible number in the given base (i.e. $(2^n - 1)$) number in binary.
  for example, for 4 bit maximum possible number $= 2^4 - 1 = (15)_{10} = (1111)_2$

**Example - 1.23** Determine 9's compliment of a decimal number 2689?

***Solution:***

For a decimal number maximum possible number of 4 digit is 9999

$$\therefore \text{ 9's compliment of 2689 is } \begin{array}{r} 9999 \\ -\,2689 \\ \hline 7310 \end{array}$$

**Example - 1.24** Determine 7's compliment of octal number 5674?

***Solution:***

⇒ For an octal number, maximum possible number, maximum possible number of 4 digit is 7777.

$$\therefore \text{ 7's compliment of 5674 is } \begin{array}{r} 7777 \\ -\,5674 \\ \hline 2103 \end{array}$$

**Example - 1.25** Determine F's compliment of HEX = 2689.

***Solution:***

$$\text{FFFF} - 2689 = \text{D976}$$

**NOTE:** To determine the $r$'s compliment, first write $(r-1)$'s compliment of given number then add 1 in the least significant position.

**Example - 1.26** Determine 8's compliment of an octal number 2670?

***Solution:***

$$\begin{array}{r} 7777 \\ -\,2670 \\ \hline \end{array}$$

7's complement ⇒ 5107

Now, 8's compliment = 7's compliment + 1

$$= 5107 + 1 = 5108 \text{ octal} = \underset{\substack{\uparrow \\ \text{Octal}}}{5108} = 5110$$

**Example - 1.27** Given that $(EOB)_H - (ABF)_H = Y$. The radix 8's compliment of $Y$ is

(a) 844
(b) 1514
(c) 6264
(d) 6263

***Solution: (c)***

$$(EOB)_H - (ABF)_H = (34C)_H$$
$$(34C)_H = (001101001100)_2 = (1514)_8$$
$$7\text{'s compliment} = (6263)_8$$
$$8\text{'s compliment} = (6263)_8 + 1 = (6264)_8$$

**One's Compliment Representation**

- In a binary number if we replace each 0 by 1 and each 1 by 0, the obtained binary number is called one's compliment of the given binary number.
- For example, while $(0111)_2$ represents $(+7)_{10}$ and $(1000)_2$ represents $(-7)_{10}$. Here, the MSB denotes the sign of the number (0 for positive and 1 for of negative number is obtained by taking one's compliment of that number).
- Thus, the maximum positive and negative number that can be represented using one's compliment are $(2^{n-1} - 1)$ and $-(2^{n-1} - 1)$ respectively.

**Example - 1.28** The one's compliment representation of a binary number 101101 is

***Solution:***

Method 1:

One's compliment can be obtained by subtracting the given number with 111111

$$\begin{array}{l} 111111 \\ \therefore\ \underline{101101} \\ 010010 \end{array} \Rightarrow \text{One's complement}$$

Method 2:

One's compliment of the given number can be obtained by replacing each 0 with 1 and each 1 with 0.

$\therefore$ $101101 \rightarrow 010010$

**Two's Compliment Representation**

- By adding a '1' to the one's compliment representation of a binary number we can get the 2's compliment of that number.
- For $n$-bit number, the maximum positive number that can be represented is $(2^{n-1} - 1)$ and the maximum negative number that can be represented is $-(2^{n-1})$.

**Example - 1.29** The two's compliment of binary number 10110.11 is

***Solution:***

One's compliment of the given binary number can be obtained by replacing its 0 by 1 and 1 by 0.

$\therefore$ $10110.11 \rightarrow 01001.00$

Two's compliment can be obtained by adding 1 at the LSB of one's complimented representation of the given number.

$$01001.00 + 1 = 01001.01$$

**Remember**

**Special case of 2's compliment representation:** Whenever a signed number has a 1 in the sign bit and all 0's for the magnitude bits, the decimal equivalent is $-2^n$, where $n$ is the number of bits in the magnitude. For example

$$(1000)_2 = -8 \text{ and } (10000)_2 = -16$$

**Characteristics of 2's compliment representation:**

- There is only one way to represent a zero.
- The 2's compliment of 0 is 0.
- In 2's compliment representation, to extend the number of bits copy MSB bit only.

  For example

  $-7 \rightarrow 1001$

  $-7 \rightarrow 111001$

  $-7 \rightarrow 1111001$

- The 2's compliment representation of a 2's complimented binary number is equal to the binary number itself.

  For example

  Binary number of 7 = 0111.

  Two's compliment of 7 is 1001

  Two's compliment of 1001 is
  $$\begin{array}{r} 0110 \\ +\quad 1 \\ \hline 0111 \end{array} \Rightarrow 7$$

- A negative number may be converted into a positive number by finding its two's compliment.

**Signed Binary Numbers**

**Table-1.3**

| Decimal | Binary | Sign Magnitude Form | 1's compliment form | 2's compliment form |
|---|---|---|---|---|
| +7 | 0111 | 0111 | 0111 | 0111 |
| +6 | 0110 | 0110 | 0110 | 0110 |
| +5 | 0101 | 0101 | 0101 | 0101 |
| +4 | 0100 | 0100 | 0100 | 0100 |
| +3 | 0011 | 0011 | 0011 | 0011 |
| +2 | 0010 | 0010 | 0010 | 0010 |
| +1 | 0001 | 0001 | 0001 | 0001 |
| +0 | 0000 | 0000 | 0000 | 0000 |
| –0 | – | 1000 | 1111 | – |
| –1 | – | 1001 | 1110 | 1111 |
| –2 | – | 1010 | 1101 | 1110 |
| –3 | – | 1011 | 1100 | 1101 |
| –4 | – | 1100 | 1011 | 1100 |
| –5 | – | 1101 | 1010 | 1011 |
| –6 | – | 1110 | 1001 | 1010 |
| –7 | – | 1111 | 1000 | 1001 |
| –8 | – | – | – | 1000 |

- Most of the digital systems perform subtraction by using 2's compliment method or by using 1's compliment method. The advantage of performing subtraction by using compliment method is reduction in hardware i.e. instead of using separate digital circuits for addition and subtraction, only adding circuit is needed.
- To perform substraction using 2's (or 1's) compliment method, represent both the subtrahend and the minuend by the same number of bits.
- Take the 2's (1's) compliment of the subtrahend including the sign bit. Keep the minuend in its original form, and add the 2's (or 1's) compliment of the subtrahend to it.

- In 2's compliment subtraction if carry is generated ignore it. If the MSB of the sumterm is a '0' the result is in its true binary form, however if the MSB is a '1' (whether there is a carry or not), the result is negative and is in its 2's compliment form in order to get the magnitude in binary we have to take 2's compliment of it.

**Example - 1.30** Subtract 17 from 46 using 8-bit 2's compliment method.

***Solution:***

⇒ Binary form of 17 → 00010001
2's compliment of 17 → 11101111
Binary form of 46 → 00101110

$$\begin{array}{r} +46 \\ -17 \\ \hline 29 \end{array} \Rightarrow \begin{array}{r} 00101110 \\ +\ 11101111 \\ \hline 1\ 00011101 \end{array}$$

Carry (ignore it)

Here, MSB is '0' so, the result is positive and in its normal binary form. Therefore, the result is 00011101 = 29.

**Example - 1.31** Subtract decimal number 22 from 17 using 8 bit 2's compliment method.

***Solution:***

Binary form of 22 → 00010110
2's compliment of 22 → 11101010
Binary form of 17 → 00010001

$$\therefore \quad \begin{array}{r} 17 \\ -22 \\ \hline -5 \end{array} \Rightarrow \begin{array}{r} 00010001 \\ +\ 11101010 \\ \hline 11111011 \end{array}$$

In the above addition, there is no carry. The MSB of the result is '1' hence, it is a negative number and in its 2's compliment form.

∴ 2's compliment (11111011) = 00000101 = $(5)_{10}$

Therefore the answer is –5 in decimal representation.

- In 1's compliment subtraction, if carry is present, bring the carry around and add it to the LSB. This is called the "end around carry". If the MSB of the result is '0' is in its binary form. However, if the MSB of the result is '1', the result is negative and is in its 1's compliment form. To get the magnitude in binary take its 1's compliment.

**Example - 1.32** Subtract 25 from 14 using 8-bit 1's compliment method.

***Solution:***

(*i*)

$(25)_{10} \rightarrow (00011001)_2$
$(14)_{10} \rightarrow (00001110)_2$
1's compliment of $(14)_{10} \rightarrow (11110001)_2$

$$\begin{array}{r} 25 \\ -14 \\ \hline 11 \end{array} \Rightarrow \begin{array}{r} 00011001 \\ +\,11110001 \\ \hline 100001010 \\ +\,1 \\ \hline 00001011 \end{array}$$

Carry

Since MSB is '0' means, the result is positive and is in pure binary form. Therefore, the result is $(00001011)_2 = (+11)_{10}$

***(ii)*** 1's compliment of $(25)_{10} \rightarrow 11100110$

$$\begin{array}{r} -25 \\ 14 \\ \hline -11 \end{array} \Rightarrow \begin{array}{r} 11100110 \\ +\,00001110 \\ \hline 11110100 \end{array}$$

There is no carry. The MSB is a '1' so, the result is negative and is in 1's compliment form. Therefore the result is

1's compliment of (11110100) = 00001011 = –11

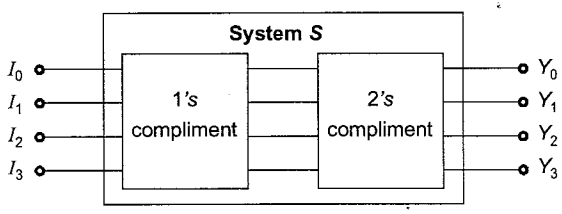
**Example - 1.33** A system has following negative numbers stored in binary form as shown. The wrongly stored number is

(a) –32 as 11100000
(b) –37 as 11011011
(c) –48 as 11101000
(d) –89 as 10100111

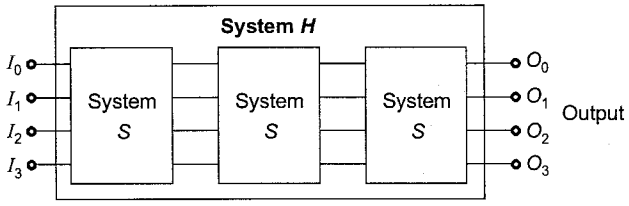
***Solution: (c)***

$$48_{10} = (00110000)_2$$
$$-48_{10} = \text{2's compliment of } 00110000$$
$$= 11001111 + 1 = 11010000$$

**Example - 1.34** Consider a System *S* as shown in the figure below:




System *S* performs 1's compliment of the input and then 2's compliment to produce output. A new System *H* is designed in which 3 System *S* are cascaded




the applied input $(I_3\, I_2\, I_1\, I_0)$ is 1010, then the output $(O_3\, O_2 O_1\, O_0)$ is

(a) 1010
(b) 0101
(c) 1101
(d) 1100

***Solution: (c)***

Let a number $N$ is given to the system

output after 1's compliment = $15 - N$

output after 2's compliment = $16 - 15 + N = N + 1$

3 such systems are connected in cascade.

So, Final output = Input $+ (3)_{10} = 1010 + 0011 = 1101$

## 1.5 Over Flow Concept

- If we add two same sign numbers and in the result if sign is opposite then it indicates "OVERLFLOW".
- When "OVERFLOW" occurs, then number of bits being increased by "1-bit in MSB".

### 1.5.1 Overflow Condition

- If $X$ and $Y$ are the MSB's of two number and $Z$ is the resultant MSB after adding two numbers then overflow condition is, $\boxed{\bar{X}\bar{Y}Z + XY\bar{Z}}$
- In 2's compliment arithmetic operation, if carry in and carry out from last bit position are different then overflow will occur.

### 1.5.2 EX-OR Logic Diagram for overflow

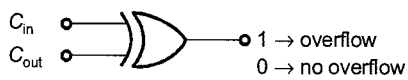



***Figure-1.1***

[Since, $A \oplus A = 0$ and $A \oplus \bar{A} = 1$]

**Example - 1.35** The number of digit 1 present in the binary representation of $3 \times 512 + 7 \times 64 + 5 \times 8 + 3$ is

(a) 8 (b) 9

(c) 10 (d) 12

***Solution: (b)***

Given, $3 \times 512 + 7 \times 64 + 5 \times 8 + 3 = (3753)_8 \rightarrow (011\ 111\ 101\ 011)_2$

Number of digit 1 is $2 + 3 + 2 + 2 = 9$.

Therefore, the number of digit 1 is 9.

**Example - 1.36** Which one of the following is the correct sequence of the numbers represented in the series $(2)_3$, $(10)_4$, $(11)_5$, $(14)_6$, $(22)_7$, ...?

(a) 2, 3, 4, 5, 6, ... (b) 2, 4, 6, 8, 10, ...

(c) 2, 4, 6, 10, 12, ... (d) 2, 4, 6, 10, 16, ...

***Solution: (d)***

Converting into decimal,

$$(2)_3 = 2 \times 3^0 = 2$$
$$(10)_4 = 1 \times 4^1 + 0 \times 4^0 = 4$$
$$(11)_5 = 1 \times 5^1 + 1 \times 5^0 = 6$$
$$(14)_6 = 1 \times 6^1 + 4 \times 6^0 = 10$$
$$(22)_7 = 2 \times 7^1 + 2 \times 7^0 = 16$$

**Example - 1.37** The base of the number system for the addition operation 24 +14 = 41 to be true is

(a) 8 (b) 7
(c) 6 (d) 5

***Solution: (b)***

Let the base is x, then

$$(24)_x + (14)_x = (41)_x$$
$$(2x^2 + 4x) + (x^2 + 4x) = 4x^2 + x$$
$$3x^2 + 8x = 4x^2 + x$$
$$\Rightarrow \quad x^2 = 7x \quad \Rightarrow \quad x = 7$$

**Example - 1.38** $X$ = 01110 and $Y$ = 11001 are two 5-bit binary numbers represented in two's compliment format. The sum of $X$ and $Y$ represented in two's compliment format using 6 bits is

(a) 100111 (b) 001000
(c) 000111 (d) 101001

***Solution: (c)***

$$X = 01110$$
$$Y = 11001$$
$$X + Y = \underline{00111}$$

Carry is discarded in the addition of numbers represented in 2's compliment form. $X + Y$ in 6 bits is 000111.

- BCD code is used in calculators, counters, digital voltmeters, digital clocks etc.
- BCD code has the advantage that it can be easily converted to and from the decimal code.
- Boudot's code is used in *Telegraphy*.
- Gray code is also known as "*minimum change code*".
- If there are "*m*" *error bits* then *HAMMING distance* is $(2m + 1)$.
- A group of bits is called "*word*".
- "CHUNKING" is replacing a longer string by a shorter one.
- The largest number that can be represented by using *N-bits* is $(2^N - 1)_{10}$.
- In self complimented weighted code, the sum of each weightage is equal to 9.
- The weighted codes 2421, 3321, 4311 and 5211 are also self complimented code.
- Excess-3 code is unweighted self complimentary code.
- Integer part will increase whenever there is conversion from higher base to lower base and vice-versa.
- Fraction part will decrease whenever there is a conversion from higher base to the lower base and vice-versa.

## Student's Assignments 1

**Q.1** List out the rules for the BCD (Binary Coded Decimal) addition with corresponding examples?

**Q.2** For two binary numbers $X$ = 1010100 and $Y$ = 1000011. Perform the subtraction.
(*i*) $X - Y$
(*ii*) $Y - X$ by using 2's compliment method.

**Q.3** What are error-correcting codes? For Hamming code write an expresssion for describing the location of possible error. Also, find out the value of '$K$' for converting BCD code into Hamming code and the bit positions of the resulting Hamming code?

***Answer: (Conventional)***

**2.** (*i*) (0010001) (*ii*) –(0010001)

## Student's Assignments 2

**Q.1** If we convert a binary sequence, $(1100101{\cdot}1011)_2$ into its octal equivalent as $(X)s$, the value of '$X$' will be
(a) (145.13) (b) (145.54)
(c) (624.54) (d) (624.13)

**Q.2** A binary $(11011)_2$ may be represented by following ways:
1. $(33)_8$ 2. $(27)_{10}$
3. $(10110)_{GRAY}$ 4. $(1\,B)_H$
Which of these above is/are correct representation?
(a) 1, 2 and 3 (b) 2 and 4
(c) 1, 2, 3 and 4 (d) only 2

**Q.3** Consider $X = (54)_b$ where '$b$' is the base of the number system. If $\sqrt{X} = 7$ then base '$b$' will be
(a) 7 (b) 8
(c) 9 (d) 10

**Q.4** Regarding ASCII codes, which one of the following characteristics is NOT correct?
(a) It is an Alphanumeric code.
(b) It is an 8-bit code.
(c) It has 128 characters including control characters.
(d) The minimum distance of ASCII code is '1'.

**Q.5** Addition of all gray code to convert decimal (0 – 9) into gray code is
(a) 129 (b) 108
(c) 69 (d) 53

**Q.6** The decimal equivalent of hexadecimal number of '2A0F' is
(a) 17670 (b) 17607
(c) 17067 (d) 10767

**Q.7** A new Binary Coded Pentary (BCP) number system is proposed in which every digit of a base-5 number is represented by its corresponding 3-bit binary code. For example, the base-5 number 24 will be represented by its BCP code 010100. In this numbering system, the BCP code 100010011001 corresponds to the following number in base-5 system
(a) 423 (b) 1324
(c) 2201 (d) 4231

***Answer Key :***

**1:** (b) **2.** (c) **3.** (c) **4.** (b) **5.** (d)
**6.** (d) **7.** (d)

CHAPTER 02

# Boolean Algebra & Minimization Techniques

## Introduction

- The binary operations performed by any digital circuit with the set of elements 0 and 1, are called logical operations or logic functions. The algebra used to symbolically represent the logic function is called Boolean algebra. It is a two state algebra invented by George Boole in 1854.
- Thus, a Boolean algebra is a system of mathematic logic for the analysis and designing of digital systems.
- A variable or function of variables in Boolean algebra can assume only two values, either a '0' or a '1'. Hence, (unlike another algebra) there are no fractions, no negative numbers, no square roots, no cube roots, no logarithms etc.

## 2.1 Logic Operations

In Boolean algebra, all the algebric functions performed is logical. These actually represent logical operations. The AND, OR and NOT are the basic operations that are performed in Boolean algebra. In addition to these operations, there are some derived operation such as NAND, NOR, EX-OR, EX-NOR that are also performed in Boolean algebra. Their operations will be discussed in detail in the next chapter.

### 2.1.1 AND Operation

The AND operation in Boolean algebra is similar to the multiplication in ordinary algebra. It is a logical operation performed by AND gate.

AND operation:

$$A \cdot A = A$$
$$A \cdot 0 = 0 \longrightarrow \text{Null law}$$
$$A \cdot 1 = A \longrightarrow \text{Identity law}$$
$$A \cdot \overline{A} = 0$$

### 2.1.2 OR Operation

The OR operation in Boolean algebra is performed by OR-gate.

OR operation:

$$\boxed{\begin{aligned} A + A &= A \\ A + 0 &= A \\ A + 1 &= 1 \\ A + \bar{A} &= 1 \end{aligned}}$$

$A + 0 = A \rightarrow$ Null law

$A + 1 = 1 \rightarrow$ Identity law

### 2.1.3 NOT Operation

The NOT operation in Boolean algebra is similar to the complementation or inversion in ordinary algebra. The NOT operation is indicated by a bar (¯) or (') over the variable.

Example: $A \xrightarrow{NOT} \bar{A}$ or $A'$ (complementation law)

and $\bar{\bar{A}} = A \Rightarrow$ double complementation law.

### 2.1.4 NAND Operation

The NAND operation in Boolean algebra is performed by AND operation with NOT operation i.e. the negation of AND gate operation is performed by the NAND gate.

### 2.1.5 NOR Operation

The NOR operation in Boolean algebra is performed by OR operation with NOT operation. i.e. the negation of OR gate operation is performed by the NOR gate.

## 2.2 Laws of Boolean Algebra

The Boolean algebra is governed by certain well developed rules and laws.

### 2.2.1 Commutative Laws

- The commutative law allows change in position of AND or OR variables. There are two commutative laws.
  - (*i*) $A + B = B + A$
    Thus, the order in which the variables are ORed is immaterial.
  - (*ii*) $A \cdot B = B \cdot A$
    Thus, the order in which the variables are ANDed is immaterial.
- This law can be extended to any number of variables.

### 2.2.2 Associative Laws

- The associative law allows grouping of variables. There are two associative laws
  - (*i*) $(A + B) + C = A + (B + C)$
    Thus, the way the variables are grouped and ORed is immaterial.
  - (*ii*) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
    Thus, the way the variables are grouped and ANDed is immaterial.
- This law can be extended to any number of variables.

### 2.2.3 Distributive Laws

- The distributive law allows factoring or multiplying out of expressions. There are two distributive laws.
  - (*i*) $A(B + C) = AB + AC$
  - (*ii*) $A + BC = (A + B)(A + C)$
- This law is applicable for single variable as well as a combination of variables.

### 2.2.4 Idempotence Laws

Idempotence means the same value. There are two Idempotence laws

(*i*) $A \cdot A = A$

i.e. ANDing of a variable with itself is equal to that variable only.

(*ii*) $A + A = A$

i.e. ORing of a variable with itself is equal to that variable only.

### 2.2.5 Absorption Laws

There are two absorption laws

(*i*) $A + AB = A(1 + B) = A$

(*ii*) $A(A + B) = A$

### 2.2.6 Involutionary Law

This law states that, for any variable 'A'

$$\bar{\bar{A}} = (A')' = A$$

## 2.3 Boolean Algebric Theorems

### 2.3.1 De Morgan's Theorem

- These are very useful in simplifying expressions in which a product or sum of variables is inverted.
- De Morgan's theorem represents two of the most important rules of Boolean algebra.

(*i*) $\overline{A \cdot B} = \bar{A} + \bar{B}$

Thus, the complement of the product of variables is equal to the sum of their individual complements.

(*ii*) $\overline{A + B} = \bar{A} \cdot \bar{B}$

Thus, the complement of a sum of variables is equal to the product of their individual complements.

- The above two laws can be extend for 'n' variables as

$$\overline{A_1 \cdot A_2 \cdot A_3 \cdots A_n} = \bar{A}_1 + \bar{A}_2 + \cdots + \bar{A}_n$$

and $$\overline{A_1 + A_2 + \cdots + A_n} = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 \cdot \bar{A}_4 \cdots \bar{A}_n$$

### 2.3.2 Transposition Theorem

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

Proof:

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

### 2.3.3 Consensus Theorem/Redundancy Theorem

- This theorem is used to eliminate redundant term.
- A variable is associated with some variable and its compliment is associated with some other variable and the next term is formed by the left over variables, then the term becomes redundant.
- It is applicable only if a Boolean function,
  - (*i*) Contains 3-variables.
  - (*ii*) Each variable used two times.
  - (*iii*) Only one variable is in complemented or uncomplemented form.

Then, the related terms to that complemented and uncomplemented variable is the answer.

- Consensus theorem can be extended to any number of variables.

e.g. $AB + \bar{A}C + BC = AB + \bar{A}C$

**Example-2.1** $AB + B\bar{C} + AC = AC + B\bar{C}$

***Solution:***

Proof:

$$\text{LHS} = AB + B\bar{C} + AC$$
$$= AB(C+\bar{C})B\bar{C}(A+\bar{A}) + A(B+\bar{B})$$
$$= ABC + AB\bar{C} + AB\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}C$$
$$= ABC + AB\bar{C} + \bar{A}B\bar{C} + A\bar{B}C$$
$$= AC(B+\bar{B}) + B\bar{C}(A+\bar{A}) = AC + B\bar{C} = \text{RHS}$$

**Example-2.2** $(A + B)(\bar{B} + C)(C + A) = (A + B)(\bar{B} + C)$

***Solution:***

Proof:

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

### 2.3.4 Duality Theorem

- It is one of the elegant theorems proved in advance mathematics.
- "Dual expression" is equivalent to write a negative logic of the given boolean relation. For this we,
  - (*i*) Change each OR sign by an AND sign and vice-versa.
  - (*ii*) Compliment any '0' or '1' appearing in expression.
  - (*iii*) Keep literals as it is.

### 2.3.5 Complementary Theorem

- For obtaining complement expression we
  - (*i*) Change each OR sign by AND sign and vice-versa.
  - (*ii*) Complement any '0' or '1' appearing in expression.
  - (*iii*) Complement the individual literals.

### 2.3.6 Summary Table

**Table-2.1**

| Theorem No. | Theorem |
|---|---|
| 1 | $A+0=A$ |
| 2 | $A\cdot 1=A$ |
| 3 | $A+1=1$ |
| 4 | $A\cdot 0=0$ |
| 5 | $A+A=A$ |
| 6 | $A\cdot A=A$ |
| 7 | $A+\bar{A}=1$ |
| 8 | $A\cdot\bar{A}=0$ |
| 9 | $A\cdot(B+C)=AB+AC$ |
| 10 | $A+BC=(A+B)(A+C)$ |
| 11 | $A+AB=A$ |
| 12 | $A(A+B)=A$ |
| 13 | $A+\bar{A}B=(A+B)$ |
| 14 | $A(\bar{A}+B)=AB$ |
| 15 | $AB+A\bar{B}=A$ |
| 16 | $(A+B)\cdot(A+\bar{B})=A$ |
| 17 | $AB+\bar{A}C=(A+C)(\bar{A}+B)$ |
| 18 | $(A+B)(\bar{A}+C)=AC+\bar{A}B$ |
| 19 | $AB+\bar{A}C+BC=AB+\bar{A}C$ |
| 20 | $(A+B)(\bar{A}+C)(B+C)=(A+B)(\bar{A}+C)$ |
| 21 | $\overline{A\cdot B\cdot C\cdot\ldots}=\bar{A}+\bar{B}+\bar{C}+\ldots$ |
| 22 | $\overline{A+B+C+\ldots}=\bar{A}\cdot\bar{B}\cdot\bar{C}\ldots$ |

**Example - 2.3** Write complement function of, $f=A\bar{B}C+\bar{A}B\bar{C}+A\bar{B}\bar{C}$

***Solution:***

$$\bar{f} = \text{Compliment of } f$$

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

**NOTE**

- For any logical expression, if we take two times Dual, we get same given expression as previous.
- For 1-*time Dual*, if we get same function or expression it is called "*Self Dual Expression*".
- With *N-variables*, maximum possible *Self-Dual Function* = $(2)^{2^{n-1}} = 2^{(2^n/2)}$.
- Remember that with *N-variables, maximum possible distinct logic functions* = $2^{2^n}$.

## 2.4 Minimization of Boolean Functions

Every Boolean expression function must be reduced to as simple form as possible before realization, because every logic operation in the expression represents a corresponding element of hardware. Realization of a digital circuit with minimal expression has several advantages as:

- The number of logic gates will reduce.
- Power dissipation will decrease.
- The complexity of the circuit reduces
- The speed of operation will increase.
- The fan in may reduce.

The simple method of minimization of Boolean functions using certain algebraic rules which results in the reduction of number of terms and/or number of arithmetic operations. The various theorems and rules that are already discussed are very useful for the simplification of Boolean expressions.

**Example - 2.4** Simplify the Boolean function $F = (\overline{A + \overline{BC}})(A\bar{B} + ABC)$.

***Solution:***

Using De' Morgan's theorem $(\overline{A + \overline{BC}}) = (\bar{A} \cdot \overline{\overline{BC}})(A\bar{B} + ABC) = (\bar{A}BC)(A\bar{B} + ABC)$

$$= \bar{A}BCA\bar{B} + \bar{A}ABCBC = 0$$

**Example - 2.5** Let $F(A, B) = \bar{A} + B$, then the value of $f(f(x + y, y), z) = ?$

***Solution:***

$\Rightarrow$ Let

$$A = x + y$$
$$B = y$$

$\therefore$

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

## 2.5 Representation of Boolean Functions

- A function of '$n$' Boolean variables denoted by $f(A_1, A_2, \cdots A_n)$ is another variable of algebra and takes one of the two possible values either 0 or 1. The various ways of representing a given function are discussed below:

**Boolean Function Representation**

- Canonical form
  - All the terms contain all the variables either in complementary or in uncomplentary form
  - Example: $F(A, B, C) = \bar{A}BC + ABC + \bar{A}B\bar{C}$
- Minimal form
  - Minimum numbers of literal
  - Example: $F(A, B, C) = A + ABC + \bar{A}BC = A$

- The term 'literal' means a binary variable either in complementary or in uncomplementary form.

### 2.5.1 Minterms and Maxterms

- $n$-binary variables have $2^n$ possible combinations.
- Minterm is a product term, it contains all the variables either complementary or uncomplementary form for that combination the function output must be '1'.
- Maxterm is a sum term, it contains all the variables either complementary or uncomplementary form for that combination the function output must be '0'.

**For 3-Variable**

**Table-2.2**

| $A$ | $B$ | $C$ | Minterm | Maxterm |
|---|---|---|---|---|
| 0 | 0 | 0 | $m_0 = \bar{A}\bar{B}\bar{C}$ | $M_0 = A+B+C$ |
| 0 | 0 | 1 | $m_1 = \bar{A}\bar{B}C$ | $M_1 = A+B+\bar{C}$ |
| 0 | 1 | 0 | $m_2 = \bar{A}B\bar{C}$ | $M_2 = A+\bar{B}+C$ |
| 0 | 1 | 1 | $m_3 = \bar{A}BC$ | $M_3 = A+\bar{B}+\bar{C}$ |
| 1 | 0 | 0 | $m_4 = A\bar{B}\bar{C}$ | $M_4 = \bar{A}+B+C$ |
| 1 | 0 | 1 | $m_5 = A\bar{B}C$ | $M_5 = \bar{A}+B+\bar{C}$ |
| 1 | 1 | 0 | $m_6 = AB\bar{C}$ | $M_6 = \bar{A}+\bar{B}+C$ |
| 1 | 1 | 1 | $m_7 = ABC$ | $M_7 = \bar{A}+\bar{B}+\bar{C}$ |

From the above table we conclude that:

- In "Minterms" we assign '1' to each uncomplemented variable and '0' to each complemented variable.
- In "Maxterms" we assign '0' to each uncomplemented variable and '1' to each complemented variable.

### 2.5.2 Sum of Product (SOP) Form

- The SOP expression usually takes the form of two or more variables ANDed together. Each product term may be minterm or implicant.

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

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

- This form is also called the "disjunctive normal form".
- The SOP expression is used most often because it tends itself nicely to the development of truth tables and timing diagrams.
- SOP circuits can also be constructed easily by using a special combinational logic gates called the "AND-OR-INVERTER" gate.
- SOP forms are used to write logical expression for the output becoming Logic '1'.

Example:

**Table-2.3**

| Input (3-Variables) | | | Output (Y) |
|---|---|---|---|
| A | B | C | Y |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

∴ Notation of SOP expression is,

$$f(A, B, C) = \Sigma m(3, 5, 6, 7)$$

∴ $$Y = m_3 + m_5 + m_6 + m_7$$

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

### 2.5.3 Product of Sum (POS) Form

- The POS expression usually takes the form of two or more ORed variables within parentheses, ANDed with two or more such terms.

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

- This form is also called the "Conjunctive normal form".
- Each individual term in standard *POS* form is called *Maxterm*.
- *POS* forms are used to write logical expression for output be coming *Logic* '0'.

  Same as Table (2.3), we get

$$f(A, B, C) = \Pi M(0, 1, 2, 4)$$

∴ $$Y = M_0 \times M_1 \times M_2 \times M_4$$

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

- We also conclude that from Table (2.3), and from above equations

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

  then, $$Y = \Pi M(0, 1, 2, 4)$$

### 2.5.4 Standard Sum of Product Form

- In this form the function is the sum of number of product terms where each product term contains all the variables of the function, either in complemented or uncomplemented form.
- It is also called canonical SOP form or expanded SOP form.

  **Example:** Function $[Y = A + B\bar{C}]$ can be represented in canonical form as:

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

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

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

### 2.5.5 Standard Product of Sum Form

- This form is also called canonical POS form or expanded POS form.

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

Then, the canonical form of the given function

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

### 2.5.6 Truth Table Form

A truth table is a tabular form representation of all possible combinations of given function.

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

Then,

| | A | B | C | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 |

**Example-2.6** Simplify the expression for given truth table.

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

***Solution:***

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

**Example-2.7** Simplify the expression for $Y(A, B) = \Pi M(1, 3)$.

***Solution:***

$$(1)_{10} = (01)_2 = \bar{A}B\,(SOP) = A + \bar{B}\,(\text{POS})$$
$$(3)_{10} = (11)_2 = AB \cdot (SOP) = \bar{A} + \bar{B}\,(\text{POS})$$

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

### 2.5.7 Dual Form

- Dual form is used to convert positive logic to the negative logic and vice-versa.
- In positive logic system, higher voltage is taken as logic '1' and in negative logic system, higher voltage is taken as logic '0'.

For example:

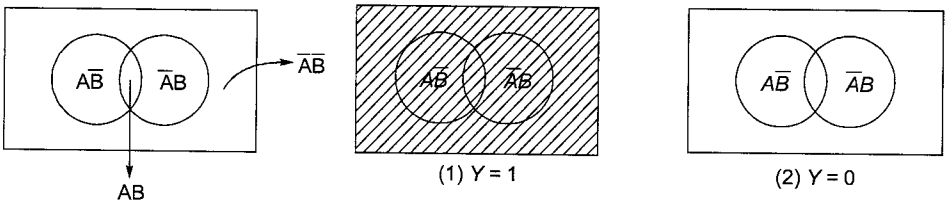
For (positive) logic

Logic '1' = 0 V; Logic '0' = –5 V

For (negative) logic

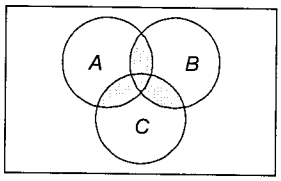
Logic '1' = –0.8 V; Logic '0' = –1.7 V

### 2.5.8 Venn Diagram Form

- A Boolean algebra can be represented by a Venn diagram in which each variable is considered as a set.
- The AND operation is considered as an intersection and the OR operation is considered as a union.
- 

(1) $Y = 1$ (2) $Y = 0$

**Example - 2.8** For the given Venn diagram minimize the expression for the shade area.




***Solution:***

The shaded area includes

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

$\therefore$

$$Y = ABC + AB\bar{C} + A\bar{B}C + \bar{A}BC$$
$$= AB + AC(B + \bar{B}) + BC(A + \bar{A}) = AB + BC + AC$$

### 2.5.9 Statement Form

**Example - 2.9** A logic circuit have three inputs $A$, $B$ and $C$ and output $Y$, $Y$ is '1' for the following combinations

(*i*) $B$ and $C$ are true (1)

(*ii*) $A$ and $C$ are false (0)

(*iii*) $A$, $B$ and $C$ are true (1)

(*iv*) $A$, $B$ and $C$ are false (0)

Then the minimized expression for $Y$ is

***Solution:***

Truth Table:

From the truth table, we get

| | $A$ | $B$ | $C$ | $Y$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 |

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

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

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

### 2.5.10 Karnaugh Map

- The "Karnaugh map" is a graphical method which provides a systematic method for simplifying and manipulating the Boolean expressions or to convert a truth table to its corresponding logic circuit in a simple, orderly process.
- In this technique, the information contained in a truth table or available in SOP or POS form is represented on K-map.
- Although this technique may be used for any number of variables, it is generally used up to 6-variables beyond which it becomes very cumbersome.
- In n-variable *K*-map there are $2^n$ cells.
- "Gray code" has been used for the identification of cells.

**Two-variable K-Map**

- Four cells
- Four minterms (maxterms)

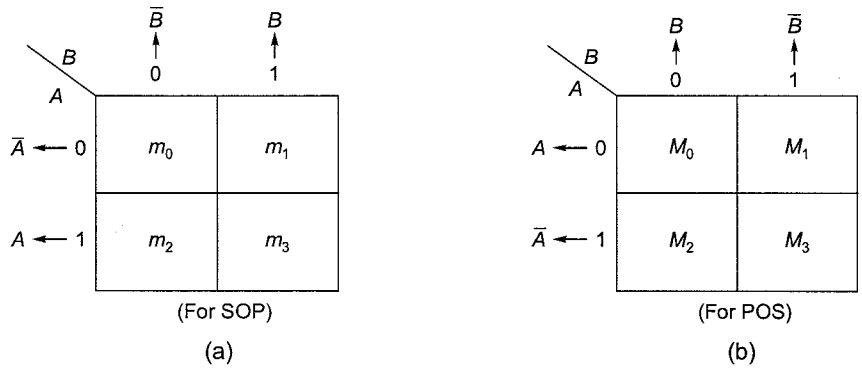



***Figure-2.1:*** *(a) and (b)*

**Three-variable K-map**

- Eight cells
- Eight minterms (maxterms)

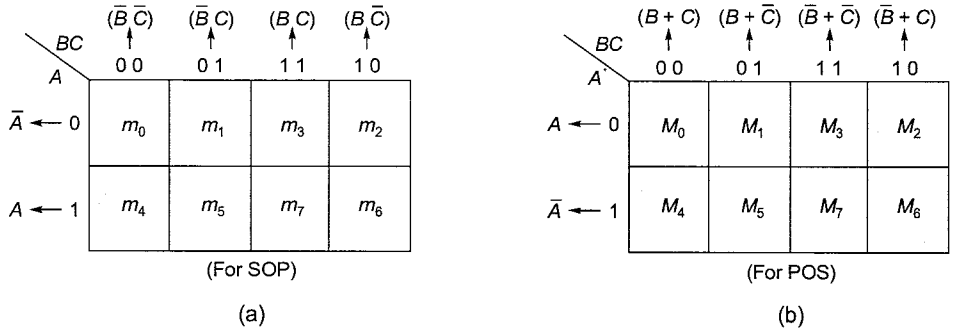



***Figure-2.2:*** *(a) and (b)*

**Four-variable K-map**

- Sixteen cells
- Sixteen minterms (maxterms)

| AB \ CD | $(\bar{C}\bar{D})$ 00 | $(\bar{C}D)$ 01 | $(CD)$ 11 | $(C\bar{D})$ 10 |
|---|---|---|---|---|
| $\bar{A}\bar{B}$ ← 00 | $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| $\bar{A}B$ ← 01 | $m_4$ | $m_5$ | $m_7$ | $m_6$ |
| $AB$ ← 11 | $m_{12}$ | $m_{13}$ | $m_{15}$ | $m_{14}$ |
| $A\bar{B}$ ← 10 | $m_8$ | $m_9$ | $m_{11}$ | $m_{10}$ |

(For SOP)

(a)

| AB \ CD | $(C+D)$ 00 | $(C+\bar{D})$ 01 | $(\bar{C}+\bar{D})$ 11 | $(\bar{C}+D)$ 10 |
|---|---|---|---|---|
| $(A+B)$ ← 00 | $M_0$ | $M_1$ | $M_3$ | $M_2$ |
| $(A+\bar{B})$ ← 01 | $M_4$ | $M_5$ | $M_7$ | $M_6$ |
| $(\bar{A}+\bar{B})$ ← 11 | $M_{12}$ | $M_{13}$ | $M_{15}$ | $M_{14}$ |
| $(\bar{A}+B)$ ← 10 | $M_8$ | $M_9$ | $M_{11}$ | $M_{10}$ |

(For POS)

(b)

***Table-2.3:*** *(a) and (b)*

**Five-variable K-map**

- 32-cells
- 32-minterms (maxterms)
- Here, we have $f(A, B, C, D, E)$

For $\boxed{A = 0}$

| BC \ DE | $\bar{D}\bar{E}$ | $\bar{D}E$ | $DE$ | $D\bar{E}$ |
|---|---|---|---|---|
| $\bar{B}\bar{C}$ | $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| $\bar{B}C$ | $m_4$ | $m_5$ | $m_7$ | $m_6$ |
| $BC$ | $m_{12}$ | $m_{13}$ | $m_{15}$ | $m_{14}$ |
| $B\bar{C}$ | $m_8$ | $m_9$ | $m_{11}$ | $m_{10}$ |

(From 0 – 15)

For $\boxed{A = 1}$

| BC \ DE | $\bar{D}\bar{E}$ | $\bar{D}E$ | $DE$ | $D\bar{E}$ |
|---|---|---|---|---|
| $\bar{B}\bar{C}$ | $m_{16}$ | $m_{17}$ | $m_{19}$ | $m_{18}$ |
| $\bar{B}C$ | $m_{20}$ | $m_{21}$ | $m_{23}$ | $m_{22}$ |
| $BC$ | $m_{28}$ | $m_{29}$ | $m_{31}$ | $m_{30}$ |
| $B\bar{C}$ | $m_{24}$ | $m_{25}$ | $m_{27}$ | $m_{26}$ |

(From 16 – 31)

(For SOP)

***Figure-2.4***

## Simplification of Logical Functions Using K-map:

Simplification of logical functions with K-map is based on the principle of combining terms in adjacent cells.

**Looping**

- The expression for output '$Y$' can be simplified by properly combining those cells in the K-map which contains 1's (for SOP form) or 0's (for POS form). The process for combining these 1's or 0's is called "Looping".
- Groups are made up of 2, 4, 8, 16 and so on.
- By folding K-map over its edges, the number of 1's or 0's overlapping forms the Group.

**Looping Groups of Two (Pairs)**

Looping a pair of adjacent 1's in a K-map eliminates the variable that appears in complemented and uncomplemented form.

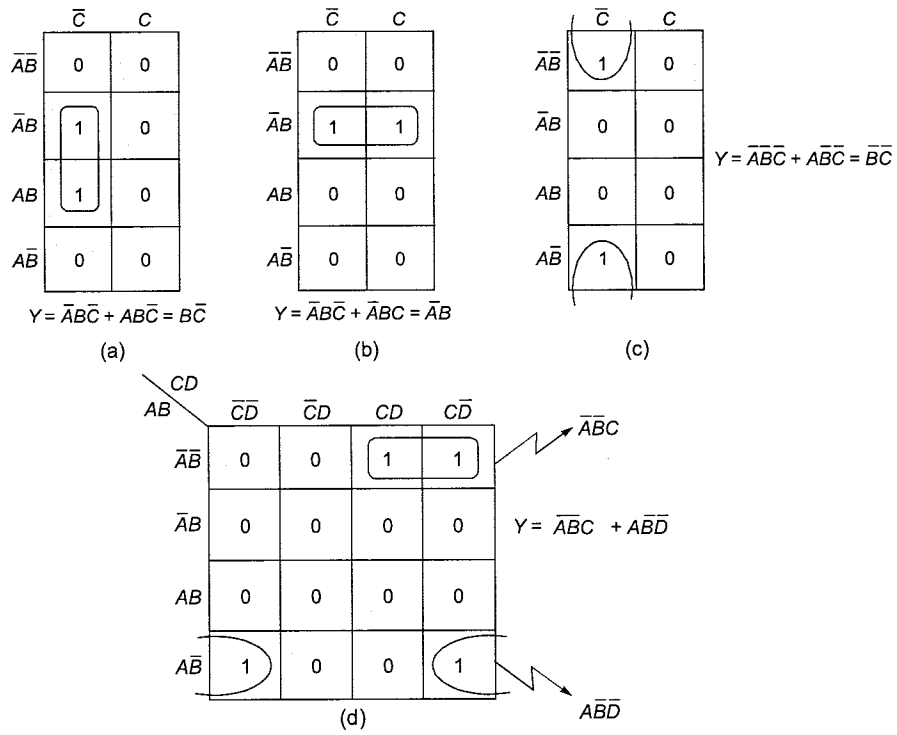



***Figure-2.5:*** *(a), (b), (c) and (d)*

**Looping Groups of Four (Quads)**

Looping a quad of 1's eliminates those two variables that appear in both complemented and uncomplemented form.

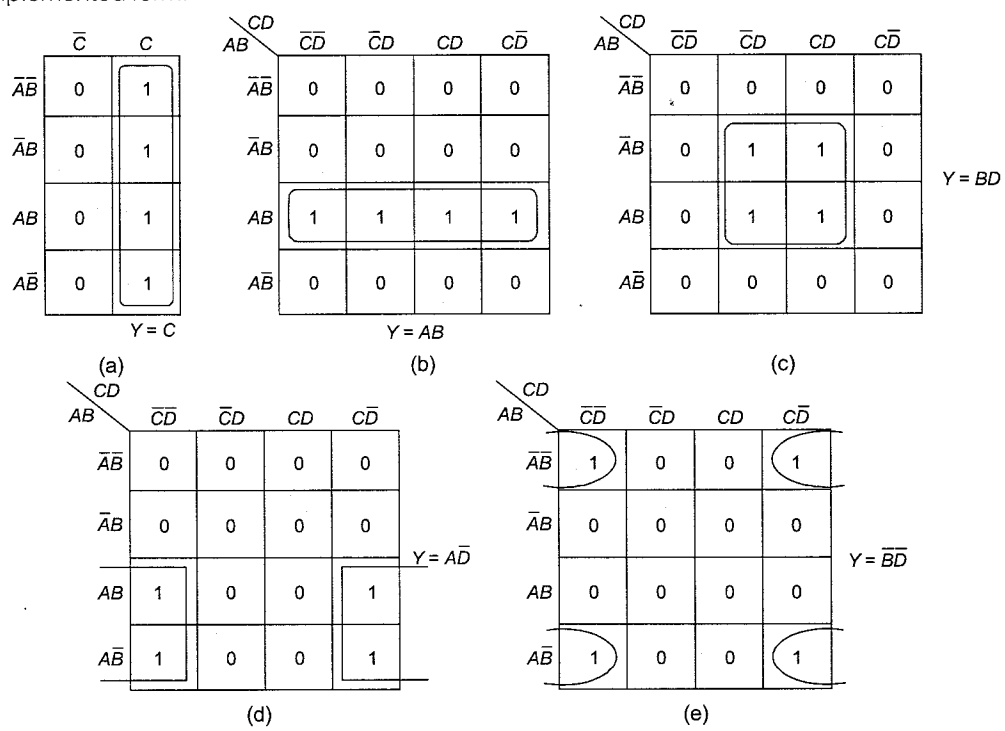



***Figure-2.6:*** *(a), (b), (c), (d) and (e)*

**Looping Groups of Eight (Octets)**

Looping an octet of 1's eliminates those three variables that appear in both complemented and uncomplemented form.

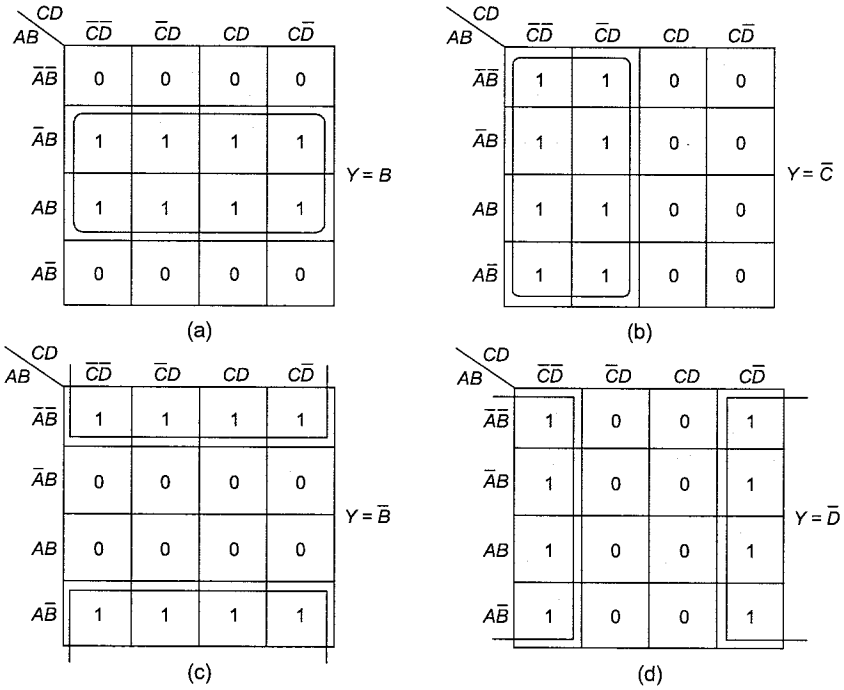



***Figure-2.7:*** *(a), (b), (c) and (d)*

**NOTE:** Those literals which are not same in the maxterm or minterms get eliminated.

## Complete Simplification Rules:

- Construct the K-map and place 1's in those cells corresponding to the 1's in the truth table. Place 0's in the other cells.
- Examine the map for adjacent 1's and loop those 1's which are not adjacent to any other 1's. These are called isolated 1's.
- Next, look for those 1's which are adjacent to only one other 1. Loop any pair containing such a 1.
- Loop any octet even it contains some 1's that have already been looped.
- Loop any quad that contains one or more 1's which have not already been looped, making sure to use the minimum number of loops.
- Loop any pairs necessary to include any 1's that have not yet been looped, making sure to use the minimum number of loops.
- Form the OR sum of all the terms generated by each loop.

**Example-2.10** Simplify a four-variable logic function using K-map.

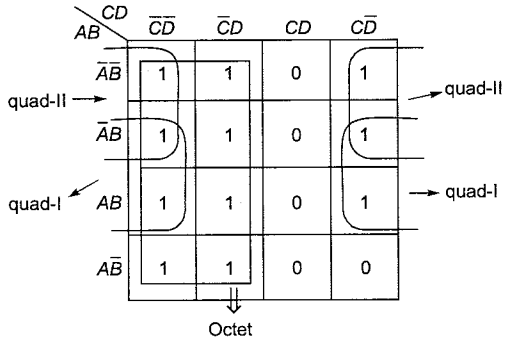
$f(A, B, C, D) = \Sigma m(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ also implement the simplified expression with AND-OR logic.

***Solution:***

Given,

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

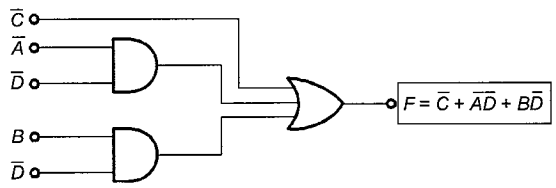
**Step-I:** Draw a 16-cell map and put 1's in such cell as given in $f(A, B, C, D)$.
**Step-II:** One group of 8, and two-groups of 4-one's are formed.
**Step-III:** Finally we write minimized SOP form.




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

Octet quad-II quad-I

⇒ Gate implementation:




**Don't Care Condition**

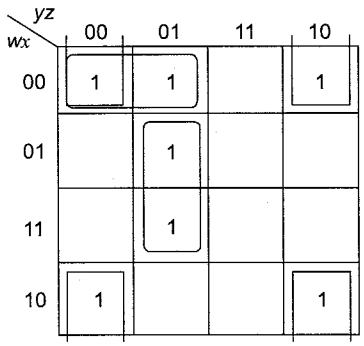
- Some logic circuits can be designed so that there are certain input combinations for which there are no specified output levels, usually because these input combinations will never occur.
- So, a circuit designer is free to make the output for any "don't care" condition either a 0 or 1 in order to produce the simplest output expression.

**Example - 2.11** Convert the Boolean function from a sum of products form to a simplified product of sums form.

$$F(w, x, y, z) = \Sigma(0, 1, 2, 5, 8, 10, 13)$$

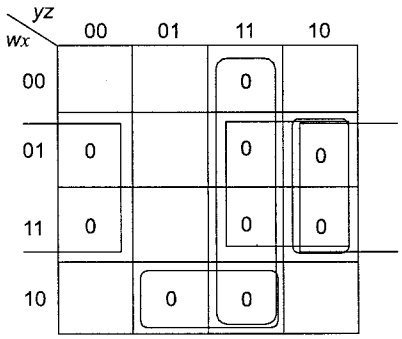
***Solution:***

Using K-Map for given function, we get




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

To convert it in POS form (using 0's), we get




$$F' = yz + xz' + xy + wx'z$$

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

**Example-2.12** Simplified the given equation in part (*i*) and (*ii*):

(*i*) In terms of SOP and don't care conditions.

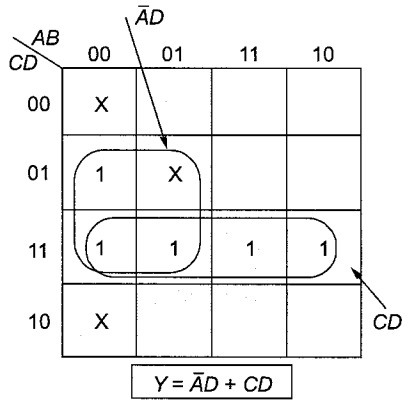
$$f(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5)$$

(*ii*) In terms of POS and don't care conditions.

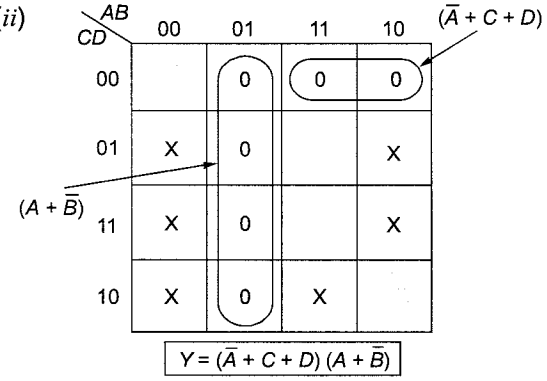
$$f(A, B, C, D) = \Pi M(4, 5, 6, 7, 8, 12).\ d(1, 2, 3, 9, 11, 14)$$

***Solution:***

(*i*)




(*ii*)




## 2.6 Implicants, Prime Implicants and Essential Prime Implicants

### Implicant

Implicant is a product term on the given function for that combination the function output must be 1.

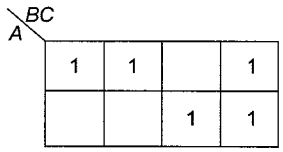
### Prime Implicant

Prime implicant is a smallest possible product term of the given function, removing any one of the literal from which is not possible.

### Essential Prime Implicant

Essential prime implicant is a prime implicant it must cover atleast one minterm, which is not covered by any other prime implicant.

**Example-2.13** For the given K-map, find implicant, Prime Implicant and Essential Prime Implicant.




***Solution:***

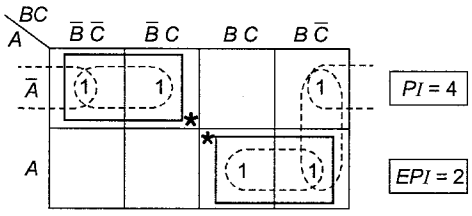
Here total number of 1's = 5

$\therefore$ Implicant = 5

$\therefore$ Implicant = $(\bar{A}\bar{B}\bar{C}), (\bar{A}\bar{B}C), (ABC), (\bar{A}B\bar{C}), (AB\bar{C})$

and Prime implicant = $\bar{A}\bar{B}, \bar{A}\bar{C}, AB, B\bar{C}$

and $EPI = \bar{A}\bar{B}, AB$



**Example-2.14** In the sum of products function

$$f(X, Y, Z) = \sum(2, 3, 4, 5),$$

the prime implicants are

(a) $\bar{X}Y + X\bar{Y}$

(b) $\bar{X}Y + X\bar{Y}\bar{Z} + X\bar{Y}Z$

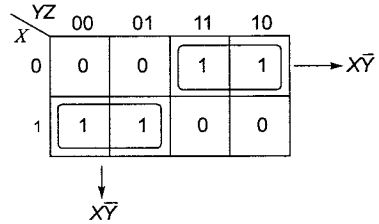
(c) $\bar{X}Y\bar{Z} + \bar{X}YZ + X\bar{Y}$

(d) $\bar{X}Y\bar{Z} + \bar{X}YZ + X\bar{Y}\bar{Z} + X\bar{Y}Z$

***Solution: (a)***

$$f(X, Y, Z) = \Sigma(2, 3, 4, 5)$$

$$\therefore \quad f(X, Y, Z) = X\bar{Y} + \bar{X}Y$$




**Example-2.15** A logic circuit implements the boolean function $F = \bar{X} \cdot Y + X \cdot \bar{Y} \cdot \bar{Z}$. It is found that the input combination $X = Y = 1$ can never occur. Taking this into account, a simplified expression for $F$ is given by

(a) $\bar{X} + \bar{Y} \cdot \bar{Z}$

(b) $X + Z$

(c) $X - Z$

(d) $Y + X \cdot \bar{Z}$

***Solution: (d)***

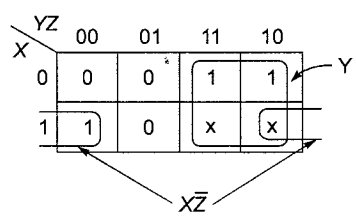
$$F = \bar{X}Y + X\bar{Y}\bar{Z} = \bar{X}Y(Z + \bar{Z}) + X\bar{Y}\bar{Z}$$

$$= \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z}$$

Truth table:

| $X$ | $Y$ | $Z$ | $F$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | $x$ |
| 1 | 1 | 1 | $x$ |

K-map:




$$\boxed{F = Y + X\bar{Z}}$$

**Example-2.16** The Boolean functions can be expressed in canonical SOP (sum of products) and POS (product of sums) form. For the functions, $Y = A + \bar{B}C$, which are such two forms

(a) $Y = \Sigma(1,2,6,7)$ and $Y = \Pi(0, 2, 4)$

(b) $Y = \Sigma(1,4,5,6, 7)$ and $Y = \Pi(0, 2, 3)$

(c) $Y = \Sigma(1,2,5,6, 7)$ and $Y = \Pi(0, 1, 3)$

(d) $Y = \Sigma(1, 2, 4, 5, 6, 7)$ and $Y = \Pi(0, 2, 3, 4)$

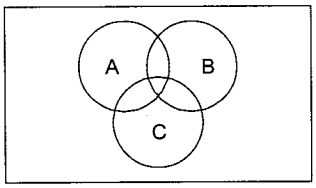
***Solution: (b)***

Plotting the 1's in Karnaugh map for the function $Y = A + \bar{B}C$ to represent SOP.

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

Therefore, according to the positions of 1's, SOP form is $Y = \Sigma(1, 4, 5, 6, 7)$.
According to the positions of 0's (positions other than 1's). POS form is $Y = \Pi(0, 2, 3)$.

**Example - 2.17** The Boolean expression for the shaded area in the Venn diagram shown is

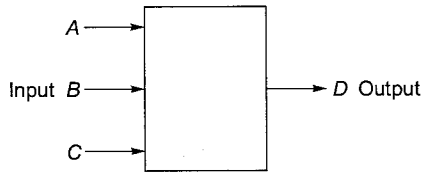



(a) $A + \bar{B} + C$
(b) $AB + \bar{A}BC$
(c) $A\bar{B}C + \bar{A}BC$
(d) $AB + \bar{A}\bar{B}C$

***Solution: (d)***

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

**Example - 2.18** For the box shown the output $D$ is true if and only if a majority of the inputs are true.




The Boolean function for the output is

(a) $D = AB\bar{C} + \bar{A}BC + A\bar{B}C$
(b) $D = ABC + \bar{A}BC + A\bar{B}C + AB\bar{C}$
(c) $D = \bar{A}\bar{B}\bar{C} + AB + AC + BC$
(d) $D = \bar{A}\bar{B}C + A\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC$

***Solution: (b)***

| A | B | C | D | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | $\Rightarrow \bar{A}BC$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $\Rightarrow A\bar{B}C$ |
| 1 | 1 | 0 | 1 | $\Rightarrow AB\bar{C}$ |
| 1 | 1 | 1 | 1 | $\Rightarrow ABC$ |

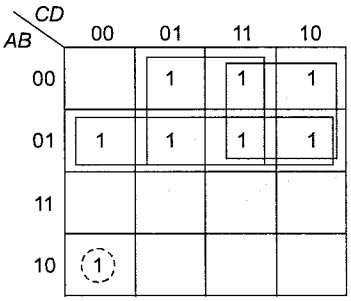
**Example - 2.19** Design a 4-bit combinational circuit 2's complimenter (i.e. the output generates 2's compliment of the input binary number). Show that the circuit can be constructed with EX-OR gates.

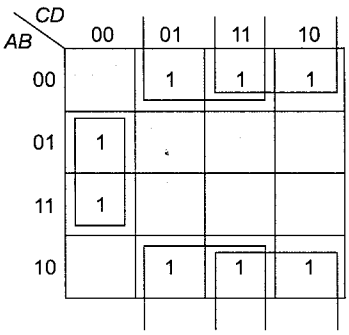
***Solution:***

Let the inputs are $A$, $B$, $C$ and $D$ and the 2's complimented outputs are $W$, $X$, $Y$ and $Z$ respectively, Therefore the truth table is

| Input | | | | Output | | | |
|---|---|---|---|---|---|---|---|
| $A$ | $B$ | $C$ | $D$ | $W$ | $X$ | $Y$ | $Z$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Using K-map



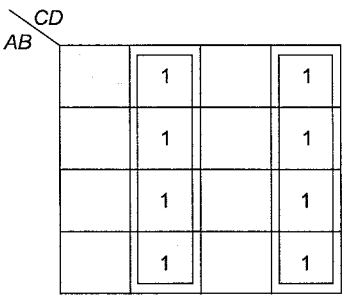



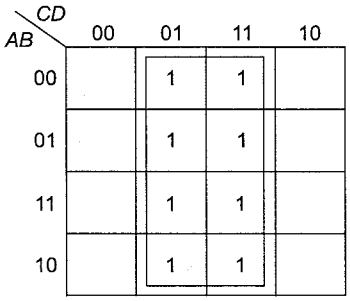

$$W = \bar{A}(B+C+D)+A\bar{B}\bar{C}\bar{D}$$

$$\Rightarrow \quad = A \oplus (B + C + D)$$

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

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




$$Y = C\bar{D}+\bar{C}D = C \oplus D$$

$$Z = D$$

Thus, the circuit can be constructed using Ex-OR gate as

$$W = A \oplus (B + C + D)$$
$$X = B \oplus (C + D)$$
$$Y = C \oplus D$$
$$Z = D$$

## Summary

- With '$n$' number of variables the maximum possible minterm or maxterm is equal to '$2^n$'.
- With '$n$' number of variables, the maximum possible logical expression is equal to '$2^{2^n}$' i.e. Boolean expression = $2^{2^n}$.
- Positive logic OR gate is equivalent to the negative logic AND gate and vice-versa.
- For 'n' number of variable the maximum possible self dual expression = $2^{2^{n-1}}$.
- Canonical form is used to expand the expressions.
- K-map will provide minimized expressions but not necessarily unique.
- In K-map, Gray code representation is used.
- The two K-maps are said to be equal if '1's are placed in the same position on both the maps. Thus, the logical expression is also same.
- The two K-maps are said to be complemented if one K-map has 1's and another K-map has 0's on the same location.

## Student's Assignments 1

**Q.1** Explain De-Morgan's first theorem using circuits of NOR gate as well as AND gate with inverted inputs. Give the relevant table using NAND gate and OR gate with inverted inputs, explain De-Morgan's second theorem.

**Q.2** Identify the number of PI and EPI for the following function as, $f(A, B, C) = \Sigma m(0, 1, 5, 7)$.

**Q.3** (*i*) What are the main advantages of K-map over the algebraic methods in digitial simplification circuitry.

(*ii*) Find the minimize form of the logical expression

$$Z = ABC + ABD + \overline{A}B\overline{C} + CD + B\overline{D},$$

by using K-map technique.

**Q.4** (*i*) The Boolean function $E = F_1 + F_2$ contains the sum of the minterms of $F_1$ and $F_2$.

(*ii*) The Boolean function of $G = F_1F_2$ contains only the minterms that are common to $F_1$ and $F_2$.

***Answer : (Conventional)***

**3.** (*ii*) $B + CD$

## Student's Assignments 2

**Q.1** The sign-magnitude form and 2's complement form of a signed binary number $(10111)_2$ are:

(a) –23 and –25 (b) –23 and –9
(c) –7 and –23 (d) –7 and –9

**Q.2** From the 7 symbols (0 – 6), how many different codes we can construct?

(a) 8! (b) 8
(c) 7! (d) 7

**Q.3** The number of digit '1' present in the binary representation of the number $(1199)_{12}$ is

(a) 7 (b) 8
(c) 10 (d) 12

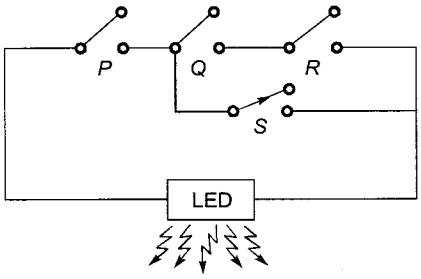
**Q.4** For an 8-bit microprocessor, the maximum possible number of self dual-functions equals to

(a) $(16)^8$ (b) $(16)^{16}$
(c) $(16)^{32}$ (d) $(16)^{64}$

**Q.5** For a 3-variable function given that
$f(A, B, C) = \Pi M(0, 1, 2, 3, 4, 5, 6, 7)$
The minimized Boolean function is

(a) $A\bar{B}C$ (b) $(A + B + C)$
(c) 0 (d) 1

**Q.6** For the switching circuit shown below, taking open as '0' and closed as '1', the expression for the circuit when LED glows is,




(a) $P + (Q + R)\,S$ (b) $P(QR + S)$
(c) $P + QR + S$ (d) LED can not glow

**Q.7** When signed numbers are used in binary arithmetic, then which one of the following notations would have unique representation for zero?

(a) Sign magnitude (b) 1's complement
(c) 2's complement (d) 9's complement

**Q.8** A signed integer has been stored in a byte using 2's complement format. We wish to store the same integer in 16-bit word. We should copy the original byte to the less significant byte of the word and fill the more significant byte with

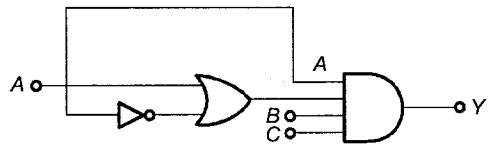
(a) 0
(b) 1
(c) equal to the MSB of the original byte
(d) complement of the MSB of the original byte.

**Q.9** A combinational circuit has input $A$, $B$, and $C$ and its K-map is as shown in figure. The output of the circuit is given by

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | 1 | | 1 |
| 01 | 1 | | 1 | |

(a) $(\bar{A}B + A\bar{B})\bar{C}$ (b) $(AB + \bar{A}\bar{B})\bar{C}$
(c) $\bar{A}\bar{B}\bar{C}$ (d) $A \oplus B \oplus C$

**Q.10** The Boolean expression for the output in the logic circuit is




(a) $A\bar{B}C$ (b) $ABC$
(c) $\bar{A}BC$ (d) $\bar{A}\bar{B}\bar{C}$

**Q.11** The result of $(54)_{10} - (54)_{16}$, expressed in 6-bit 2's complement representation is given by

(a) 1 1 1 1 1 0 (b) 1 0 1 1 1 0
(c) 1 0 0 0 1 0 (d) 1 0 0 0 1 1

***Answer Key:***

| | | | | |
|---|---|---|---|---|
| **1.** (d) | **2.** (a) | **3.** (a) | **4.** (c) | **5.** (c) |
| **6.** (b) | **7.** (c) | **8.** (c) | **9.** (d) | **10.** (b) |
| **11.** (c) | | | | |

CHAPTER 03

# Logic Gates and Switching Circuits

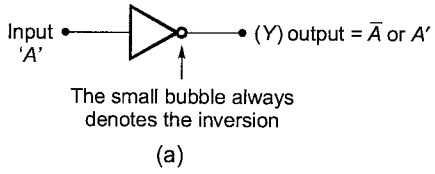
## Introduction

- Logic gates are the fundamental building blocks of any digital system. They are usually constructed in LSI and VLSI circuits along with other devices.
- The name logic gate is derived from the ability of such a device to make decisions, in the sense that, it produces outputs for different combinations of applied inputs.
- The inputs and outputs of logic gates can occur only in two levels as HIGH and LOW, MARK and SPACE, TRUE and FALSE, ON and OFF, or simply 1 and 0.
- The function of each logic gate will be represented by Boolean expression.
- Logic gates are classified as
  - Basic Gates : NOT, AND, OR
  - Universal Gate : NAND, NOR
  - Special purpose Gate : EX-OR, EX-NOR
- A "truth table" is a means for describing how a logic circuit's output depends on the logic levels present at circuit's input.
- The number of output combinations will be $2^N$ for "$N$-input" truth table.

## 3.1 Basic Gates

### 3.1.1 The NOT Gate

- The NOT gate has a single input variable and a single output variable.
- The NOT operation is also referred to as 'INVERSION' or 'COMPLEMENTATION'.
- Thus, its output logic level is always opposite to the logic level of its input.
- *Figure 3.1 (a)* and *(b)* respectively shows the logic symbol and the truth table of an inverter. However the waveform sample is shown in *Figure 3.1 (c)*.




| Input (A) | Input (Y) |
|---|---|
| 0 | 1 |
| 1 | 0 |

(b)

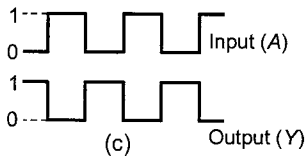



***Figure-3.1*** *The NOT Gate (a) Logical symbol (b) Truth table (c) Sample waveform*

- The symbol of NOT operation is represented by '—' (bar) or (′). Therefore, for input $A$ the output of the NOT gate is $\boxed{Y = \bar{A} = A'}$.
- The switching circuit of a NOT gate is in *Figure (3.2)*, where as the transistor circuit is shown in *Figure (3.3)*.

⇒ When switch $A$ is open i.e. logic '0' then, the bulb glows (shows logic '1').

⇒ When switch is closed i.e. logic '1' then, the bulb does not glow (shows logic '0').

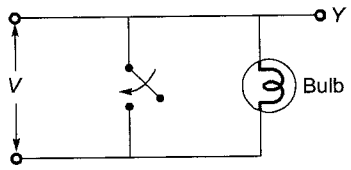



***Figure-3.2:*** *Switching circuit of NOT gate*

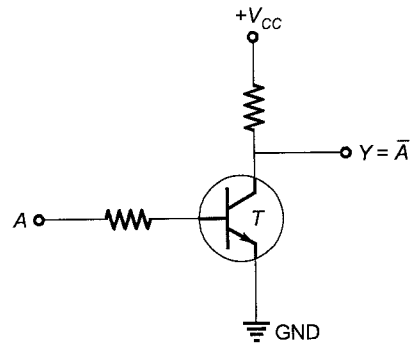



***Figure-3.3*** *Transistor as an inverter*

Similarly for transistor circuit

when, $A = 0$ ; $T$ = OFF and $Y = +V_{CC}$

$A = 1$ ; $T$ = ON and $Y$ = GND

- When even number of NOT gates/inverters are connected in feedback, it acts like a bistable multivibrator (basic memory element) shown in *Figure 3.4 (a)*.

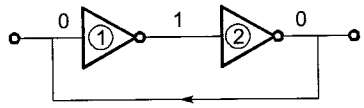



***Figure-3.4 (a)*** *Basic memory element*

- Even number of NOT gates/inverters without feedback act like a butter.

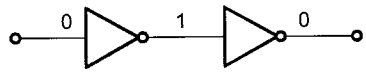



***Figure-3.4 (b)*** *Inverter as a butter*

**NOTE:** Buffers are used to increase the driving capacity of a gate.

- Odd number of NOT gates/inverters connected in feedback act like an astable multivibrator or, a square wave generator, or a clock pulse generator or, a free running oscillator.

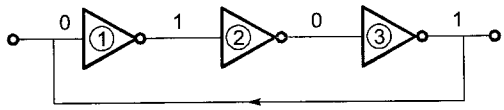



***Figure-3.4 (c)*** *Inverter as an astable MV*

If time taken by each gate to respond to the input is $t_{pd}$ (propagation delay).

Then, for astable multivibrator, $\boxed{T = 2 \times Nt_{pd}}$

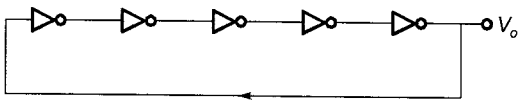
where, $t_{pd}$ = Propagation delay time of inverter

$T$ = Time period of a square wave generator

$N$ = Number of inverters

and $$\boxed{\text{Frequency of oscillation} = \frac{1}{2Nt_{pd}}}$$

**Example-3.1** For the given ring oscillator, the propagation delay of each inverter is 100 pico sec. What is the fundamental frequency of the oscillator output?



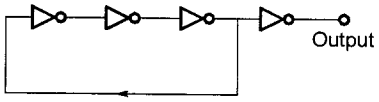

***Solution:***

Here, $N = 5$

$t_{pd} = 100 \times 10^{-12}$ sec.

$$\therefore \quad f = \frac{1}{2Nt_{pd}} = \frac{1}{2 \times 5 \times 100 \times 10^{-2}} = 1 \text{ GHz}$$

**Example-3.2** In the circuit shown below, the propagation delay of each NOT gate is 2 nsec then, the time period of generated square wave is




(a) 6 nsec (b) 16 nsec

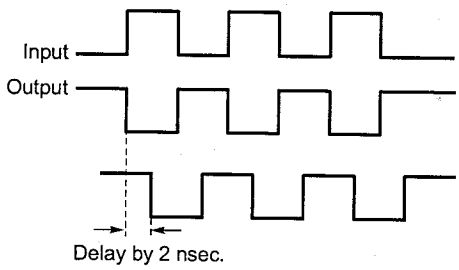
(c) 12 nsec (d) 10 nsec

***Solution: (c)***

$N = 3$ and $t_{pd} = 2$ nsec

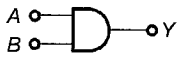
$T = 2N\,t_{pd} = 2 \times 3 \times 2 \times 10^{-9} = 12$ nsec

* Here, the last inverter is only used to invert and delay the generated square wave by the astable multivibrator.




### 3.1.2 The AND Gate

- The AND gate can have two or more inputs but only one output.
- The logic symbol and the truth table of a two input AND gate is shown in *Figure 3.5 (a) and (b)* respectively.



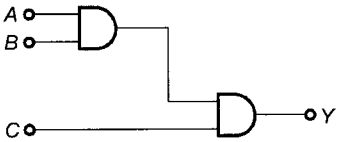

(a)

| Input | | Output |
|---|---|---|
| A | B | Y |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

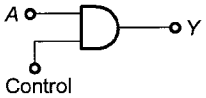
(b)

***Figure-3.5*** *The AND Gate (a) Logic symbol and (b) Truth table*

- The logical expression is $\boxed{Y = AB}$
- It is clear from the truth table that, if all the inputs or any of the input is LOW (logic '0') the output is also at logic '0'. However the output is 1 only when all the inputs are 1.
- AND gate follows both commutative and associative law as:




  (*i*) Commutative law: $AB = BA$
  (*ii*) Associative law: $ABC = (AB)C = A(BC)$
- **Enable and disable inputs:**
  For AND operations




  If control = 0;

| A | Control | Y |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |

  No change in the output, hence **logic '0'** is considered as **disable** input for AND gate.
  If control = 1;

| A | Control | Y |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |

  Due to change in the input output also changes therefore **logic '1'** is **enable** input for AND gate.
- The switching circuit diagram of AND gate is shown in *Figure (3.6)*.

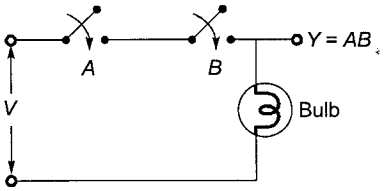


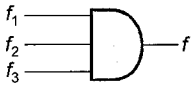

***Figure-3.6*** *Switching circuit*

  The bulb will glow only when both the switches *A* and *B* are closed or at logic '1'.
- The AND operation is performed exactly like ordinary multiplication of 1's and 0's.
- In multi input AND gate, the unused input can be connected to
  (*i*) Logic '1' or pull up (enable)
  (*ii*) One of the used input
  (*iii*) Left open for TTL logic circuit
  Out of these three procedure the best way is to connect the logic '1' or pull up.

**Example-3.3** Consider the logical functions given below.

$$f_1(A, B, C) = \Sigma(2, 3, 4)$$

$$f_2(A, B, C) = \pi(0, 1, 3, 6, 7)$$




If $f$ is logic zero, then maximum number of possible minterms in function $f_3$ are

***Solution:***

$$f_1(A, B, C) = \Sigma(2, 3, 4)$$

$$f_2(A, B, C) = \pi(0, 1, 3, 6, 7) = \Sigma(2, 4, 5) \; ; \; f_1 \cdot f_2 = \Sigma m(2, 4)$$

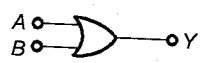
For function $f$ to be zero

$$f_3(A, B, C) = \overline{[f_1(A, B, C) \cap f_2(A, B, C)]} = \Sigma(0, 1, 3, 5, 6, 7)$$

Maximum minterms possible are 6.

### 3.1.3 The OR Gate

- The OR gate can have two or more inputs but only one output. The logic symbol and the truth table for OR gate is shown in *Figure 3.7 (a)* and *(b)* respectively.



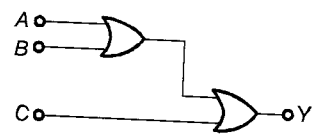

(a)

| Input | | Output |
|---|---|---|
| **A** | **B** | **Y** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

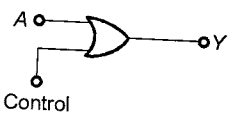
(b)

***Figure-3.7*** *The OR Gate (a) Logic symbol and (b) Truth table*

- Thus, the logical expression is $\boxed{Y = A + B}$
- It is clear from the truth table that, if all the inputs or any of the input is high the output $Y$ is HIGH. Where as if all the inputs are LOW then the output $Y$ is low.
- OR gate follows both commutative and associative laws as:
  - (*i*) Commutative law: $A + B = B + A$
  - (*ii*) Associative law: $(A + B + C) = (A + B) + C = A + (B + C)$




- **Enable and disable inputs:**
  For an OR gate




For control = 0;

| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

The change in the input causes the change in the output hence, for OR gate logic '0' is an enable input.

For control = 1;

| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 1 |

Due to change in the input, output remains the same therefore for OR gate logic '1' is a disable input.

- The switching circuit diagram for OR gate is shown in *Figure (3.8)*.

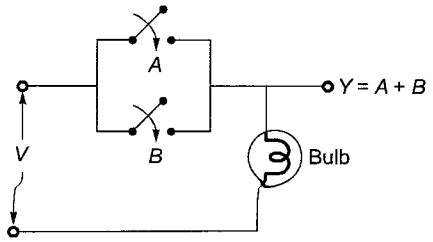
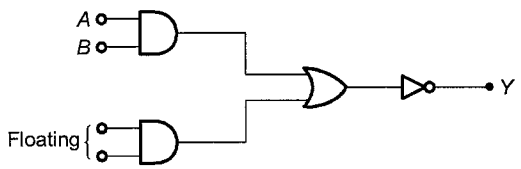



***Figure-3.8*** *Switching circuit*

Here, the bulb will glow when any of the switch (either *A* or *B*), or both the switches are closed.

- In a multi input OR gate an unused input can be connected to
  (*i*) Logic '0' (Enable) or pull-down. (*ii*) Any of the used input.
  (*iii*) Left open or floating in case of ECL logic.
  Out of these procedures, the best way is to connected to the logic '0' or pull down.

**Example-3.4** The circuit shown in the below figure is a TTL circuit. The output of the circuit is




(a) 0 (b) 1
(c) $AB$ (d) $\overline{AB}$

***Solution: (a)***

In case of TTL logic, the floating input is considered as logic '1'.
Therefore, Input to the OR gate = $AB$ and 1 = Input to the NOT gate = 1 and $Y = 0$
Note that in the above question if ECL logic is given instead of TTL logic then the floating inputs are considered as '0'.
Therefore, input to the OR gate = $AB + 0$
Output of the OR gate = 0
Hence, output of the inverter $Y = 1$.

**Example-3.5** A car alarm system is to be designed considering 4 inputs, door closed (*D*), key in (*K*), seat pressure (*S*) and seat belt closed (*B*).

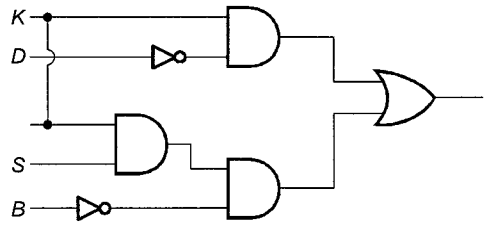
The alarm (*A*) should sound if
1. the key is in and door is not closed or
2. the door is closed, the key is in, driver in the seat and seat belt is not closed.

The system is to be designed with 2 input basic gates and inputs are available in basic form only. The number of gates required are

(a) 6 (b) 7
(c) 8 (d) 9

***Solution : (a)***

$$A = K\bar{D} + KDS\bar{B} = K\bar{D} + KS\bar{B}$$

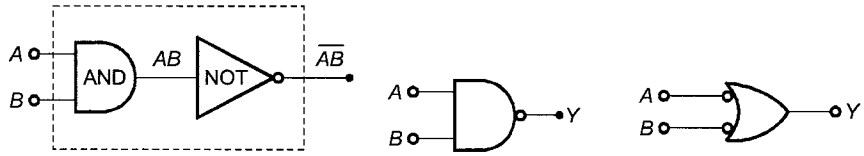



## 3.2 Universal Gates

The NAND gate and NOR gate are called Universal gates because they can perform all the three basic functions of AND, OR and NOT gates.

### 3.2.1 The NAND Gate

- The NAND gate is a AND gate followed by NOT gate. Thus, we can say, it is a NOT-AND operation. It may have two or more inputs but only one output. The logical symbols of a NAND gate are shown in *Figure 3.9 (a), (b)* and *(c)* and the truth table is shown in *Figure 3.9 (d).*




| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(a) (b) (c) (d)

***Figure-3.9*** *The NAND Gate (a), (b), (c) and (d)*

- Thus, the logical expression for the output is $\boxed{Y = \overline{A \cdot B} = \bar{A} + \bar{B}}$.

  It is clear from the truth table of two input NAND gate, that the output is 1 when either *A* or *B* or when both the inputs are at logic '0'. We can say that, if $\bar{A} = 1 = \bar{B}$ are both $\bar{A}$ and $\bar{B}$ are '1', the output is 1. Therefore, the NAND gate can perform the OR function by inverting the inputs and then ORing it *(Figure 3.9 (c)).*
- In NAND gate output is exact inverse of the AND gate for all possible input combinations.
- The OR gate with inverted inputs is called bubbled OR gate or negative OR gate. The NAND gate is also called active low OR gate.
- The switching circuit diagram for a NAND gate is shown in *Figure (3.10).*

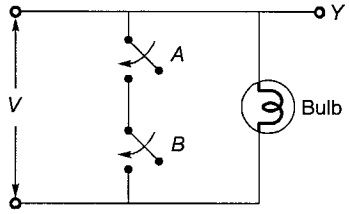


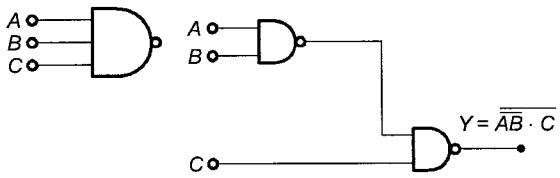

***Figure-3.10*** *Swithcing Circuit*

The bulb will glow when any of the switches $A$ or $B$ will open (logic 0).

- NAND gate follows the commutative law as

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

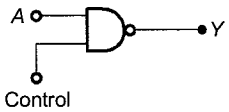
- NAND gate will not follow the associative law as




$$Y = \overline{\overline{AB} \cdot C} = \overline{\overline{AB}} + \overline{C} = A + B + \overline{C}$$

- **Enable and disable inputs:**

For a NAND Gate




For control = 0;

| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 1 |

Thus, **logic '0'** is the **disable** input for NAND gate

For control = 1;

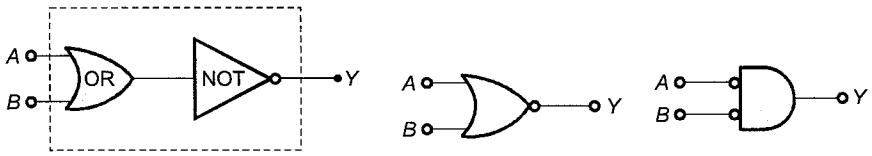
| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 0 |

Thus, **logic '1'** is the **enable** input for NAND gate.

**NOTE:** Unused input in any multi input NAND gate is always connected to the enable logic or Pull-up.

### 3.2.2 The NOR Gate

- A NOR gate is a combination of OR gate and NOT gate. So we can say it is a NOT-OR operation. It may have two or more input and an output. The logical symbols of NOR gate is shown in *Figure 3.11 (a), (b) and (c)*, whereas, the truth table is listed in *Figure 3.11 (d)*.




(a) (b) (c)

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

(d)

***Figure-3.11*** *The NOR Gate (a), (b) (c) and (d)*

- It is clear from the truth table that the output is '1' only if all the inputs are at logic '0'. It can also say that if the inputs $\overline{A} = \overline{B} = 1$, then the output $Y$ is 1. Thus, the NOR gate is equivalent to the AND gate with inverted inputs and it can be realized by a bubbled AND gate as shown in *Figure 3.11 (c)*.
- The logical expression for the output is

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

- The NOR gate is also called active LOW AND gate.
- *Figure (3.12)* shows the switching circuit of a NOR gate.

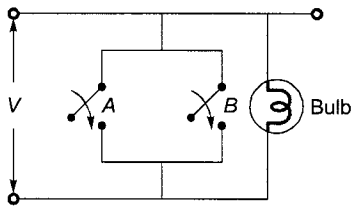


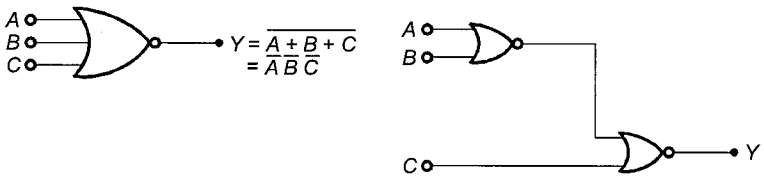

***Figure-3.12*** *Switching circuit*

When any of the switches either $A$ or $B$ is closed, the bulb will not glow.

- NOR gate follows the commutative law as

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

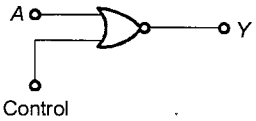
However, it does not follow the associative law.




Here, $$Y = \overline{(\overline{A+B}+C)} = \overline{\overline{A+B}}\cdot\overline{C} = (A+B)\cdot\overline{C}$$

- **Enable and disable inputs:**

For a NOR Gate




If control = 0;

| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |

Thus, **logic '0'** is **enable** input

If control = 1;

| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 0 |

Thus, **logic '1'** is **disable** input.

**NOTE:** Unused input in any multi input NOR gate is always connected to the enable logic or Pull-Down.

## 3.3 Special Purpose Gate

### 3.3.1 The Exclusive-OR Gate (EX-OR)

- A basic EX-OR gate is a two input single output logic gate, whose output is assumed to be HIGH only when one and only one of its inputs is HIGH. The logic symbol and the truth table for a two input EX-OR gate is shown in *Figure 3.13 (a)* and *(b)* respectively.

(a)

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(b)

***Figure-3.13:*** *The EX-OR Gate (a) Logic symbol and (b) Truth table*

- If input variables are represented by $A$ and $B$ then the logical expression for output is

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

- Practically three or more input EX-OR gate does not exist. But when more than two variables are EX-ORed, a number of two input EX-OR gate is used. Where, the output is assumed to be '1' when odd number of input variables are '1'.
  That is why, EX-OR gate is known as odd number of 1's detector in the input.
- The most important application of EX-OR gate is in "parity generation and detection".
- It is also known as "stair case switch".
- The switching circuit of EX-OR gate is shown in *Figure (3.14)*.

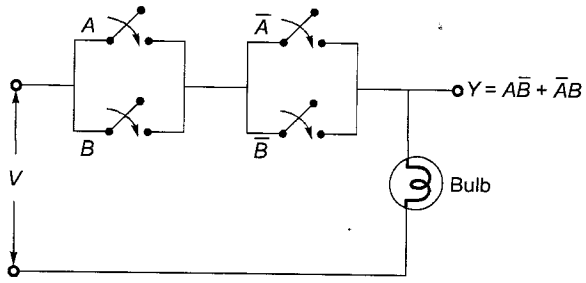
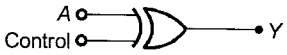



***Figure-3.14*** *Switching circuit*

$$Y = (A+B)(\bar{A}+\bar{B}) = A\bar{B} + \bar{A}B \text{ or } \boxed{A \oplus B}$$

- The EX-OR gate follows both commutative and associative law.
- **Enable and disable inputs:**

For an EX-OR Gate




For control = 0;

| A | Control | Y |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Thus, the EX-OR gate acts as a **buffer** for controlled input of **logic '0'**.
For control = 1;

| $A$ | Control | $Y$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 0 |

Thus, the EX-OR gate acts as an **inverter** for control input of **logic '1'**.

- Properties of EX-OR gate:
  (*i*) $A \oplus A = 0$ (*ii*) $A \oplus 0 = A$
  (*iii*) $A \oplus \bar{A} = 1$ (*iv*) $A \oplus 1 = \bar{A}$
  (*v*) $A \oplus A \oplus A = A$
- Internal diagram of EX-OR gate.

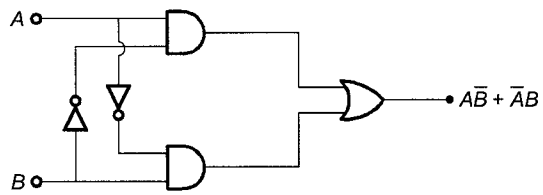



***Figure-3.15***

**Example-3.6** $Y = B \oplus B \oplus B \cdots$ the variable $B$ is appearing in the equation $n$ times, then the value of $Y$ is

(a) Zero when $n$ = even
(b) $B$, when $n$ = even
(c) Zero when $n$ = odd
(d) $B$, when $n$ = odd

***Solution: (a) & (d)***

For even number $Y = 0$ as $B \oplus B = 0$
For odd number $Y = B$ as $B \oplus B \oplus B = B$

**Example-3.7** Show that the dual of the exclusive OR is equal to its complement.

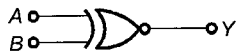
***Solution:***

Let $A$ and $B$ are the two inputs

Then,
$$A \oplus B = A\bar{B} + \bar{A}B$$
$$\text{Dual of } (A \oplus B) = (A+\bar{B})\cdot(\bar{A}+B) = AB + \bar{A}\bar{B}$$
$$\text{Complement of } (A \oplus B) = (A\bar{B} + \bar{A}B)' = (\bar{A}+B)(A+\bar{B})$$
$$= AB + \bar{A}\bar{B} = \text{Dual of } (A \oplus B)$$

### 3.3.2 The Exclusive NOR (EX-NOR) Gate

- An EX-NOR gate is an EX-OR gate followed by NOT gate. The EX-NOR gate is a two input and one output logic circuit in which the output is '1' only when both the inputs are same. Thus, it is also called **"gate of equivalence"** or **"coincidence logic"**.
- The logic symbol and the truth table of a two input EX-NOR gate are shown in *Figure 3.16 (a)* and *(b)* respectively.




(a)

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(b)

***Figure-3.16*** *The EX-NOR Gate (a) Logical symbol and (b) Truth table*

Thus, for input variables $A$ and $B$ the logical expression for the output of an EX-NOR gate is given by

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

- Practically, three or more variable EX-NOR gate does not exist. However if number of variables have to be EX-NORed then a number of two variable EX-NOR gates will be used in which the output is '1' for even number of input variables are 1. Thus it is also known as "even number of 1's detector". When the number of input variables are odd then it is also known as "odd number of 1's detector".
- Switching circuit for EX-NOR gate is shown in *Figure (3.17)*.

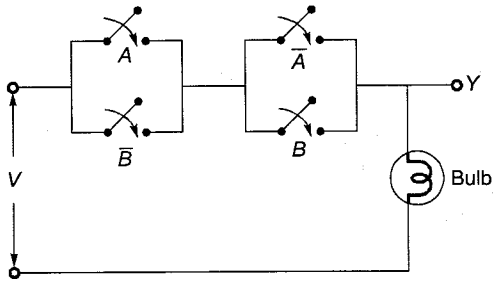
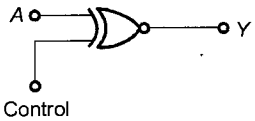



***Figure-3.17*** *Swithcing circuit*

- EX-NOR gate follows both commutative and associative law.
- **Enable and disable inputs:**

  For an EX-NOR Gate




For control = 0;

| A | Control | Y |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |

Thus, it acts as an inverter for logic '0'.

For control = 1;

| A | Control | Y |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |

Thus, it acts like a **buffer** for **logic '1'**.

- Properties of EX-NOR gate:

(*i*) $A \odot A = 1$ (*ii*) $A \odot 1 = A$

(*iii*) $A \odot 0 = \overline{A}$ (*iv*) $A \odot \overline{A} = 0$

(*vi*) $A \odot A \odot A \odot A = 1$

Thus for odd number of same input the output is same as input and for the even number of same input the output is equals to logic '1'.

**Note:** $(A \odot B) \odot C = A \odot (B \odot C)$

**Example-3.8** Find (*i*) $\overline{A} \oplus B = ?$ (*ii*) $A \oplus B \oplus AB = ?$

***Solution:***

(*i*)
$$\overline{A} \oplus B = \overline{A}\,\overline{B} + \overline{\overline{A}}\,B$$
$$\overline{A}\,\overline{B} + AB = A \odot B$$

(*ii*)
$$A \oplus B \oplus AB = (\overline{A}B + A\overline{B}) \oplus AB = (\overline{A}B + A\overline{B})'AB + (\overline{A}B + A\overline{B})\,\overline{AB}$$
$$= (A + \overline{B})(\overline{A} + B)\,AB + (\overline{A}B + A\overline{B})(\overline{A} + \overline{B})$$
$$= AB + \overline{A}B + \overline{B}A = A + B$$

**Example-3.9** 7E H and 5F H are EX-ORed. The output is multiplied by 10H. The result is

(a) 0210 H (b) 7E5F H

(c) 5F7E H (d) 2100 H

***Solution: (a)***

7EH → 0111 1110

5FH → 0101 1111

Ex-ORed 0010 0001 = 21H

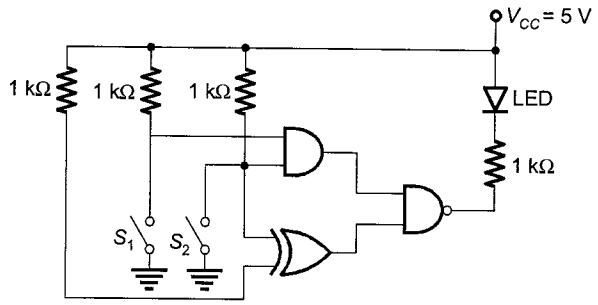
21 H

× 10 H

Result → 0210 H

**NOTE:** For two inputs EX-OR gate if one of the input is in complemented form, then it gives EX-NORed output and vice-versa.

**Example-3.10** In the figure, the LED



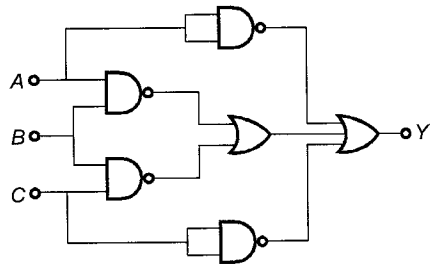
(a) emits light when both $S_1$ and $S_2$ are closed.
(b) emits light when both $S_1$ and $S_2$ are open.
(c) emits light when only of $S_1$ and $S_2$ is closed.
(d) does not emit light, irrespective of the switch positions.

***Solution: (d)***

For LED to be ON, output of NAND gate = 0

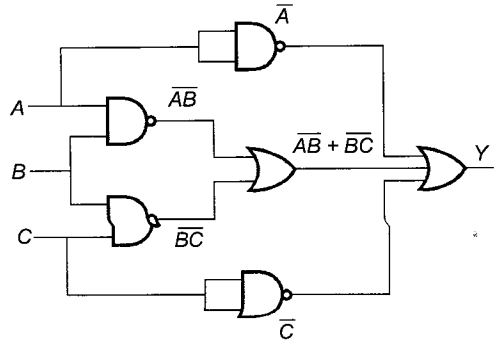
No condition of $S_1$ and $S_2$ gives output of NAND gate zero. So LED will never glow.

**Example-3.11** For the logic circuit shown in Figure, the output is equal to



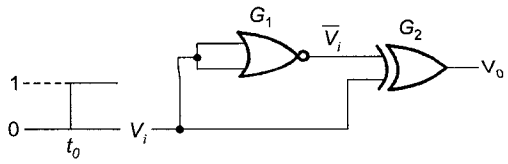

(a) $\overline{ABC}$
(b) $\bar{A}+\bar{B}+\bar{C}$
(c) $\overline{AB}+\overline{BC}+\bar{A}+\bar{C}$
(d) $\overline{AB}+\overline{BC}$

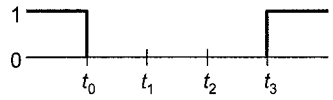
***Solution: (b)***

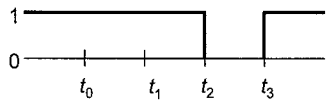


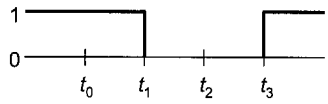

$$Y = \bar{A}+\overline{AB}+\overline{BC}+\bar{C} = \bar{A}+(\bar{A}+\bar{B})+(\bar{B}+\bar{C})+\bar{C} = \bar{A}+\bar{B}+\bar{C}$$

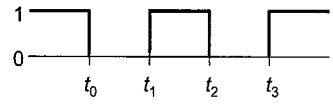
**Example-3.12** The gates $G_1$ and $G_2$ in the figure have propagation delays of 10 nsec and 20 nsec respectively. If the input $V_i$ makes an abrupt change from logic 0 to 1 at time $t = t_0$, then the output waveform $V_0$ is



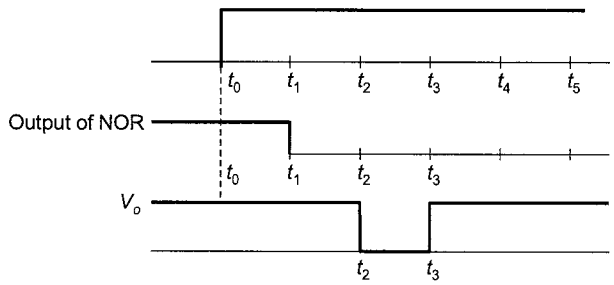

(a)



(b)


(c)



(d)



***Solution: (b)***




**Example-3.13** A bulb in a staircase has two switches, one switch being at the ground floor and the other one at the first floor. The bulb can be turned ON and also can be turned OFF by any one of the switches irrespective of the state of the other switch. The logic of switching of the bulb resembles

(a) an AND gate
(b) an OR gate
(c) an XOR gate
(d) a NAND gate

***Solution: (c)***

Truth table of XOR gate

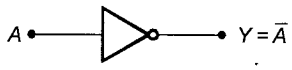
So, from the XOR gate truth table it is clear that the bulb can be turned ON and also can be turned OFF by any one of the switches irrespective of the state of the other switch.

| $A$ | $B$ | $Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

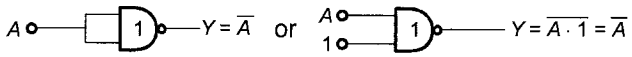
## 3.4 Realization of Logic Gates Using Universal Gates

### 3.4.1 Realization of basic gates using NAND and NOR Gate

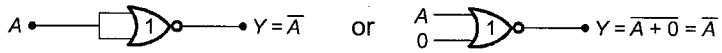
**(i) NOT Gate Realization**



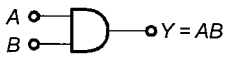

Using NAND gate:



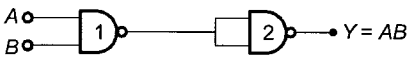

Using NOR gate:



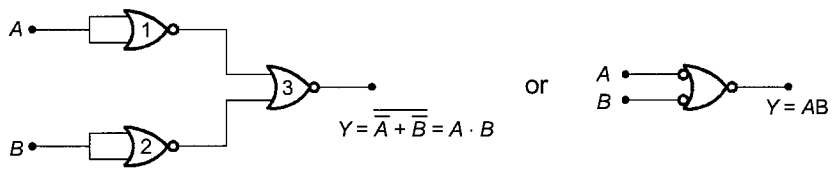

**(ii) AND Gate Realization**



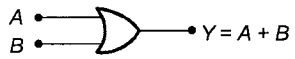

Using NAND Gate:



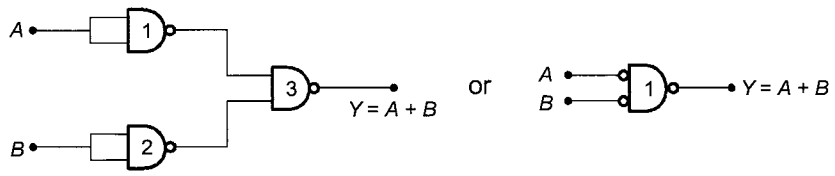
Using NOR Gate:



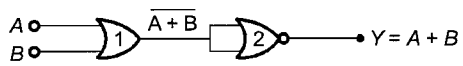

**(iii) OR Gate Realization**




Using NAND Gate:

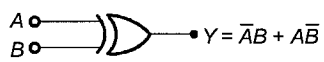



Using NOR gate:

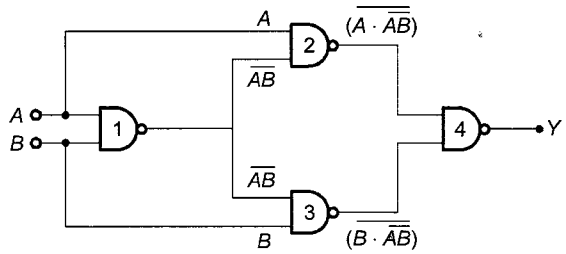



## 3.4.2 Realization of special purpose gate using NAND and NOR gate

**(i) EX-OR Gate Realization**



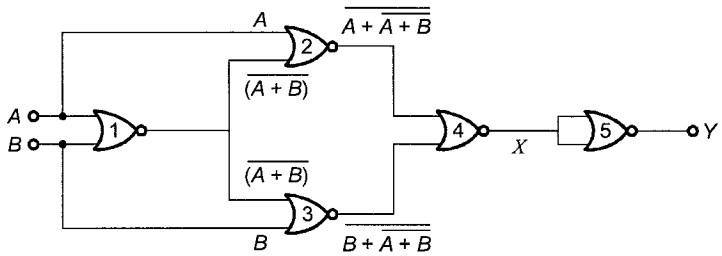

Using NAND Gate:




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

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

Using NOR Gate:

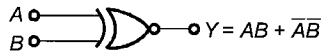


Here, $X = \overline{\overline{A+\overline{A+B}}+\overline{B+\overline{A+B}}} = \left[A+(\overline{A+B})\right]\cdot\left[B+(\overline{A+B})\right]$

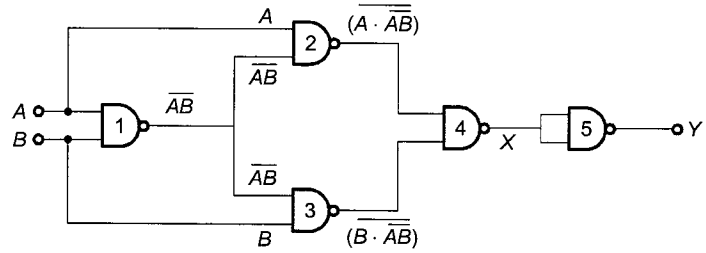
$= \left[A+(\overline{A}\,\overline{B})\right]\left[B+\overline{A}\,\overline{B}\right] = AB+\overline{A}\,\overline{B}$

Now, $Y = \overline{X} = (\overline{AB+\overline{A}\,\overline{B}}) = A\overline{B}+\overline{A}B$

**(ii) EX-NOR Gate Realization**




Using NAND Gate:

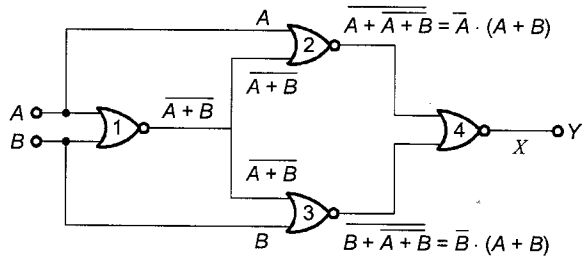



Here, $X = \overline{\overline{A\cdot\overline{AB}}\cdot\overline{B\cdot\overline{AB}}} = (A\cdot\overline{AB}+B\cdot\overline{AB})$

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

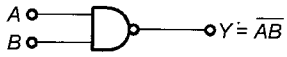
Now, $Y = \overline{X} = \overline{A}\,\overline{B}+AB = A\odot B$

Using NOR Gate:

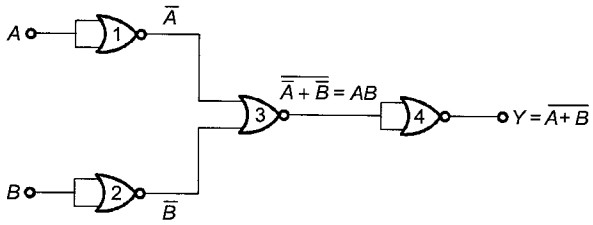



$Y = \overline{\overline{A}B+A\overline{B}} = \overline{A}\cdot\overline{B}+AB = A\odot B$

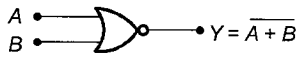
### 3.4.3 Realization of NAND gate using NOR gate



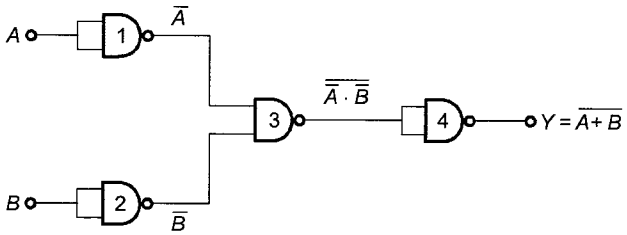

NAND Gate




### 3.4.4 Realization of NOR Gate using NAND Gate




NOR Gate

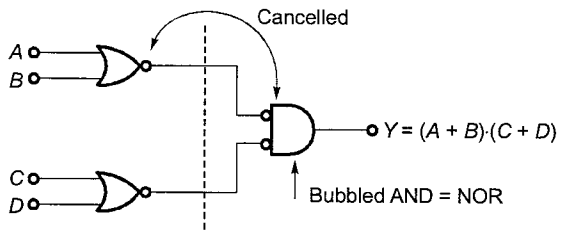



### 3.4.5 Shortcut for Realization of Logic Gates using NAND and NOR

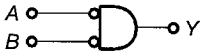
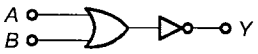
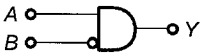
| | NAND $\xleftrightarrow{4}$ | NOR |
|---|---|---|
| NOT | 1 | 1 |
| AND | 2 | 3 |
| OR | 3 | 2 |
| EX-OR | 4 | 5 |
| EX-NOR | 5 | 4 |

**Example-3.14** How many NOR gates are required to implement the Boolean function $(A + B)\cdot(C + D)$?

***Solution:*** **(3)**




**Example-3.15** Consider the following digital logic circuits.

1. 
2. 
3.
4. 

Which of these above digital circuit acts as an Inhibitor?

(a) Only 3
(b) Only 4
(c) 1 and 4
(d) Both 3 and 4

***Solution: (b)***

If one of the inputs to AND-gate is inverted then it becomes an "Inhibitor".

**Example-3.16** Draw a logic diagram using only two-input NOR gate to implement $F(A, B, C, D) = (A \odot B)'\,(C \oplus D)$

***Solution:***

$$F(A, B, C, D) = (A \odot B)'\,(C \oplus D)$$

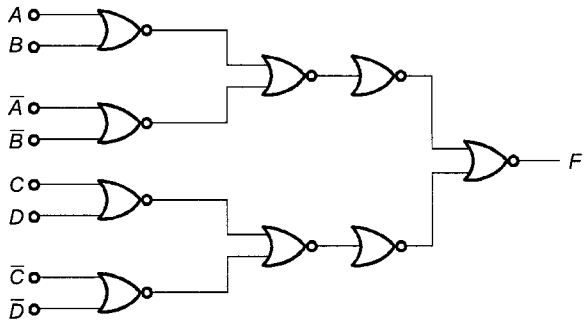
$$= (\bar{A}\bar{B} + AB)'\,(C\bar{D} + \bar{C}D) = (\bar{A}B + A\bar{B})\,(C\bar{D} + \bar{C}D)$$

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

or,

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

or, $$F = \left\{\left[(A+B)' + (\bar{A}+\bar{B})'\right] + \left[(C+D)' + (\bar{C}+\bar{D})'\right]\right\}'$$




## Summary

- Universal gates do not follow associative law.
- EX-OR gate is also called, controlled inverter (control = 0).
- EX-NOR gate is also called, controlled inverter (control = 1).
- For SOP implementation AND-OR logic is used.
- For POS implementation OR-AND logic is used.
- The AND-OR logics equivalent to NAND-NAND logic.
- The OR-AND logic is equivalent to NOR-NOR logic.
- NAND gate is preferrable than NOR gate, because it consumes less power.
- In EX-OR gate if the number of variables are '$n$', then the total number of minterms in Boolean expression is $n^2/2$.
- AND gate is known as detector logic.
- Degenerated form: two level logic circuit is represented by a single gate:
  - (*i*) AND-AND → AND
  - (*ii*) OR-OR → OR
  - (*iii*) AND-NAND → NAND
  - (*iv*) OR-NOR → NOR
  - (*v*) NOR-NAND → OR
  - (*vi*) NAND-NOR → AND

## Student's Assignments 1

**Q.1** Design a circuit using NAND gates only that has one control line and three data lines. When the control line is HIGH, the circuit is to detect when one of the data lines has a '1' on it. No more than one data line will ever have a '1' on it. When the control line is low, the circuit will output a '0', regardless of what is on the data lines.

**Q.2** (*i*) Simplify the following logic expression $F = A\bar{B} + AB\bar{C} + ABCD + ABC\bar{D}$ and realize it by using NAND gates only.

(*ii*) For implementing the function, $F = (\bar{X}+\bar{Y})(Z+W)$. How many minimum number of two input NAND gates required?

**Q.3** Realise the logical expressions:

(*i*) $Y = \overline{AB} + A + (\overline{B+C})$ using NAND-gates only.

(ii) $Y = (A+C)(A+\overline{D})(A+B+\overline{C})$ using NOR-gates only.

***Answer: (Conventional)***

**2.** (i) $A$ (ii) 4

## Student's Assignments 2

**Q.1** Which one of the following statements is correct? For a 4-input NOR gate, when only two inputs are to be used, the best option for the unused input is to

(a) Connect them to the ground
(b) Connect them to $V_{CC}$
(c) Keep them open
(d) Connect them to the used inputs

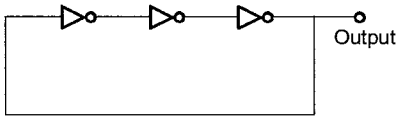
**Q.2** Assuming that only the $X$ and $Y$ logic inputs are available and their complements $\overline{X}$ and $\overline{Y}$ are not available, what is the minimum number of two input NAND gates required to implement $X \oplus Y$?

(a) 2 (b) 3
(c) 4 (d) 5

**Q.3** A gate is disabled when its disable input is at logic 0. The gate is

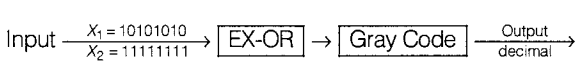
(a) OR (b) AND
(c) NOR (d) EX-OR

**Q.4** The circuit shown in the figure is




(a) an oscillating circuit and its output is a square wave
(b) one whose output remains stable in '1' state
(c) one whose output remains stable in '0' state
(d) having a single pulse of 3 times of propagation delay

**Q.5** For the figure shown below, the input is considered as signed number. The decimal equivalent of output is




(a) +255 (b) +127
(c) +143 (d) +31

**Q.6** Consider a 4-input NAND gate, how many number of input conditions are possible, that will produce output "HIGH".

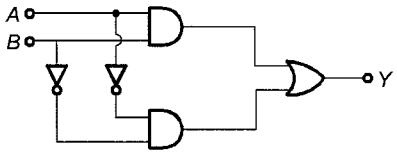
(a) 0 (b) 1
(c) 2 (d) 15

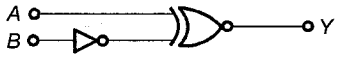
**Q.7** If $A \neq B$, then the value of Boolean expression,

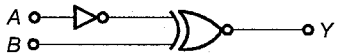
$\overline{\overline{(\overline{A}+B)} + \overline{(A+\overline{B})}} + \overline{(\overline{A}B)}\,\overline{(A\overline{B})}$ equals to,

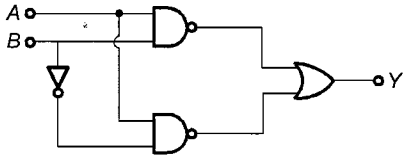
(a) 0 (b) 1
(c) $A + B$ (d) $AB$

**Q.8** Which one of the following digital circuits does not represents the 'Stair-case switch"?

(a)



(b)



(c)



(d)



**Q.9** After the simplification of a 3-variable Boolean function,

$f(A, B, C) = (A \oplus B \oplus AB)\,(A \oplus C \oplus AC)$

we requires,

(a) 1-AND and 1-OR gate
(b) 1-AND and 2-OR gate
(c) 2-AND and 1-OR gate
(d) Only 1-AND gate

***Answer Key:***

| | | | | |
|---|---|---|---|---|
| **1.** (a) | **2.** (d) | **3.** (b) | **4.** (a) | **5.** (c) |
| **6.** (d) | **7.** (a) | **8.** (d) | **9.** (a) | |

CHAPTER 04

# Combinational Logic Circuits

## Introduction

A combinational circuit consists of an interconnection of logic gates, whose outputs at any instant of time are determined from present combination of inputs only. A block diagram of a combinational circuit is shown in Figure 4.1. The '$n$' input binary variables come from external sources; the '$m$' output variables are produced by the internal combinational logic circuit and go to an external destination. Each input and output variables exist physically as an analog signal whose values are interpreted to be a binary signal that represents logic '1' and logic '0'.

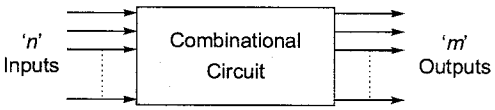



***Figure-4.1*** Basic diagram of combinational circuit

For '$n$' input variables, there are $2^n$ possible combinations of binary inputs. For each possible input combination, there is only one possible output combination. Therefore, a combinational circuit can be described by '$m$' boolean functions. One for each output variable. Each output function is expressed in terms of $n$ input variable. A combinational circuit can also be specified by a truth table that lists the output for each combination of input variables.

## 4.1 Design Procedure for Combinational Circuit

The design of combinational circuits starts from the specification of design objective and ends in a set of Boolean functions from which a logic circuit diagram can be easily obtained. The procedure involves the following steps:

(*i*) From the specification of given circuit; determine the number of input and output variables.
(*ii*) Assign symbols to each variable.
(*iii*) Derive the truth table which defines the required relationship between input and output variables.
(*iv*) Obtain the simplified Boolean function for each output in Sum of Product (SOP) or Product of Sum (POS).
(*v*) Implement the logic circuit.

**Example-4.1** Consider the following:

Any combinational circuit can be built using

1. NAND gates
2. NOR gates
3. Ex-OR gates
4. Multiplexers

Which of these are correct?

(a) 1, 2 and 3
(b) 1, 3 and 4
(c) 2, 3 and 4
(d) 1, 2 and 4

***Solution: (d)***

NAND & NOR are universal gates, so along with multiplexers they can be used to implement any combinational circuit.

## 4.2 Arithmetic Circuits

### 4.2.1 Adders

**The Half-Adder**

- The half adder is an arithmetic circuit used to perform the addition of two single bits.
- The two input variables to the half adder designate the augend and addend bits and the output variables produce the sum and carry. The block diagram of a half adder is shown in Figure 4.2. The truth table for half adder is listed in Table 4.1.

  Let A and B are the two input variables and sum (S) and carry (C) are the two output variables then,

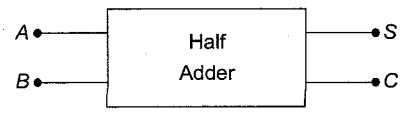
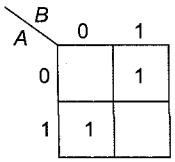
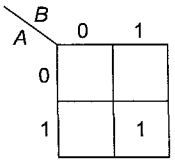



***Figure-4.2*** *Block diagram of Half-adder*

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




$S = A \oplus B$




$C = AB$

***Table-4.1*** *Truth table of a Half-adder*

The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The logical expressions for sum and CARRY are:

$$\boxed{S = \bar{A}B + A\bar{B} = A \oplus B} \quad \text{and} \quad \boxed{C = AB}$$

- **Implementation of half adder circuit:** From the given expression, a half adder can be realized by using one Ex-OR gate and an AND gate as shown in Figure 4.3 (a).

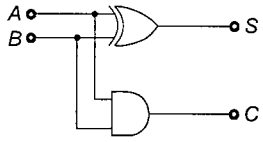



***Figure-4.3 (a)*** *Logic diagram of half-adder*

- A half-adder can also be realized in universal logic by using either only NAND gates or only NOR gates as shown in Figure 4.3 (b) and (c) respectively.

  NAND logic:

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

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

$$C = AB = \overline{\overline{AB}}$$

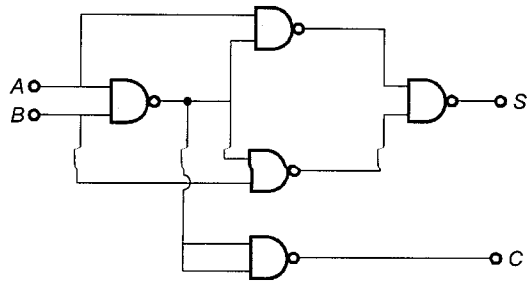



***Figure-4.3 (b)*** *NAND realization of a half-adder*

NOR logic:

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

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

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

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

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

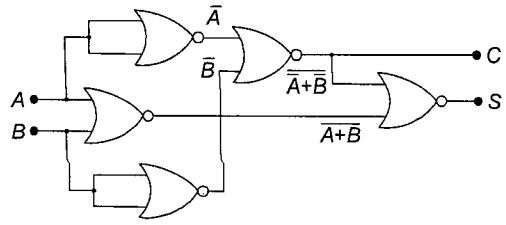



***Figure-4.3 (c)*** *NOR realization of a half-adder*

**Do You Know**

The Half-adder can also be realized using universal logic, i.e., either only NAND gates or only NOR gates. The total number of NAND/NOR gates required to implement a half-adder is equals to "5".

- Implementation of half-adder circuit by minimum number of logic gates, if we have all gates except EX-OR and EX-NOR is "3".

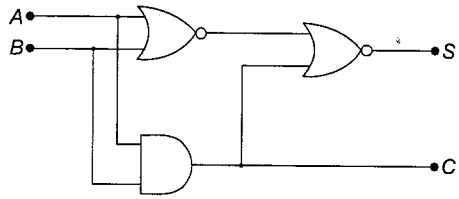



***Figure-4.3 (d)***

**Example-4.2** The half-adder circuit in the given Figure has inputs $AB = 11$

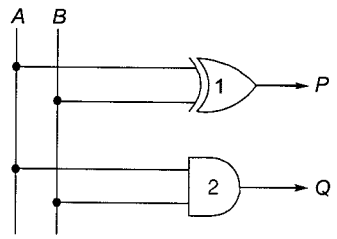
The logic level of $P$ and $Q$ outputs will be

(a) $P = 0$ and $Q = 0$

(b) $P = 0$ and $Q = 1$

(c) $P = 1$ and $Q = 0$

(d) $P = 1$ and $Q = 1$



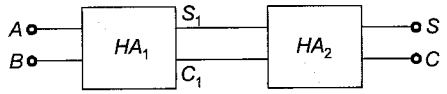

***Solution: (b)***

$$A = 1, B = 1$$

$$P = A \oplus B = A\bar{B} + \bar{A}B = 0$$

$$Q = AB = 1$$

**Example-4.3** Two Half Adders are connected in cascade as shown in figure below. The output "$S$" and "$C$" are




(a) $S = A \oplus B,\ C = AB$
(b) $S = A \odot B,\ C = 0$
(c) $S = A + B,\ C = 0$
(d) $S = AB,\ C = 0$

***Solution: (c)***

$$S_1 = A \oplus B$$
$$C_1 = AB$$
$$S = (A \oplus B) \oplus AB = (A \oplus B) \cdot \overline{AB} + (\overline{A \oplus B}) \cdot AB$$
$$= (A\bar{B} + \bar{A}B)(\bar{A} + \bar{B}) + (AB + \bar{A}\bar{B})(AB) = A\bar{B} + \bar{A}B + AB = A + B$$
$$C = (A \oplus B) \cdot AB = (A\bar{B} + \bar{A}B) \cdot AB = 0$$

**The Full-adder**

- A full-adder is a combinational circuit that performs the arithmetic sum of three input bits. It consists of three input variables designated by augend, addend and the carry bit. The two output variables produce the SUM and CARRY.
- The block diagram of a full adder is shown in Figure 4.4 and the truth Table for full-adder is listed in Table 4.2.

  Let the two input variables are denoted by A and $B$, represent the two significant bits to be added. The third input $C$ represents the carry from the previous lower significant position. The two outputs are sum ($S$) and CARRY ($C_{out}$).

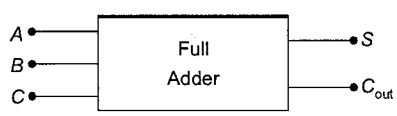
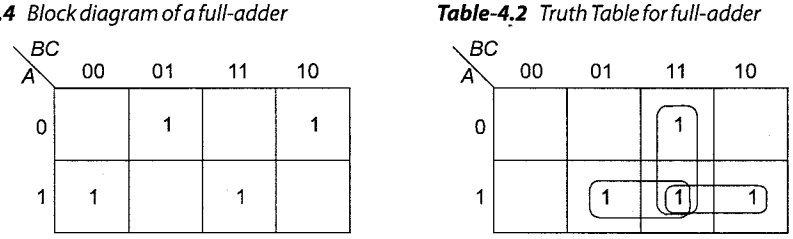



***Figure-4.4*** *Block diagram of a full-adder*

| INPUT | | | OUTPUT | |
|---|---|---|---|---|
| A | B | C | S | $C_{out}$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

***Table-4.2*** *Truth Table for full-adder*




(i) K-map for SUM  (ii) K-map for CARRY

***Figure-4.5*** *K-maps for full-adder*

- From the truth Table and the maps for the outputs of the full-adder as shown in Table 4.2 and Figure 4.5 respectively, the logical expressions for sum and CARRY are:

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

Thererore, $S = A \oplus B \oplus C$ or $S = \Sigma_m(1, 2, 4, 7)$

and $C_{out} = AB + C(A \oplus B)$ or $C_{out} = \Sigma_m(3, 5, 6, 7)$

- **Implementation of full-adder circuit:** From the given expression a full adder circuit can be realized in SOP form or with two half adder and an OR gate as shown in Figure 4.6 (a) and 4.6 (b) respectively.

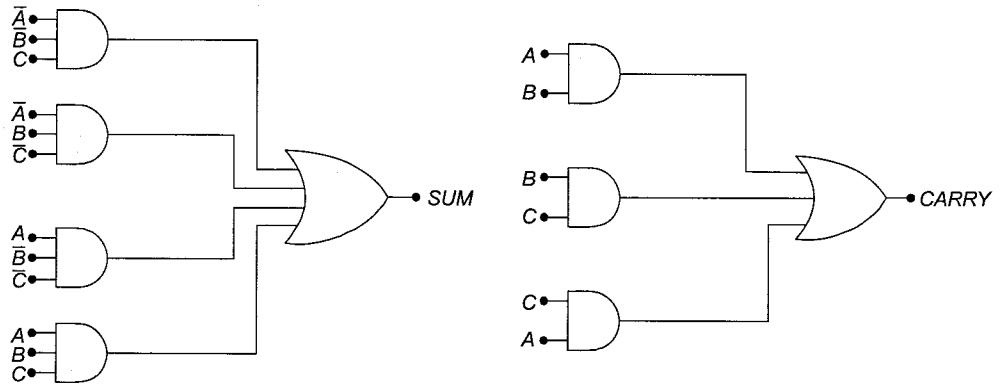



***Figure-4.6 (a)*** *Logic diagram of full-adder in SOP form*

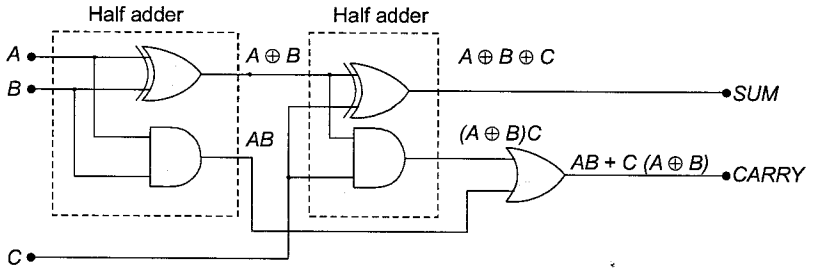



***Figure-4.6 (b)*** *Implementation of full-adder with two half adders and OR gate*

- The ANSI/IEEE logic diagram for full-adder is shown as.

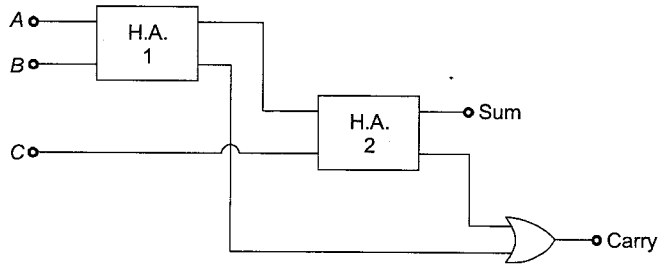



***Figure-4.6 (c)*** *IEEE/ANSI logic diagram for full-adder*

**Do You Know?**

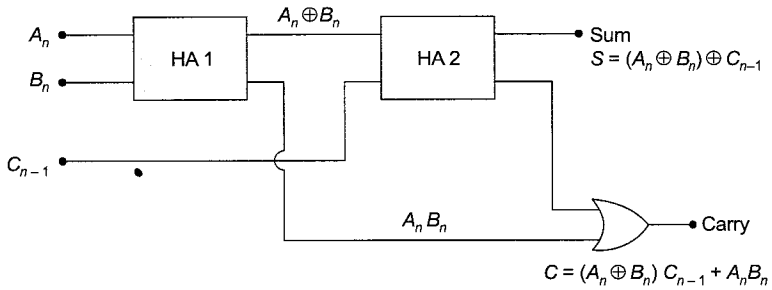
The full-adder can also be realized using universal logic, i.e., either only NAND gates or only NOR gates. The total number of NAND/NOR gates required to implement a full-adder is equals to "9".

**Example 4.4** Which one of the following statements is not correct?

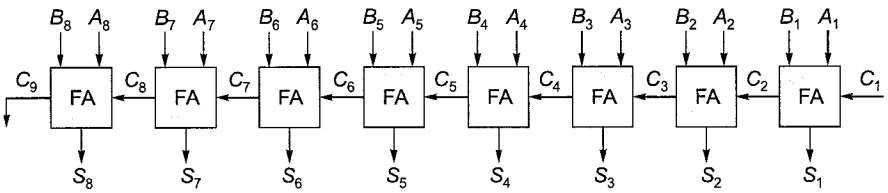
(a) A full adder can be constructed using two half-adders and an OR gate.
(b) Two four bit parallel adders can be cascaded to construct 8-bit parallel adder.
(c) Ripple carry adder has addition time independent of the number of bits.
(d) Carry look ahead is used to speed up the parallel addition.

***Solution: (c)***

(*i*) Full adder using two half adders and an OR gate:




(*ii*) Two 4-bit parallel adders can be cascaded to construct 8-bit parallel adder.




(*iii*) Look-ahead carry is used to reduce the carry delay time and thus to speed up the parallel addition.

## Binary Parallel Adder

- A binary parallel adder is a combinational digital circuit that adds two binary number in parallel form. It consists of full-adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder.
- For addition of $n$-bit numbers a chain of $n$ full adders is required or a chain of one half adder and $(n-1)$ full adders is required. However in former case, the input carry to the least significant position is fixed at '0'. Figure 4.7 shows the interconnection of four full-adder (FA) circuits to provide a four bit binary parallel adder. Let the two four bit inputs designated as A (Augend) and B (Addend) then,

$$\begin{array}{rrrrrr} & & A_3 & A_2 & A_1 & A_0 \\ & + & B_3 & B_2 & B_1 & B_0 \\ \hline carry \Leftarrow & C_4 & S_3 & S_2 & S_1 & S_0 \end{array}$$

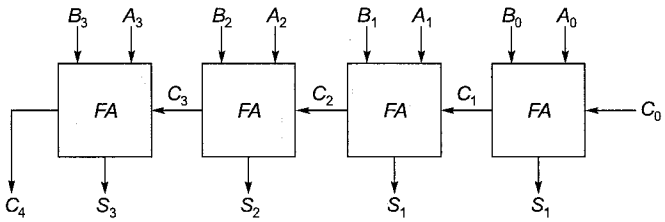



***Figure-4.7*** *Block diagram of four bit parallel adder*

Hence,

'$n$' binary parallel adder requires-

'$n$' full adder.

or $(n-1)$ full adder and a half adder

or $(n-1)$ [2 half adder + 1 OR gate] and a half adder

or $(2n-1)$ half adder and $(n-1)$ OR gate.

To demonstrate with a specific example, consider the two binary numbers

$A = 1\,0\,1\,1$ and $B = 0\,0\,1\,1$

| Subscript ($i$) | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| Input carry ($C_i$) | 0 | 1 | 1 | 0 |
| Augend ($A_i$) | 1 | 0 | 1 | 1 |
| Addend ($B_i$) | 0 | 0 | 1 | 1 |
| Sum ($S_i$) | 1 | 1 | 1 | 0 |
| Output carry ($C_{out}$) | 0 | 0 | 1 | 1 |

Here, the input carry $C_0$ in the least significant position must be 0. The value of $C_{i+1}$ in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left. The sum bits are thus generated starting from the right most position and are available as soon as the corresponding previous carry bit is generated.

**Carry Propagation**

The addition of two binary numbers in parallel implies that all the bits of the input variables are available for computation at the same time. As, in any combinational circuit, the signal must propagate through the gates before the output is available in the output terminals. The total propagation time is equal to the total propagation delay of typical gates in the circuit. The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adder. Thus, carry propagation delay for each full adder is the time between the application of the carry in and the occurrence of carry out. In parallel adder, propagation delay is present from input carry to output carry, hence it is also called **'RIPPLE CARRY ADDER'**.

- In '$n$' bit parallel adder, minimum delay to provide the final result is equal to $2nt_{pd}$, (where $t_{pd}$ = propagation delay of each gate) that is why parallel adder becomes very slow.
- To overcome this difficulty a new type of adder is used which is called **"LOOK AHEAD CARRY ADDER"**.

**Example - 4.5** A one bit full adder takes 75 nsec to produce sum and 50 nsec to produce carry. A 4 bit parallel adder is designed using this type of full adder. The maximum rate of additions per second can be provided by 4 bit parallel adder is $A \times 10^6$ additions/sec. The value of $A$ is

***Solution:***

Minimum propagation delay = $(3 \times 50 + 75)$ nsec = 225 nsec

$$\text{Maximum additions per second} = \frac{1}{225\text{nsec}} = 4.44 \times 10^6 \text{ additions/sec} \Rightarrow A = 4.44$$

**The Look-ahead Carry Adder**

- In the case of parallel adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay.
- It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.
- Consider the full adder circuit shown in Figure 4.8.

  We have, $P_i = A_i \oplus B_i \Rightarrow$ Carry propagates

  $G_i = A_i B_i \Rightarrow$ Carry generates

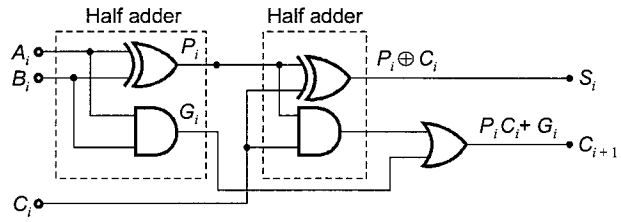



***Figure-4.8*** *Look ahead carry adder*

Thus, the output sum and carry can respectively be expressed as

$$\boxed{S_i = P_i \oplus C_i} \quad \text{and} \quad \boxed{C_{i+1} = G_i + P_i C_i}$$

Based on these, the expressions for sum and carry for four bit addition are:

$$C_0 = \text{input carry}; \quad C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 \quad \text{or} \quad C_2 = G_1 + P_1 (G_0 P_0 C_0)$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 \cdot [P_1 P_0 C_0 + P_1 G_0) + G_1]$$

or

$$C_3 = P_2 P_1 P_0 C_0 + P_2 P_1 G_0 + P_2 G_2 + G_2$$

Since, the boolean function for each output carry can be expressed in sum-of-product form, thus each function can be implemented with one level of AND gates followed by an OR gate.

- The construction of a four bit adder with a look ahead carry scheme is shown in Figure 4.9. It is clear that $C_3$ does not have to wait for $C_2$ and $C_1$ to propagate; infact $C_3$ is propagated at the same time as $C_2$ and $C_1$. So that delay for output carry must be reduced and the increase in the speed of operation is achieved at the expense of additional hardware complexity.

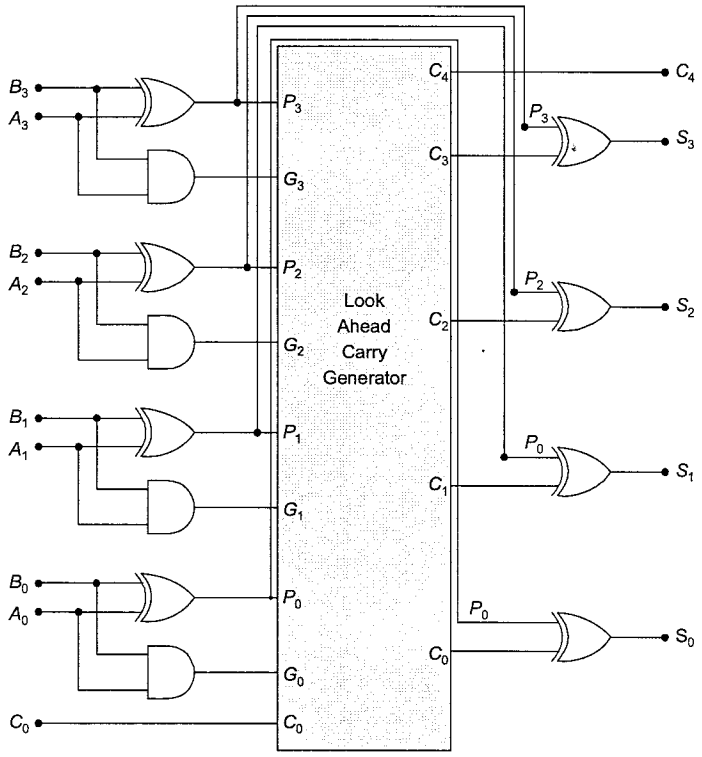



***Figure-4.9*** *A four bit adder with look ahead carry scheme*

**Remember**

- A Look-ahead-carry scheme for '$n$' bit addition requires $n$ OR gates and $\frac{n(n+1)}{2}$ AND gates.
- To generate the carry output, the carry circuit requires 3 logic gate delay and for sum output it requires 4 logic gate delay.

### The Serial Adder

- A serial adder is used to add two binary numbers in serial form.
- The two binary numbers to be added serially are stored in two shift registers designated as $A$ and $B$. The circuit adds one pair at a time with the help of one full adder circuit as shown in Figure 4.10. The carry output from the full adder is applied to a $D$ flip-flop, the output of which is then used as a carry input for the next pair of significant bits. However the sum bit $S$ from the output of the full adder can be transferred into a third shift register.

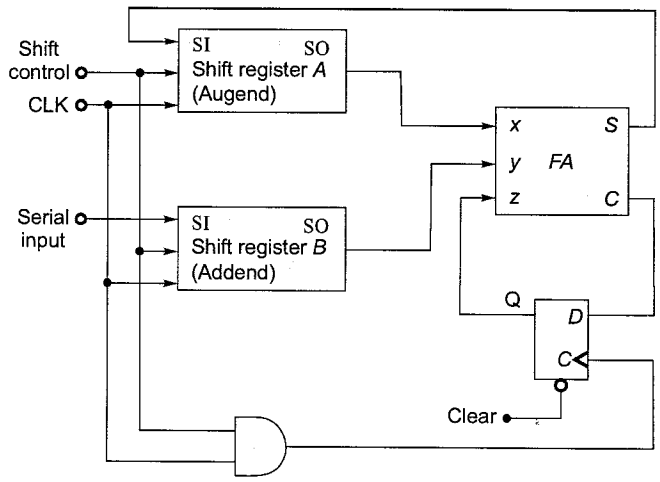



***Figure-4.10*** *Block diagram of serial adder*

### Difference between Serial adder and Parallel adder

**Table-4.3** *Difference between serial and parallel adder*

| Serial Adder | Parallel Adder |
|---|---|
| 1. A serial adder uses shift registers | 1. A parallel adder uses registers with parallel loads. |
| 2. A serial adder requires only one full adder and a carry flip flop. | 2. The numbers of full adder circuits in the parallel adder is equal to the number of bits in the binary number. |
| 3. Serial adder is a sequential circuit. | 3. Parallel adder, excluding registers, is a combinational circuit |

### The BCD Adder

The BCD addition can be performed in accordance with the following steps:

1. Add two 4-bit BCD code groups for each decimal digit position using straight binary addition.
2. For those positions where the sum is 9 (1001) or less, the SUM is in proper form and no correction is required.

3. When the sum of two digits is greater than 1001 (decimal 9), a correction of 0110 should be added to this sum and the generated carry must be added to the next decimal position.

Therefore unlike the binary parallel adder the circuitry for a BCD addition must include the logic needed to detect whenever the sum is greater than 1001 (decimal 9), so that the correction can be added in. Those cases; where the sum is greater than 1001 are listed in Table 4.4.

For example, if the two BCD code groups $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ are applied to a 4 bit adder, the adder will output $S_4 S_3 S_2 S_1 S_0$ where $S_4$ is actually $C_4$, the carry out of the MSB bits.

**Table-4.4**

| $S_4$ | $S_3$ | $S_2$ | $S_1$ | $S_0$ | Decimal Number |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 10 |
| 0 | 1 | 0 | 1 | 1 | 11 |
| 0 | 1 | 1 | 0 | 0 | 12 |
| 0 | 1 | 1 | 0 | 1 | 13 |
| 0 | 1 | 1 | 1 | 0 | 14 |
| 0 | 1 | 1 | 1 | 1 | 15 |
| 1 | 0 | 0 | 0 | 0 | 16 |
| 1 | 0 | 0 | 0 | 1 | 17 |
| 1 | 0 | 0 | 1 | 0 | 18 |

Let us define a logic output '$X$' that will go HIGH only when the sum is greater than 01001. Thus from the above table, it is clear that '$X$' will be HIGH for either of the following conditions.

1. Whenever $S_4 = 1$
2. Whenever $S_3 = 1$ and either $S_2$ or $S_1$ or both are 1.

This condition can be expressed as: $X = S_4 + S_3 (S_2 + S_1)$

Whenever '$X$' = 1, it is compulsory to add the correction factor 0110 to the sum bit, and to generate a carry.

A BCD adder that adds two BCD digits and produces a sum digit in BCD as shown in Figure 4.11. Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry of the second BCD adder is connected as the carry-in of the third BCD adder and so on.

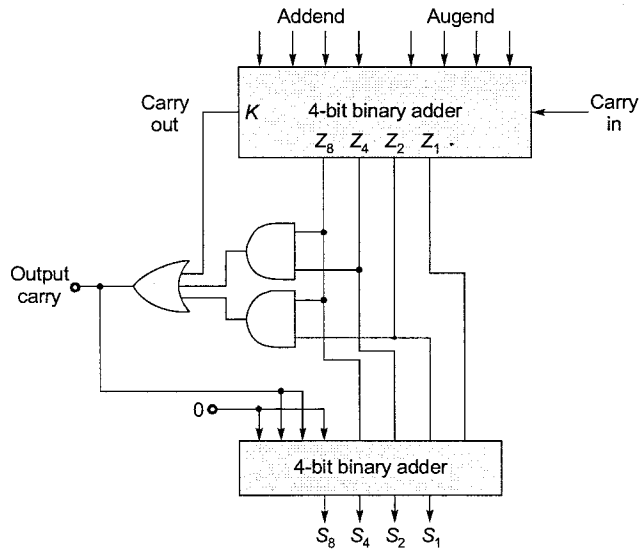



***Figure-4.11*** *Block diagram of a BCD adder*

### The Excess-3 Adder

The excess-3 adders are used for the addition of two excess-3 code groups. In this addition if carry = 1, add 0011 (decimal 3) to the sum of those two code groups. However if carry = 0, subtract 0011 (decimal 3) or add 1101 (decimal 13) to the sum of those two code groups.

*For example:* Add decimal numbers 8 and 6 in Excess-3.

⇒

```
8 in Excess-3  →   1 0 1 1
6 in Excess-3  →  +1 0 0 1
                  ①0 1 0 0
carry           0 0 1 1 0 0 1 1
                0 1 0 0 0 1 1 1
                  (1)     (4) = 14 in Excess-3
```

### 2's Complemented Adder

If we perform $(A - B)$ operation, then we take it as $A + (-B)$ i.e. $A$ (as it is) + (2's complement of $B$) For this purpose we use a control circuit by using Ex-OR gate. Consider a 4-bit 2's complement parallel adder as shown in Figure 4.12 below:

Let, $A = A_3\ A_2\ A_1\ A_0$

$B = B_3\ B_2\ B_1\ B_0$

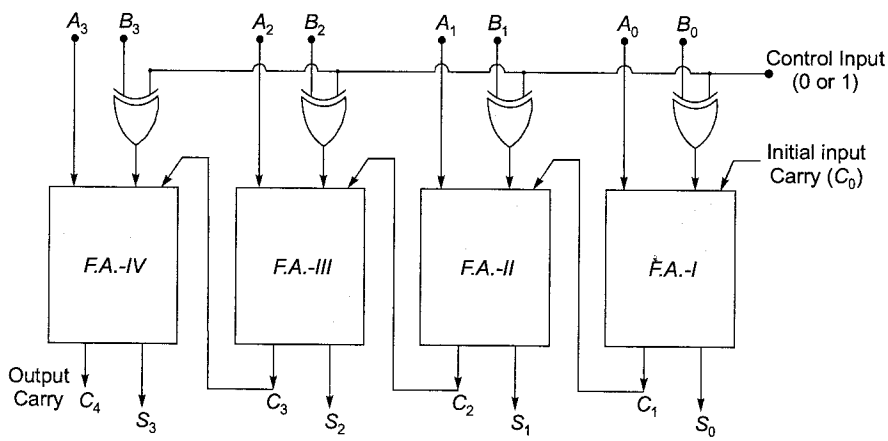
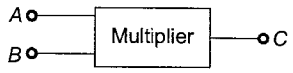



***Figure-4.12*** *2's Complemented Adder*

**NOTE**

- When **control input = "0"** ⇒ circuit acts as an **"ADDER"**.
- When **control input = "1"** ⇒ circuit acts as a **"SUBTRACTOR"**.

**Example - 4.6** Consider a 3-bit number $A$ and 2 bit number $B$ are given to a multiplier. The output of multiplier is realized using AND gate and one bit full adders. If minimum number of AND gates required are $X$ and one bit full adders required are $Y$, then $X + Y =$ _____.



***Solution:***

Let,

$$
\begin{array}{rcccc}
A = & & a_2 & a_1 & a_0 \\
B = & & & b_1 & b_0 \\
\hline
A \times B = & & a_2b_0 & a_1b_0 & a_0b_0 \\
 & b_1a_2 & b_1a_1 & b_1a_0 & \downarrow \\
\hline
 & b_1a_2 & (a_2b_0 + a_1b_1) & (a_1b_0 + b_1a_0) & a_0b_0 \\
 & C_3 & C_2 & C_1 & C_0
\end{array}
$$

Number of AND gates required $X = 6$

Number of one bit full adders required $Y = 3$

$$X + Y = 6 + 3 = 9$$

## 4.2.2 Subtractors

### The Half Subtractor

- A half subtractor is a combinational circuit that subtracts one bit from the other and produces the difference.
- A half subtractor circuit with two input variables designated as $A$ (minuend) and $B$ (subtractend); and output variables as Difference '$d$' and Borrow '$b$' is shown in Figure 4.13. The truth Table of half subtractor is listed in Table 4.5.

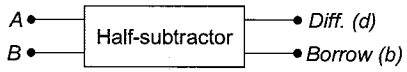
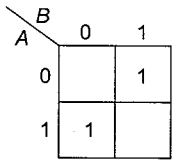
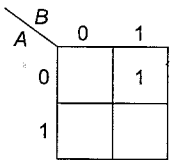



***Figure-4.13*** *Block diagram of half subtractor*

| Input | | Output | |
|---|---|---|---|
| A | B | Difference (d) | Borrow (b) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |




$d = A \oplus B$




$b = \bar{A}B$

***Table-4.5*** *Truth Table of half subtractor*

- The simplified Boolean expression for the two output can be obtained directly from the truth Table. The logical expressions for difference (d) and Borrow (b) are:

$$\boxed{d = \bar{A}B + A\bar{B} = A \oplus B} \quad \text{and} \quad \boxed{b = \bar{A}B}$$

- Implementation of half subtractor circuit:

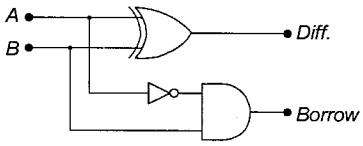



***Figure-4.13(a)*** *Logic diagram of half Subtractor*

- A half subtractor can also be realized using either only NAND gates or only NOR gates as shown in Figure 4.13 (b) and (c) respectively.

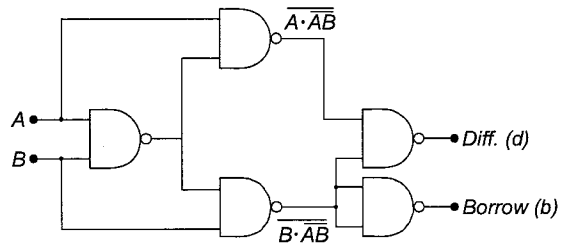



***Figure-4.13 (b)*** *Logic diagram of half Subtractor*

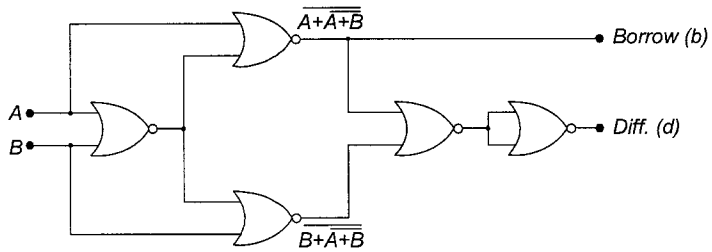



***Figure-4.13 (c)*** *Logic diagram of half Subtractor*

- Half subtractor/Half adder circuit by using control action of Ex-OR gate ( Figure 4.13 (d)):

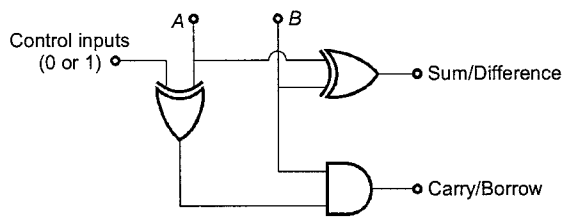



***Figure-4.13 (d)***

**Remember**

- When **Control** input = 0 ⇒ Half adder operation
- When **Control** input = 1 ⇒ Half subtractor operation.

**Example-4.7** For a binary subtractor having two inputs $A$ and $B$, the correct set of logical expressions for the output $D$ (= $A$ minus $B$) and $X$ (= Borrow) are:

(a) $D = AB + \bar{A}\bar{B}, X = \bar{A}B$

(b) $D = \bar{A}B + A\bar{B}, X = A\bar{B}$

(c) $D = \bar{A}B + A\bar{B}, X = \bar{A}B$

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

***Solution: (c)***

Truth Table for half subtractor

| $A$ | $B$ | $D$ (Difference) | $X$ (Borrow) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

∴ $D = \bar{A}B + A\bar{B} = A \oplus B$

$X = \bar{A}B$

**The Full Subtractor**

- A full subtractor is an arithmetic circuit which performs a subtraction between two bits taking into account that a '1' may have been borrowed by a lower significant stage. Thus a full subtractor has three inputs and two outputs.
- The block diagram of a full subtractor is shown in Figure 4.14. Table 4.6 represents the truth table for full subtractor.

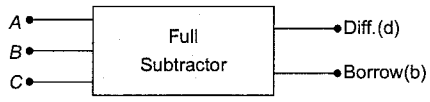
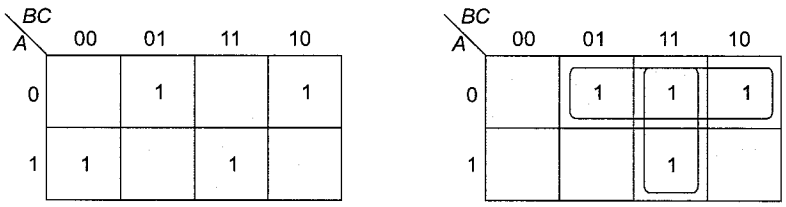



***Figure-4.14*** *Block diagram of a full subtractor*

| Input | | | Output | |
|---|---|---|---|---|
| *A* | *B* | *C* | *d* | *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 |

***Table-4.6*** *Truth Table for full subtractor*




(i) K-map for Difference (ii) K-map for Borrow

***Figure-4.15*** K-maps for full subtractor

- From the truth Table and the maps for the outputs of the full adder as shown in Table 4.6 and Figure 4.15 respectively, the logical expressions for Difference and Borrow are:

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

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

Also, $\boxed{d = A \oplus B \oplus C}$ or $\boxed{d = \Sigma_m(1, 2, 4, 7)}$

$\boxed{b = \bar{A}C + (\overline{A \oplus B}) \cdot C}$ or $\boxed{b = \Sigma_m(1, 2, 3, 7)}$

- **Implementation of full subtractor circuit:** From the above expression a full subtractor circuit can be realized in SOP form or with two half subtractor and an OR gate as shown in Figure 4.16 (a).

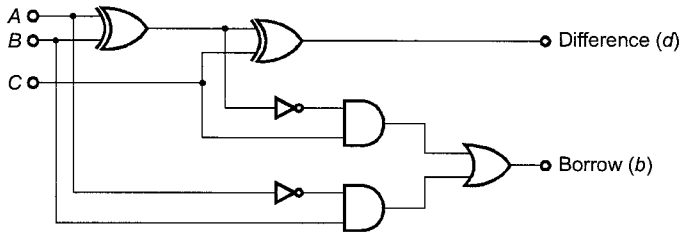



***Figure-4.16 (a)*** *Logic diagram of full subtractor in SOP form*

- The ANSI/IEEE standard logic diagram of full subtractor is shown as

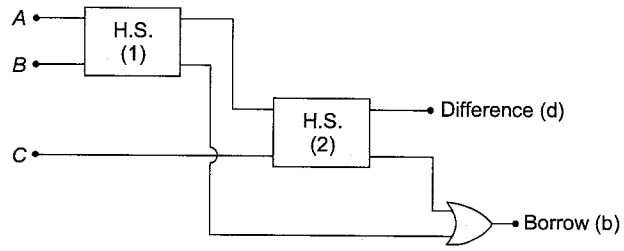



***Figure-4.16 (b)*** *ANSI/IEEE standard diagram for full subtractor*

**Do You Know**

The full subtractor can also be realized using universal logic i.e., either only NAND gates or only NOR gates. The total number of NAND gates/NOR gates required to implement a full subtractor is equal to "9".

## 4.3 Non-arithmetic Circuit

### 4.3.1 Code Converters

- The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different systems. Therefore a conversion circuit is inserted between the two system if each uses different codes for the same information. Hence a **"code converter"** is a logic circuit which changes data presented in one type of binary code to another type of binary code. It makes two systems compatible even though each uses a different binary code.
- Code converters are usually multiple output circuits.

**Binary to Gray Code Converter**

- The input to the 4-bit binary-to-gray code converter circuit is a 4-bit binary and the output is a 4-bit gray code. There are 16 possible combinations of 4-bit binary input and all of them are valid. Hence no don't care.
- The 4-bit binary and the corresponding gray code are shown in the conversion Table 4.7.

**Table-4.7** *Conversion of 4-bit binary to gray code*

| Binary code | | | | Gray Code | | | |
|---|---|---|---|---|---|---|---|
| $B_3$ | $B_2$ | $B_1$ | $B_0$ | $G_3$ | $G_2$ | $G_1$ | $G_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

- From the conversion Table, we observe that the expressions for the outputs are:

$$G_3 = \Sigma_m(8, 9, 10, 11, 12, 13, 14, 15); \quad G_2 = \Sigma_m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$G_1 = \Sigma_m(2, 3, 4, 5, 10, 11, 12, 13); \quad G_0 = \Sigma_m(1, 2, 5, 6, 9, 10, 13, 14)$$

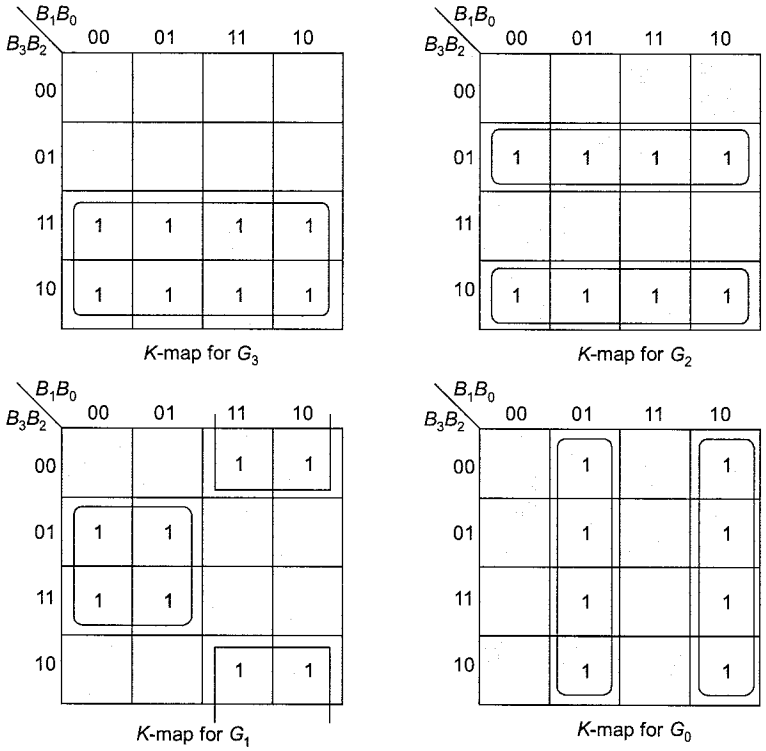



***Figure-4.17(a)*** *K-map for Binary-to-Gray code converter*

- We have, the minimal expressions for the output obtained from the K-map as:

$$G_3 = B_3; \quad G_2 = B_3 \oplus B_2$$
$$G_1 = B_2 \oplus B_1; \quad G_0 = B_1 \oplus B_0$$

- Based on the above expressions, a circuit can be drawn as shown in Figure 4.17(b)

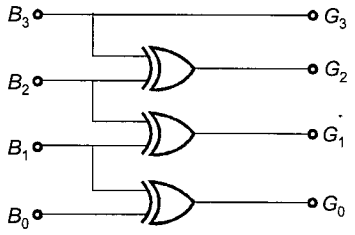



***Figure-4.17(b)*** *Logic Diagram of Binary-to-Gray Code Converter*

- Thus, the basic conversion concept is,

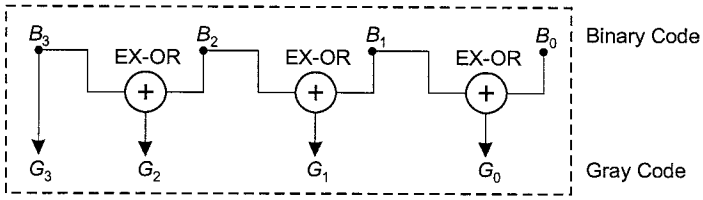



***Figure-4.17 (c)*** *Conversion Concept for Binary-to-Gray Code converter*

**Gray-to-Binary code Converter**

- The input to the 4-bit gray to binary code converter circuit is a 4-bit gray code and the output is a 4-bit binary. Thus, total 16 combinations are possible and all of them are valid. Hence no don't care.
- The 4-bit input gray code and the corresponding output binary numbers are shown in the conversion Table 4.8.

**Table-4.8** *Conversion of 4-bit Gray-to-Binary code*

| Gray Code | | | | Binary Code | | | |
|---|---|---|---|---|---|---|---|
| $G_3$ | $G_2$ | $G_1$ | $G_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

- From the conversion Table, we observe that the expression for the outputs are:

$$B_3 = \Sigma_m(8, 9, 10, 11, 12, 13, 14, 15); \quad B_2 = \Sigma_m(4, 5, 6, 7, 8, 9, 10, 11)$$

$$B_1 = \Sigma_m(2, 3, 4, 5, 8, 9, 14, 15); \quad B_0 = \Sigma_m(1, 2, 4, 7, 8, 11, 13, 14)$$

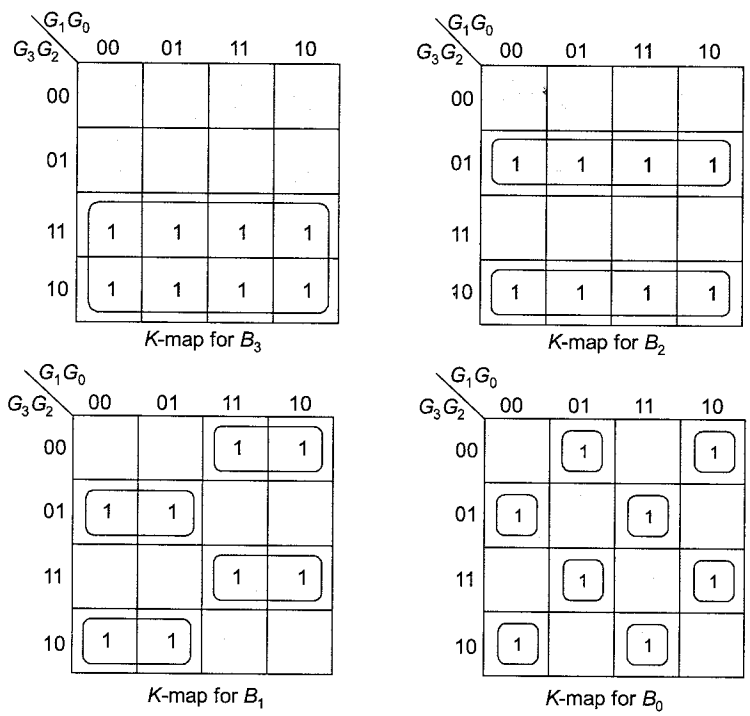



***Figure-4.18 (a)*** *K-map for Gray-to-Binary code conversion*

- We have, the minimal expressions for the outputs obtained from the K-map as

$$B_3 = G_3; \qquad B_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1; \qquad B_0 = B_1 \oplus G_0$$

- Based on the above expressions, a logic circuit can be drawn as shown in Figure 4.18(b).

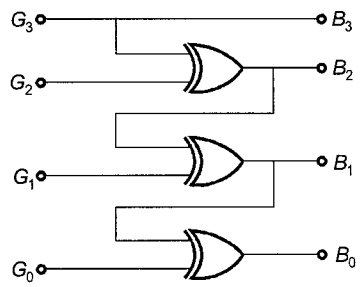



***Figure-4.18 (b)*** *Logic Diagram of Gray-to-Binary Code Converter*

- Thus, the basic conversion concept is

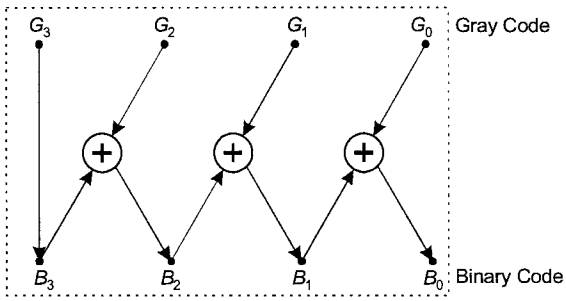



***Figure-4.18 (c)*** *Conversion concept for Gray-to-Binary code conversion*

**BCD to Excess-3 Converter**

- *BCD* code is basically 8421 code, hence 4-bit input *BCD* code (*A B C D*) is converted into the excess-3 code output (*W X Y Z*) as shown in conversion Table 4.9. The input combinations 1010, 1011, 1100, 1101, 1110 and 1111 are invalid in *BCD* therefore they are treated as don't cares.

**Table-4.9** *BCD to Excess-3 code converter*

| BCD Code | | | | Excess-3 Code | | | |
|---|---|---|---|---|---|---|---|
| *A* | *B* | *C* | *D* | *W* | *X* | *Y* | *Z* |
| 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 |

- From the conversion Table, we observe that the expression for the outputs are:

$$W = \Sigma_m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$
$$X = \Sigma_m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$$
$$Y = \Sigma_m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$$
$$Z = \Sigma_m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$$

Drawing the K-map for the above expressions

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | 1 | 1 | 1 |
| 11 | X | X | X | X |
| 10 | 1 | 1 | 1 | 1 |

*K*-map for *W*
$W = A + BC + BD$

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | 1 | 1 | 1 |
| 01 | 1 | | | |
| 11 | X | X | X | X |
| 10 | | 1 | X | X |

*K*-map for *X*
$X = B\bar{C}\bar{D} + \bar{B}D + \bar{B}C$

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | 1 | |
| 01 | 1 | | 1 | |
| 11 | X | X | X | X |
| 10 | 1 | | X | X |

*K*-map for *Y*
$Y = \bar{C}\bar{D} + CD$

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | | 1 |
| 01 | 1 | | | 1 |
| 11 | X | X | X | X |
| 10 | 1 | | X | X |

*K*-map for *Z*
$Z = \bar{D}$

***Figure-4.19 (a)*** *K-map for BCD to Excess-3 converter*

- Thus, we have the minimal expressions for the outputs obtained from K-map as

$$Z = \bar{D}$$
$$Y = \bar{C}\bar{D} + CD$$
$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$
$$W = A + BC + BD$$

- Based on the above expressions, a logic circuit can be drawn as shown in Figure 4.19 (b).

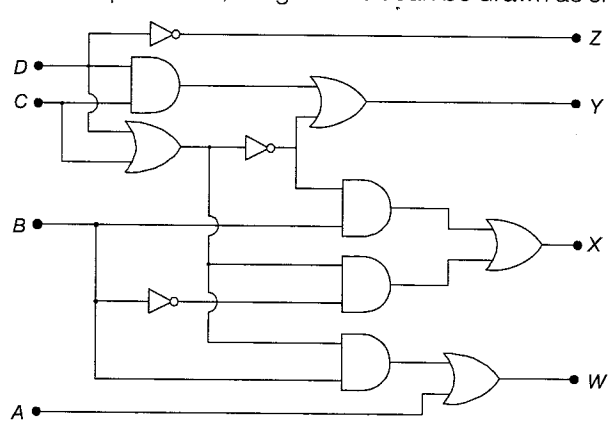



***Figure-4.19 (b)*** *Logic Diagram of BCD to Excess-3 Converter*

**Example - 4.8** The minimal function that can detect a "divisible by 3" 8421 *BCD* code digit (representation is $D_8\ D_4\ D_2\ D_1$) is given by

(a) $D_8 D_1 + D_4 D_2 + \overline{D_8} D_2 D_1$

(b) $D_8D_1 + D_4D_2\bar{D}_1 + \bar{D}_4D_2D_1 + \bar{D}_8\bar{D}_4\bar{D}_2\bar{D}_1$

(c) $D_8D_1 + D_4D_2 + \bar{D}_8\bar{D}_4\bar{D}_2\bar{D}_1$

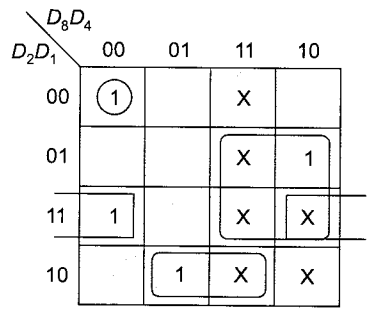
(d) $D_4D_2\bar{D}_1 + D_4D_2D_1 + D_8\bar{D}_4D_2D_1$

***Solution: (b)***

Truth Table:

| | $D_8$ | $D_4$ | $D_2$ | $D_1$ | $Y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 |

K-Map:




$Y = D_8D_1 + D_4D_2\bar{D}_1 + \bar{D}_4D_2D_1 + \bar{D}_8\bar{D}_4\bar{D}_2\bar{D}_1$

**Example - 4.9** Design a combinational circuit that accepts a 3-bit number as input and generates an output binary number equal to square of the input number.

***Solution:***

Largest 3 bit number is = $(111)_2 = (7)_{10}$

So square of $(7)^2 = (49)_{10} = (110001)_2$, that is, it requires 6 bits to represent output.

Let these outputs are: $Y_5, Y_4, Y_3, Y_2, Y_1, Y_0$

| | Input | | | Output | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $A$ | $B$ | $C$ | $Y_5$ | $Y_4$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 6 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Therefore,

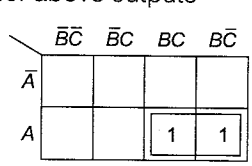
$Y_5 = \Sigma(6, 7)$; $Y_4 = \Sigma(4, 5, 7)$

$Y_3 = \Sigma(3, 5)$; $Y_2 = \Sigma(2, 6)$

$Y_1 = 0$; $Y_0 = \Sigma(1, 3, 5, 7)$

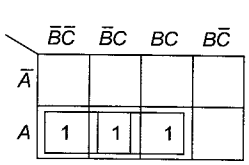
Solving for above outputs

$Y_5 \Rightarrow$



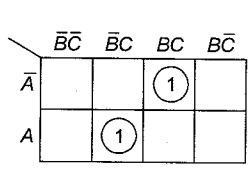

$Y_5 = AB$

$Y_4 \Rightarrow$




$Y_4 = A\bar{B} + AC$

$Y_3 \Rightarrow$




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

$Y_2 \Rightarrow$

| | $\bar{B}\bar{C}$ | $\bar{B}C$ | $BC$ | $B\bar{C}$ |
|---|---|---|---|---|
| $\bar{A}$ | | | | 1 |
| $A$ | | | | 1 |

$Y_2 = B\bar{C}$

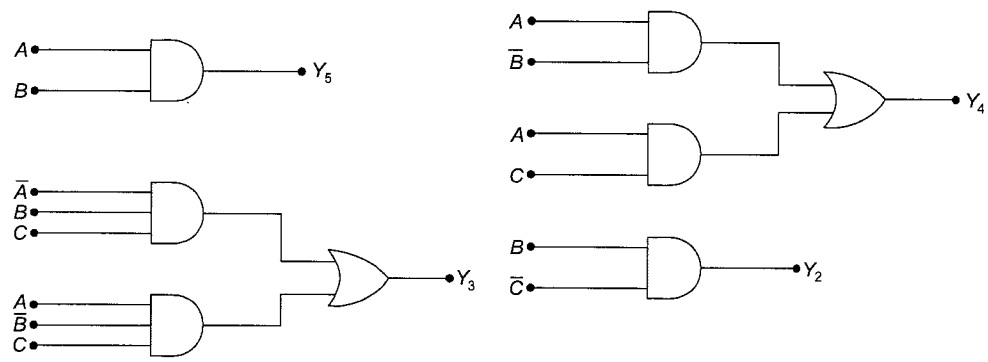
$Y_1 = 0$

$Y_0 \Rightarrow$

| | $\bar{B}\bar{C}$ | $\bar{B}C$ | $BC$ | $B\bar{C}$ |
|---|---|---|---|---|
| $\bar{A}$ | | 1 | 1 | |
| $A$ | | 1 | 1 | |

$Y_0 = C$

Now the circuit will be




$$Y_1 = 0 \quad \text{and} \quad Y_0 = C$$

Assuming inverted inputs are available, also 3-input AND gates are available.

### 4.3.2 Magnitude Comparator

- A magnitude comparator is a combinational circuit which is used to compare two numbers $A$ and $B$ and determines their relative magnitudes.
- Let $A$ and $B$ are the two inputs for which the output of the comparator is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$.

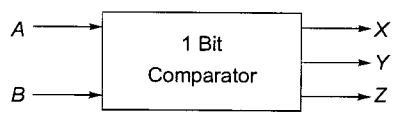



***Figure-4.20*** *Block diagram of 1-bit comparator*

- The EX-NOR gate is a basic comparator, because its output is a 1 only if its input bits coincide. For example, two 4-bit binary numbers, $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ are equal if and only if, $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$. Thus, the implementation of this logic:

$$\text{Equality} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

It is clear that this circuit can be expanded or compressed to accommodate binary numbers with any other number of bits.

- The block diagram of a 1-bit comparator is shown in Figure 4.20 and the truth table and the logic circuit for 1-bit comparator is shown in Table 4.10 and Figure 4.21 respectively.

| $A$ | $B$ | $X$ | $Y$ | $Z$ | Output |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | $A = B$ |
| 0 | 1 | 1 | 0 | 0 | $A < B$ |
| 1 | 0 | 0 | 0 | 1 | $A > B$ |
| 1 | 1 | 0 | 1 | 0 | $A = B$ |

***Table-4.10*** *Truth table for 1-bit comparator*

*from the truth table it is clear that*

if $A = 1$ and $B = 0$, then $A > B$

then, $A > B$; $X = A\bar{B}$

if $A = 0$ and $B = 1$, then $A < B$

then, $A < B$; $X = \bar{A}B$

if $A$ and $B$ coincide, i.e., $A = B = 0$ or if $A = B = 1$, then $A = B$

then, $A = B$; $Y = A \odot B$

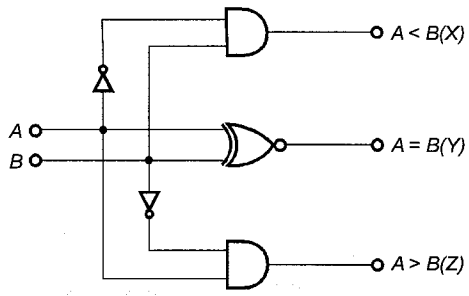



***Figure-4.21*** *Logic circuit for 1-bit comparator*

- **Applications of Magnitude Comparator:** These are often used as part of the address decoding circuitry used in a computer to select a specific **"input-output device"** or area of memory for the storage or retrieval of data. It is also useful in control applications where a binary number representing the physical variable being controlled (e.g. position, speed) is compared with a reference value.

**Example-4.10** Implement a circuit which gives 4-bit equality condition.

***Solution:***

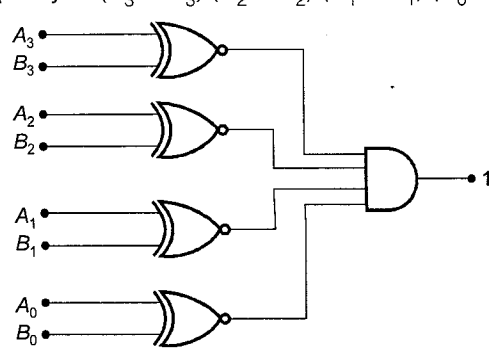
Let, the two inputs as

$$A = A_3\, A_2\, A_1\, A_0 \quad \text{and} \quad B = B_3\, B_2\, B_1\, B_0$$

Equality condition holds when $A_3$ coincides with $B_3$, $A_2$ coincides with $B_2$, $A_1$ coincides with $B_1$ and $A_0$ coincides with $B_0$.

Therefore the implementation of the circuit involves.

$$\text{Equality} = (A_3 \odot B_3)\,(A_2 \odot B_2)\,(A_1 \odot B_1)\,(A_0 \odot B_0)$$




**Example-4.11** The output $Y$ of a 2-bit comparactor is logic 1 whenever the 2-bit input $A$ is greater than the 2-bit input B. The number of combination for which the output is logic 1, is

(a) 4 (b) 6

(c) 8 (d) 10

***Solution: (b)***

Ouptut will be 1 if $A > B$.

If $B = 00$ then there will be three combinations for which output will be 1 i.e. when $A = 01$, 10 or 11.

If $B = 01$ there will be two conditions i.e. $A = 10$ and 11.

If $B = 10$ there will be one condition i.e. $A = 11$.

So, total 6 combinations are there for which output will be 1.

### 4.3.3 Multiplexers (MUX)

- The 'Multiplexer' is a special, versatile and one of the most widely used standard circuit in digital design.
- It is a combinational circuit that selects binary information from one of many input lines and directs it to single output line.
- The selection of a particular input line is controlled by a set of selection lines. Generally, there are '$2^n$' input lines and '$n$' selection lines whose bit combinations determine which input is selected.
- Multiplexer can also be named as
  - (*i*) Data selector
  - (*ii*) Many to one circuit
  - (*iii*) Universal logic converter
  - (*iv*) Parallel to serial converter
  - (*v*) Waveform generator
- If '$m$' be the total number of data input and '$n$' be the number of selection lines, then

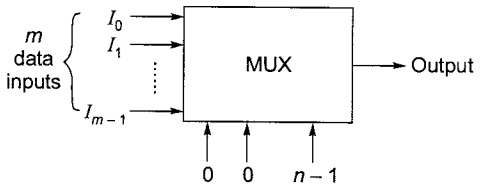



***Figure-4.22*** *Functional diagram of a digital multiplexer*

**The Basic 2-line to 1-line MUX**

Figure 4.23 shows the logic circuitry and function table for 2 : 1 MUX. It has two input lines as $I_0$ and $I_1$ and one data select line $S$. The output line is designated as $Y$.

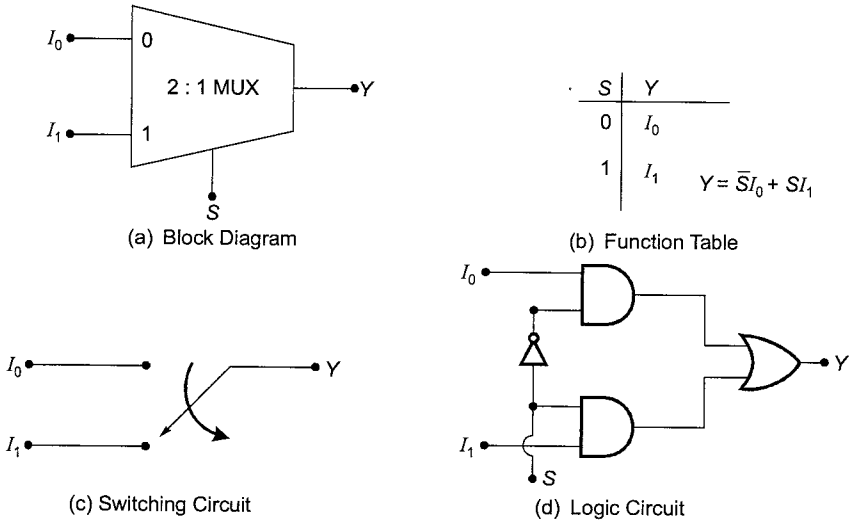
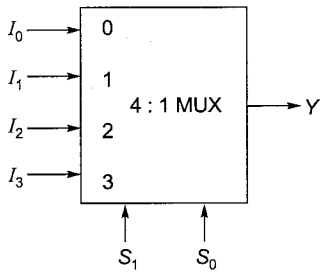



***Figure-4.23*** *2 : 1 Multiplexer*

**The 4-line to 1-line MUX**

A 4 : 1 line multiplexer is shows in Figure 4.24. Each of the four inputs, $I_0$, $I_1$, $I_2$ and $I_3$ are applied to the input of MUX and logic levels applied to the selection lines $S_0$ and $S_1$.

The logic circuitry based on the function table is shown in Figure 4.25.




(a) Block Diagram

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

(b) Function Table

***Figure-4.24*** *4 : 1 Multiplexer*

from the function table, we have

$$Y = \overline{S_1}\,\overline{S_0}\,I_0 + \overline{S_1}\,S_0\,I_1 + S_1\,\overline{S_0}\,I_2 + S_1\,S_0\,I_3$$

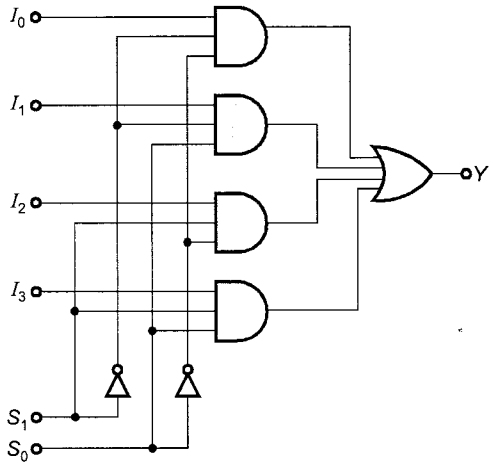
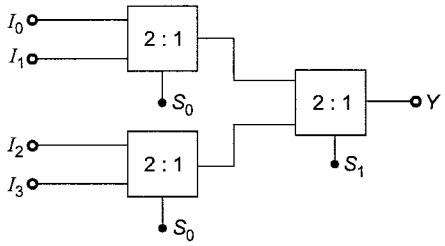



***Figure-4.25*** *Logic circuit of 4 : 1 MUX*

- Implementation of higher order using lower order MUX:

  4 : 1 MUX by 2 : 1 MUX




  8 : 1 MUX by 2 : 1 MUX

$$\frac{8}{2} = 4\text{ MUX} \Rightarrow \frac{4}{2} = \text{MUX} \Rightarrow \frac{2}{2} = 1\text{ MUX}$$

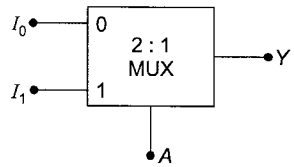
$\therefore$ Total number of 2 : 1 MUX required = 4 + 2 + 1 = 7

| Given MUX | To be implemented | Required |
|---|---|---|
| 2 : 1 | 16 : 1 | 8 + 4 + 2 + 1 = 15 |
| 4 : 1 | 16 : 1 | 4 + 1 = 5 |
| 4 : 1 | 64 : 1 | 16 + 4 + 1 = 21 |
| 8 : 1 | 64 : 1 | 8 + 1 = 9 |
| 8 : 1 | 256 : 1 | 32 + 4 + 1 = 37 |

**NOTE:** To implement $2^n$ : 1 MUX by using 2 : 1 MUX, the total number of 2 : 1 MUX required is $(2^n - 1)$

**MUX as an Universal Logic Gate**

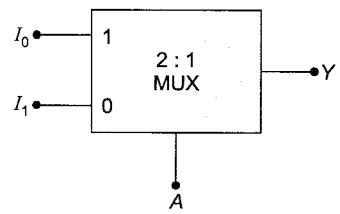
- Buffer:




$Y$ = Output = $A$

- NOT gate/Invertor:

As we know,




$$Y = \overline{S_0} I_0 + S_0 I_1$$

$$Y = \overline{A} I_0 + A I_1$$

$\because$ $Y = \overline{A}$ For NOT gate

$\therefore$ $I_0 = 1$ and $I_1 = 0$

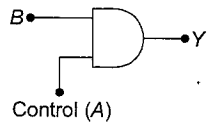
- AND gate:

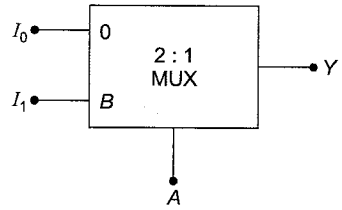
For AND operation

$$Y = AB$$

$$Y = \overline{S_0} I_0 + S_0 I_1$$

Logic




for, $A = 0$ : $Y = 0 = I_0$

for $A = 1$ : $Y = 0 = I_1$

$$Y = \overline{A} \cdot 0 + AB$$

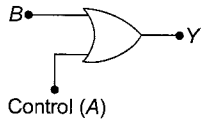
So, $I_0 = 0$ and $I_1 = B$

- OR gate:

For OR operation

$$Y = A + B$$

$\therefore$



for, $A = 0 : Y = B = I_0$

for $A = 1 : Y = 1 = I_1$

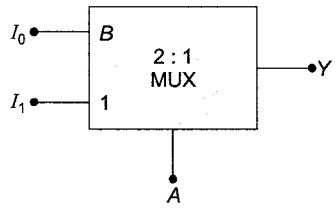
Therefore,

$$Y = \overline{S_0} I_0 + S_0 I_1$$

$$Y = \overline{A} I_0 + A I_1$$

$$Y = \overline{A} \cdot B + A \cdot 1$$

So, $I_0 = B$ and $I_1 = 1$

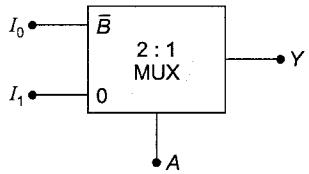



- **NOR gate:**

For NOR operation $Y = \overline{A+B}$

i.e., complement of OR gate.

Hence, required MUX is 2, one to perform OR operation and other one to perform complement.



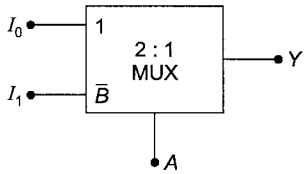

$\therefore \quad I_0 = \overline{B}$ and $I_1 = 0$

- **NAND gate:**

For NAND operation, $Y = \overline{AB}$

i.e., complement of AND gate.

Hence, required MUX is 2, one to perform AND operation and other one to perform complement.

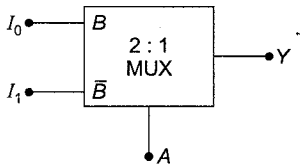



$\therefore \quad I_0 = 1$ and $I_1 = \overline{B}$

- **EX-OR gate:**

For Ex-OR operation

$$Y = A \oplus B = \overline{A}B + A\overline{B} \text{ and } Y = \overline{S_0} I_0 + S_0 I_1 = \overline{A} I_0 + A I_1$$




$\therefore \quad I_0 = B$ and $I_1 = \overline{B}$

Number of MUX required = 2

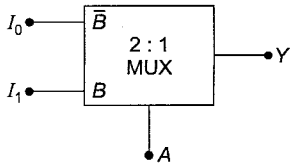
- ***EX-NOR* gate:**

For Ex-NOR operation

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

$$\because \quad Y = \overline{A} I_0 + A I_0$$

Number of required MUX = 2.

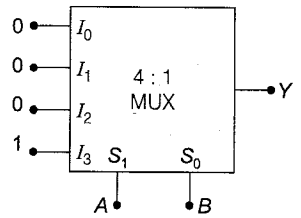



$\therefore \quad I_0 = \overline{B}$ and $I_1 = B$

- In order to implement logic gates with 4 : 1 MUX, truth table is required.
  *For example,*
  AND gate

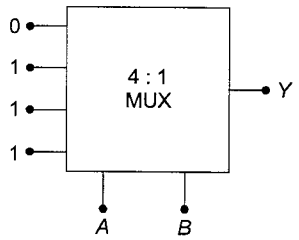
Truth table:

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

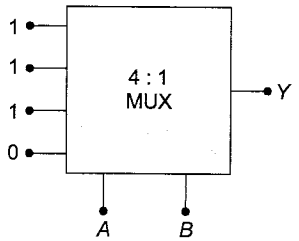



Similarly,

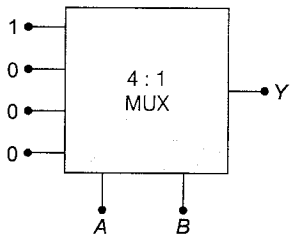
OR gate:



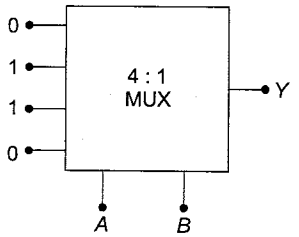

NAND gate:



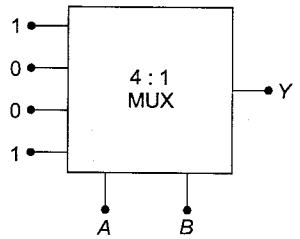

NOR gate:




EX-OR gate:




EX-NOR gate:




**NOTE**

- For implementation of basic gates, the required number of 2 : 1 MUX is 1
- For implementation of universal and special purpose gates the required number of 2 : 1 MUX is 2.

**Boolean Function Implementation by using MUX**

For implementing any boolean function of $n$-variables with a $2^n$ : 1 MUX, we required some general procedure.

(*i*) Express the function in its sum of product (SOP) form.

(*ii*) In the ordered sequence of $n$-variables, connect $(n-1)$ variables to the select line and the single highest order position variable to the input line with complemented or uncomplement form including 0 and 1.

(*iii*) List the inputs of MUX (all the minterms) in two rows. The 1st row lists all those minterms where single variable is complemented and in 2nd row with uncomplemented form.

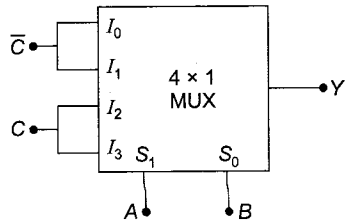
(*iv*) Circle all the minterms of the function and inspect each column seperately.

(*v*) If two minterms in a column are not circled, apply '0' to the corresponding MUX input.

(*vi*) If two minterms are circled, apply '1' to the corresponding MUX input.

(*vii*) If one minterm is circled (either upper row or lower row), then its front value is the corresponding MUX input.

**Example - 4.12** For given 4 : 1 MUX, the minimize expression is



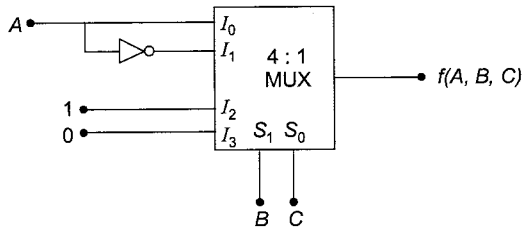

(a) $Y = A \oplus C$
(b) $Y = A \odot C$
(c) $Y = B \odot C$
(d) $Y = B \oplus C$

***Solution: (b)***

$$Y = \overline{S_0}\,\overline{S_1} I_0 + S_0 \overline{S_1} I_1 + \overline{S_0} S_1 I_2 + S_0 S_1 I_3$$
$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A} B \overline{C} + A \overline{B} C + ABC$$
$$= \overline{A}\,\overline{C}(\overline{B} + B) + AC(\overline{B} + B)$$
$$= \overline{A}\,\overline{C} + AC = A \odot C$$

| A | B | Y |
|---|---|---|
| 0 | 0 | $\overline{C}$ |
| 0 | 1 | $\overline{C}$ |
| 1 | 0 | C |
| 1 | 1 | C |

**Example - 4.13** The output of the given MUX equals to




(a) $f(A, B, C) = \Sigma m(1, 2, 4, 6)$
(b) $f(A, B, C) = \Sigma m(1, 2, 6)$
(c) $f(A, B, C) = \Sigma m(2, 4, 5, 6)$
(d) $f(A, B, C) = \Sigma m(1, 5, 6)$

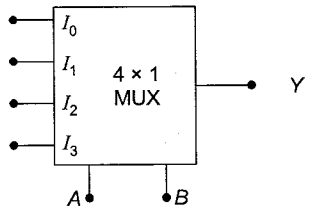
***Solution: (a)***

From the given 4 : 1 MUX we get,

$$f(A, B, C) = A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + B\overline{C} \cdot 1$$
$$= A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + B\overline{C}(A + \overline{A})$$
$$= A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + AB\overline{C} + \overline{A} B \overline{C}$$
$$\approx 100, \quad 001, \quad 110, \quad 010$$
$$\boxed{f(A, B, C) \approx \Sigma m(1, 2, 4, 6)}$$

| B | C | f |
|---|---|---|
| 0 | 0 | A |
| 0 | 1 | $\overline{A}$ |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Example - 4.14** Implement $f(A, B, C) = \Sigma m(0, 1, 4, 6, 7)$ by using the MUX given below



***Solution:***

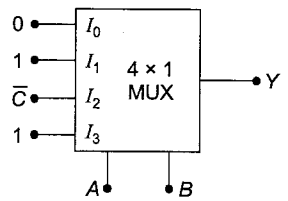
For 3 variable function, the truth table is

| | A | B | C | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 |

Let '$A$' and '$B$' are the select lines and '$C$' be the input

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| $\overline{C}$ | 0 | 2 | 4 | 6 |
| $C$ | 1 | 3 | 5 | 7 |
| | 1 | 0 | $\overline{C}$ | 1 |

Thus, for the implementation of given logical function, required is one 4 × 1 MUX and an Inverter.




**Application of MUX**

Multiplexer finds numerous applications in digital systems of all types. These applications include:

(*i*) Data selection
(*ii*) Data routing
(*iii*) Operation sequencing
(*iv*) Parallel-to-serial conversion
(*v*) Waveform generation
(*vi*) Logic function generation

**NOTE**

- By using 4 : 1 MUX any two variable function and some of three variable functions can be implemented. However using 4 : 1 MUX and a NOT gate all two variables as well as all three variable functions can be implement.
- By using 8 : 1 MUX any three variable function and some of four variable functions can be implement. However using 8 : 1 MUX with a NOT gate all three variables as well as four variable functions can be implemented.

**Example - 4.15** Consider a multiplexer with $X$ and $Y$ as data inputs and $Z$ as control input. $Z = 0$ selects input $X$ and $Z = 1$ selects input $Y$. What are the connections required to realize the 2-variable Boolean function $f = T + R$, without using any additional hardware?

(a) $R$ to $X$, 1 to $Y$, $T$ to $Z$
(b) $T$ to $X$, $R$ to $Y$, $T$ to $Z$
(c) $T$ to $X$, $R$ to $Y$, 0 to $Z$
(d) $R$ to $X$, 0 to $Y$, $T$ to $Z$

***Solution: (a)***

Multiplexer function, $f = f = X\overline{Z} + YZ$

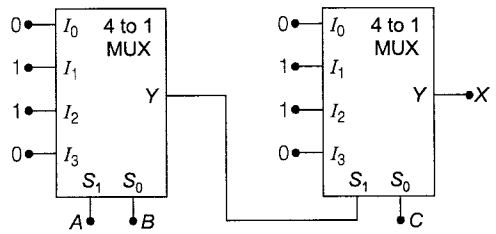
Given Boolean function, $f = T + R$

Let $X = R,\ Y = 1$ and $Z = T$

$$f = R\bar{T} + 1.T = R\bar{T} + T = R\bar{T} + T(1+R) = R\bar{T} + TR + T$$

$$= R(\bar{T} + T) + T = R + T$$

**Example-4.16** In the below circuit, $X$ is given by



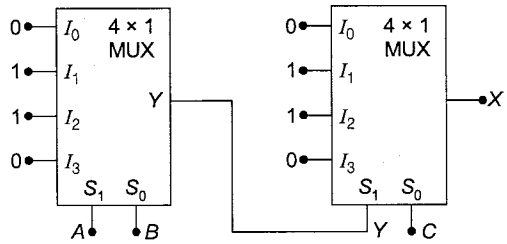

(a) $X = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$

(b) $X = \bar{A}BC + A\bar{B}C + AB\bar{C} + \overline{ABC}$

(c) $X = AB + BC + AC$

(d) $X = \overline{AB} + \overline{BC} + \overline{AC}$

***Solution: (a)***




| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$\Rightarrow\ Y = \bar{A}B + A\bar{B} = A \oplus B$ and

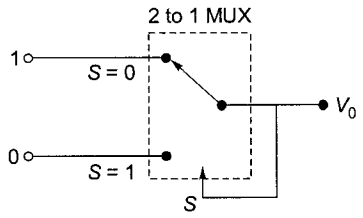
| Y | C | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$\Rightarrow\ X = \bar{Y}C + Y\bar{C} = Y \oplus C$

$$X = \overline{A \oplus B}\,C + (A \oplus B)\bar{C} \quad \Rightarrow \quad X = (\bar{A}\bar{B} + AB)C + (\bar{A}B + A\bar{B})\bar{C}$$

$$\Rightarrow \quad X = \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C} \quad \Rightarrow \quad X = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

**Example-4.17** A 2-to-1 digital multiplexer having a switching delay of 1 ms is connected as shown in Figure. The output of the multiplexer is tied to its own select input $S$. The input which gets selected when $S = 0$ is tied to 1 and the input that gets selected when $S = 1$ is tied to 0. The output $V_0$ will be




(a) 0

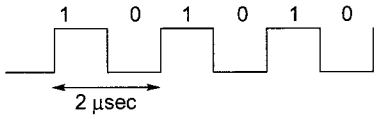
(b) 1

(c) pulse train of frequency 0.5 MHz

(d) pulse train of frequency 1.0 MHz

***Solution: (c)***

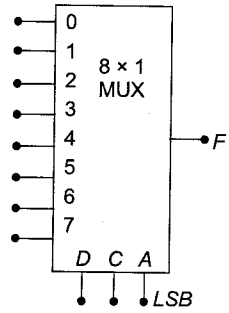
When output is 1, then $S$ will be also 1 and swtich will go to 1 after 1 msec and output become zero, again $S$ becomes 0 and switch will go to 0 after 1 msec and output goes to 1 this will continue and we will get the waveform at output as



so, $T = 2\,\mu\text{sec}$

then, $f = \dfrac{1}{T} = \dfrac{1}{2\times10^{-6}} = 0.5\times10^{6} = 0.5\text{ MHz}$

**Example-4.18** The logic function $F = AC + ABD + ACD$ is to be realized using an 8 to 1 multiplexer shown in the Figure, using $A$, $C$ and $D$ as control inputs.




(*i*) Indicate the inputs to the applied at the terminals 0 to 7.

(*ii*) Can the function be realize using a 4 to 1 multiplexer?

***Solution:***

(*i*) $F = AC + ABD + ACD$

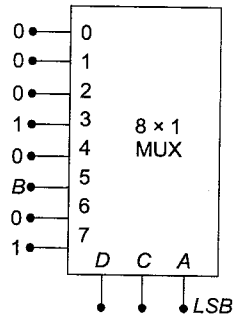
$= ACB + AC\bar{B} + ABDC + ABD\bar{C} + ACD\bar{B}$

$= ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}CD + ABCD + AB\bar{C}D$

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

$= \underbrace{\bar{D}CA}_{3^{rd}\text{ line}}(B+\bar{B}) + \underbrace{DCA}_{7^{th}\text{ line}}(B+\bar{B}) + \underbrace{D\bar{C}A}_{5^{th}\text{ line}}B$

$= \bar{D}CA(1) + DCA(1) + D\bar{C}A\cdot(B)$




(*ii*) It can not be realized with a 4 to 1 multiplexer as $5^{th}$ and $7^{th}$ lines are being used.

### 4.3.4 Decoders

- A decoder have many inputs and many outputs.
- It is a combinational logic circuit that converts binary information from '$n$' bit input lines to a maximum '$2^n$' unique output lines such that only one output line is activated for each one of possible combinations of input.
- Decoders are used to identify or convert a particular code for example:
  1. Binary to Octal (3-8 line decoders)
  2. Binary to Hexadecimal (4-16 line decoders)
  3. BCD to decimal (4-10 line decoders)
  4. BCD to 7-segment display
- If the $n$-bit decoded information has unused or don't care combinations, the decoder output will have less than $2^n$ outputs.

  i.e. if, $n$ = total number of input lines.

  $m$ = total number of output lines.

  then, $m \le 2^n$
- Decoders are widely used in memory system of computers.

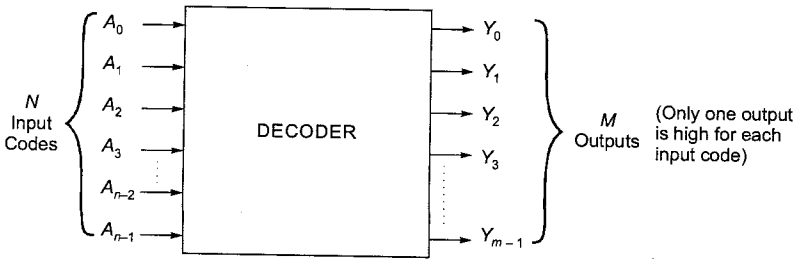
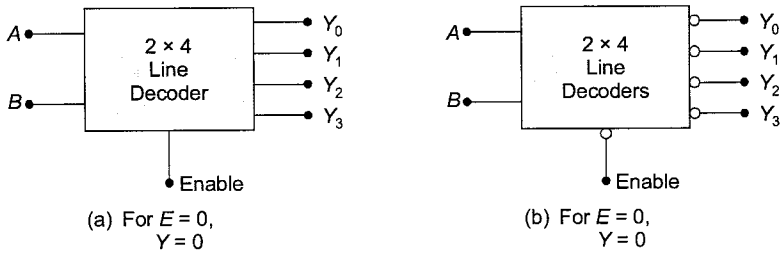



***Figure-4.26*** *General block diagram of a decoder*

### The 2 × 4 Line Decoder

- 2 × 4 decoders are minimum possible decoders.
- Let $A$ and $B$ are the two inputs and $Y_0$, $Y_1$, $Y_2$ and $Y_3$ are the four outputs.




(a) For $E = 0$, $Y = 0$ (b) For $E = 0$, $Y = 0$

***Figure-4.27*** *Two to four line decoder with enable input*

- For the Figure 4.27(b), we have,

| $E$ | $A$ | $B$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|---|
| 1 | x | x | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |

***Table-4.11*** Truth table for 2 × 4 line decoder

- Some decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their *complemented form. Furthermore, decoders include one or more enable inputs to control the circuit* operation. Figure 4.28 shows a 2 × 4 line decoder with an enable input using NAND gates. The circuit operated with complemented outputs and complemented enable input.

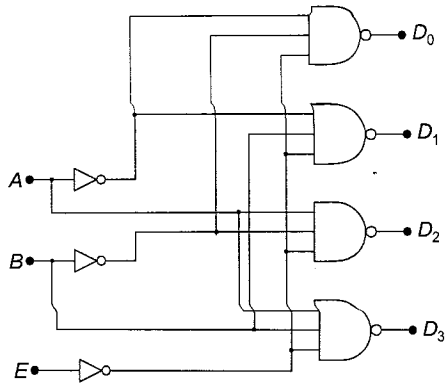



***Figure-4.28*** *Logic diagram of 2 × 4 line decoder*

**Remember**

- The internal circuitry of decoder and DEMUX are exactly same.
- A decoder with enable input can function as a demultiplexer.
- 2 × 4 line decoder may acts like a 1 : 4 DEMUX and vice-versa.
- Decoder contains AND-gates or NAND-gates

- A half adder or a half subtractor circuit can be implemented by 2 × 4 decoder and OR gate.
  *For example:* Implementation of half adder using 2 × 4 line decoder
  As we know, for half adder

$$S = A \oplus B \quad \text{and} \quad C = A\,B$$

∴ from 2 × 4 line decoder

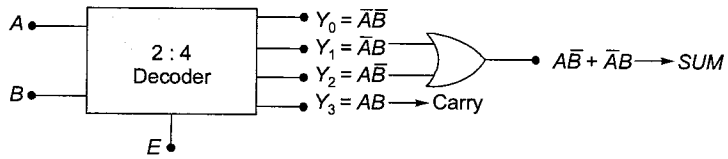



***Figure-4.29*** *Half Adder using 2 × 4 line decoder*

**The 3 × 8 Line Decoder**

- Figure 4.30 shows the circuitry for a decoder with three inputs $A$, $B$ and $C$ and eight outputs $Y_0$, $Y_1$.....$Y_7$. It uses all AND gates, and therefore the output are active HIGH. For active LOW outputs. NAND gates are used. The truth table is shown in Table 4.12.
- Truth Table:

| A | B | C | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ | $Y_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

***Table-4.12*** *Truth table for 3 × 8 line decoder*

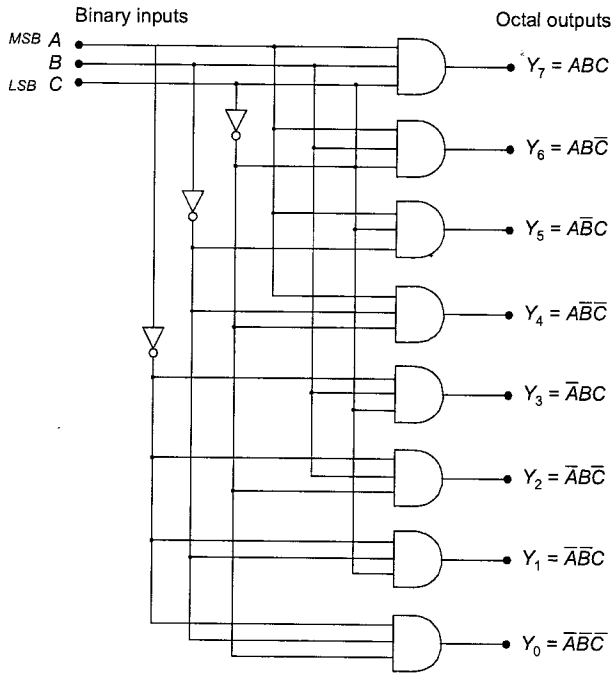



***Figure-4.30*** *Logic diagram of 3 × 8 line decoder*

- To implement a full adder or a full subtractor 3 × 8 decoder and two OR gates are required.
  *For example:* Implementation of full adder circuit using 3 × 8 line decoder:
  As we know, for full adder circuit

  $$\text{SUM} = A \oplus B \oplus C$$

  or $\Sigma m\,(1, 2, 4, 7)$

  and $$\text{CARRY} = AB + BC + CA$$

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

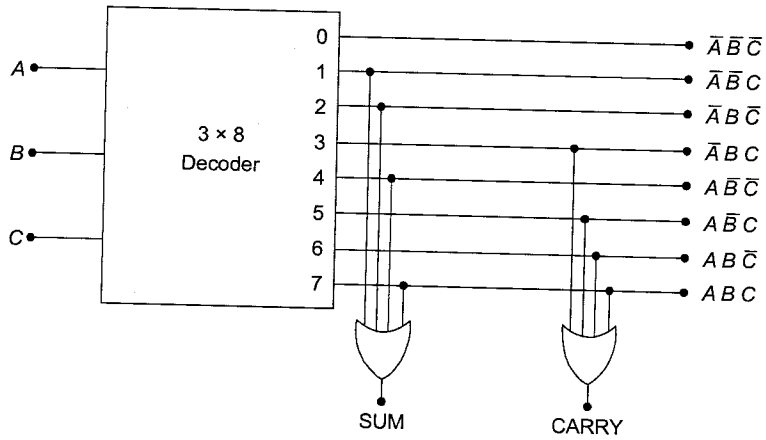



***Figure-4.31*** *Full adder using 3 × 8 line decoder*

- Implementation of 4 × 16 decoder using 3 × 8 decoder:

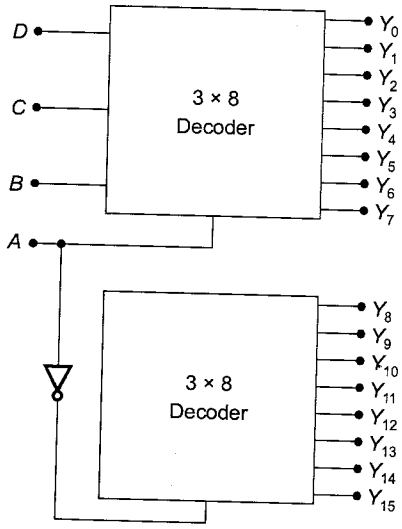



***Figure-4.32*** *Implementation of 4 × 16 decoder from 3 × 8 decoder*

- Implementation of Higher order decoder using lower order decoder

| Given Decoder | To be implemented | Required no. of decoders |
|---|---|---|
| (2 × 4) line | (4 × 16) line | 1 + 4 = 5 |
| (2 × 4) line | (3 × 8) line | 2 + 1 NOT gate |
| (4 × 16) line | (8 × 256) line | 1 + 16 = 17 |

**Example-4.19** Impliment $A+\bar{B}\bar{C}$ by using 3 × 8 line decoder.

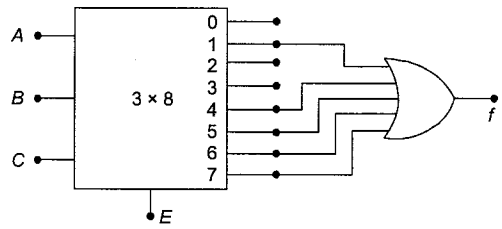
***Solution:***

Given, $f = A+\bar{B}\bar{C}$

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

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

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




**Example-4.20** Consider the following statements:

A 4 : 16 decoder can be constructed (with enable input) by:

1. using four 2:4 decoders (each with an enable input) only.
2. using five 2:4 decoders (each with an enable input) only.
3. using two 3:8 decoders (each with an enable input) only.
4. using two 3:8 decoders (each with an enable input) and an inverter.

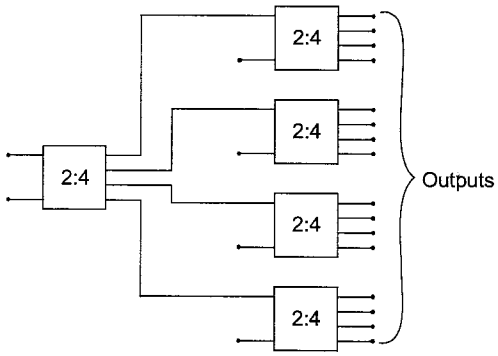
Which of the statements given above is/are correct

(a) 2 and 3 (b) 1 only

(c) 2 and 4 (d) None of the above

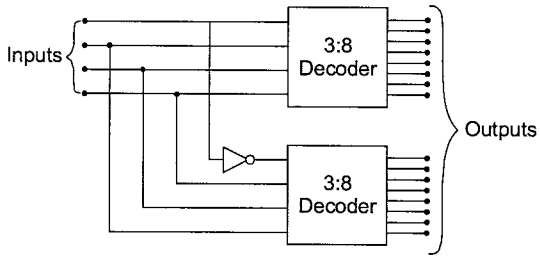
***Solution: (c)***

4 : 16 decoder using only 2 : 4 decoders:



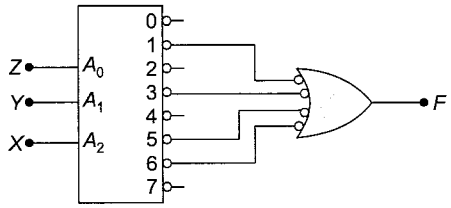

Five 2 × 4 decoders are required

4 : 16 decoder using two 3 : 8 decoders and an inverter:




Two 3 × 8 decoder and one inverter is required.

**Example - 4.21** A 3 to 8 line decoder, with active low outputs, is used to implement a 3 variable Boolean function as shown in the figure.




The simplified form of Boolean function $F(A,B,C)$ implemented in 'Product of Sum' form will be

(a) $(X+Z)\cdot(\bar{X}+\bar{Y}+\bar{Z})\cdot(Y+Z)$

(b) $(\bar{X}+\bar{Z})\cdot(X+Y+Z)\cdot(\bar{Y}+\bar{Z})$

(c) $(\bar{X}+\bar{Y}+Z)\cdot(\bar{X}+Y+Z)\cdot(X+\bar{Y}+Z)\cdot(X+Y+\bar{Z})\cdot(X+Y+\bar{Z})$

(d) $(\bar{X}+\bar{Y}+Z)\cdot(\bar{X}+Y+\bar{Z})\cdot(X+\bar{Y}+Z)\cdot(X+\bar{Y}+\bar{Z})$

***Solution: (a)***

Let us consider active high input

| X \ YZ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |

$$F = \Sigma(1, 3, 5, 6) = \Pi M(0, 2, 4, 7) = (Y+Z)\cdot(X+Z)\cdot(\bar{X}+\bar{Y}+\bar{Z})$$

**BCD to Seven Segment Display Decoder**

- One of the most popular and simplest methods for displaying numerical digits is a "7-segment display" to form the decimal characters '0' through '9' and sometimes the HEX-characters '*A*' through '*F*.
- It is used in electronic calculator, digital multimeters, watches (wrist and railways)
- It is used to take a 4-bit BCD input and provide the outputs that will pass current through the appropriate segments to display the decimal digit.
- Figure 4.33, shows a seven segment display consisting of seven light emitting segments. However the logic circuit of an 8-4-2-1 BCD-to-seven segment decoder is shown in Figure 4.34.

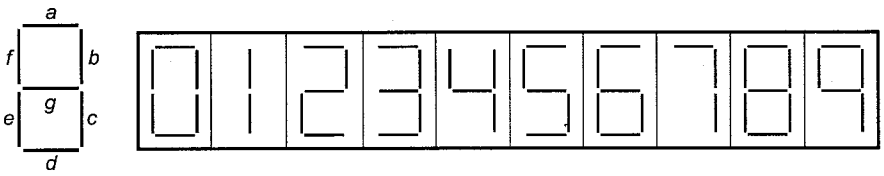



***Figure-4.33*** *Letters and Numerals used to designate the segments*

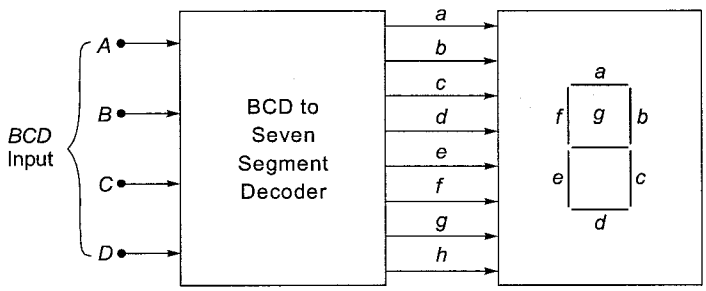



***Figure-4.34*** *Logic Circuit of BCD 7-segment Display Decoder*

- Truth Table:

**Table-4.13** *Truth table of BCD-to-7-segment decoder*

| Decimal digit displayed | Input | | | | Output | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *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 | 0 | 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 | 0 | 0 | 1 | 1 |

- Minimization of Boolean function (after using *K*-map):

$$a = \bar{B}\bar{D} + BD + CD + A; \qquad b = \bar{B} + \bar{C}\bar{D} + CD$$

$$c = B + \bar{C} + D = \overline{\bar{B}C\bar{D}}; \qquad d = \bar{B}\bar{D} + C\bar{D} + \bar{B}C + B\bar{C}D$$

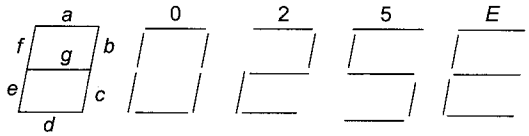
$$e = \bar{B}\bar{D} + C\bar{D}; \qquad f = A + \bar{C}\bar{D} + B\bar{C} + B\bar{D}$$

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

**NOTE:** The outputs of display has AND-OR logic, so it can be implemented by '24' NAND gates.

**Example-4.22** Two products are sold from a vending machine, which has two push buttons $P_1$ and $P_2$. When a button is pressed, the price of the corresponding product is displayed in a 7-segment display.

If no buttons are pressed, '0' is displayed, signifying '₹ 0'
If only $P_1$ is pressed, '2' is displayed, signifying '₹ 2'
If only $P_2$ is pressed, '5' is displayed, signifying '₹ 5'
If both $P_1$ and $P_2$ are pressed, '*E*' is displayed, signifying 'Error'
The names of the segments in the 7-segment display, and the glow of the display for '0', '2', '5' and '*E*' are shown below.




Consider:
(*i*) push button pressed/not pressed is equivalent to logic 1/0 respectively,
(*ii*) a segment glowing/not glowing in the display is equivalent to logic 1/0 respectively
If segments *a* to *g* are considered as functions of $P_1$ and $P_2$, then which of the following is correct?

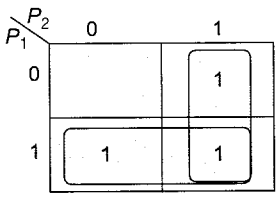
(a) $g = \bar{P}_1 + P_2, d = c + e$
(b) $g = P_1 + P_2, d = c + e$
(c) $g = \bar{P}_1 + P_2, e = b + c$
(d) $g = P_1 + P_2, e = b + c$

***Solution: (b)***

We have two input variables here as $P_1$ and $P_2$. Firstly we draw a truth table according to statement of problem.

| Inputs | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $P_1$ | $P_2$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | g | |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | → display '0' |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | → display '5' |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | → display '2' |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | → display 'E' |

K-map for 'g':




$\boxed{g = P_1 + P_2}$

also, we have

$a = 1,\ b = \overline{P}_2,\ c = \overline{P}_1,\ d = 1,\ e = P_1 + \overline{P}_2,\ f = \overline{P}_1 + P_2$

since, $e \neq b + c$

but, $d = c + e$

$= P_1 + \overline{P}_2 + \overline{P}_1 = 1 + \overline{P}_2 = 1$

so, $g = P_1 + P_2$ and $d = c + e$

### 4.3.5 Demultiplexer

- A demultiplexer is a combinational circuit that receives information on a single line and transmits this on one of "$2^n$" possible output lines. However the selection of a specific output line is controlled by the bit values of '$n$' selection lines.
- A 'decoder' with an enable input can function as a 'demultiplexer'.
- Demultiplexer is also known as:
  - (*i*) Data distributor
  - (*ii*) Serial to parallel converter
  - (*iii*) One to many circuit
- It is used to perform the reverse operation of MUX.

**NOTE:** DEMUX can not be used as an universal gate

- If '$n$' be the number of selection lines then,
  Total number of output = $m = 2^n$ or $n = \log_2 m$

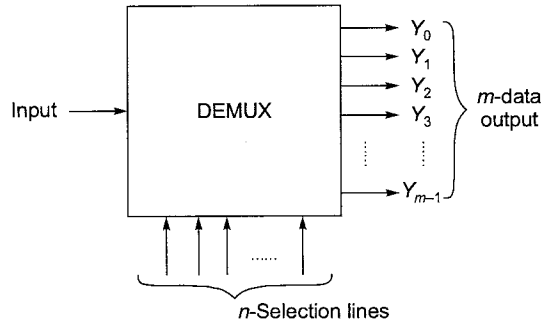
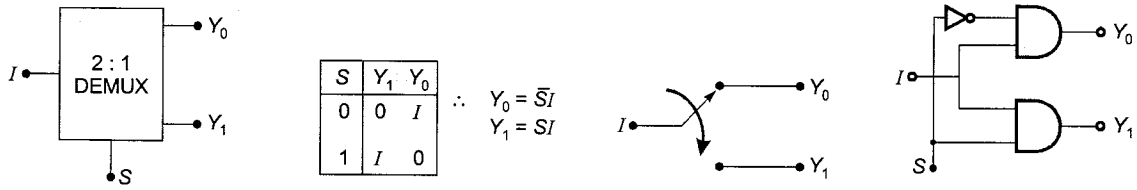



***Figure-4.35*** *Functional diagram of a digital DEMUX*

### The 1 to 2 Line Demultiplexer

Figure 4.36 shows the logic circuitry and function table for 1 : 2 line DEMUX. It has an input '$I$' for which output $Y_0$ and $Y_1$ are obtained. The selection line S is used to select the specific output.




(a) Block Diagram

| $S$ | $Y_1$ | $Y_0$ |
|---|---|---|
| 0 | 0 | $I$ |
| 1 | $I$ | 0 |

$\therefore$ $Y_0 = \bar{S}I$, $Y_1 = SI$

(b) Function Table

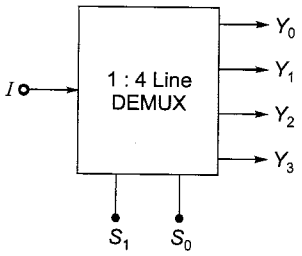
(c) Switching Diagram

(d) Logic Circuit

***Figure-4.36*** *1 : 2 Line Demultiplexer*

### The 1 to 4 Line Demultiplexer

A 1 : 4 line DEMUX is shown in Figure 4.37. The input data line goes to all of the AND gates. The two select lines $S_0$ and $S_1$ enable only one gate at a time and the data appearing on the input line will pass through the selected gate to the associated output line. The function table is shown in Figure 4.37(b).




(a) Block Diagram

| $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $E$ |
| 0 | 1 | 0 | 0 | $E$ | 0 |
| 1 | 0 | 0 | $E$ | 0 | 0 |
| 1 | 1 | $E$ | 0 | 0 | 0 |

(b) Function Table

where, $Y_0 = \bar{S}_1\bar{S}_0 I$; $Y_1 = \bar{S}_1 S_0 I$; $Y_2 = S_1\bar{S}_0 I$ and $Y_3 = S_1 S_0 I$

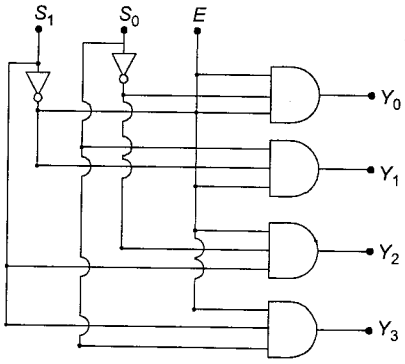
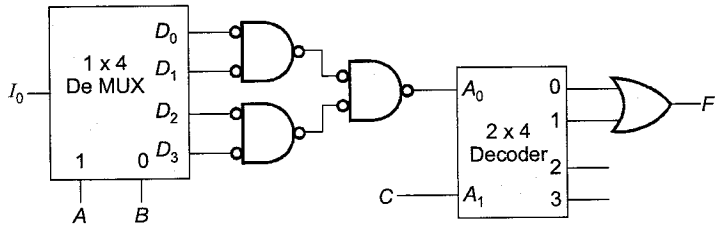



***Figure-4.37*** *1 : 4 Line Demultiplexer*

- Implementation of higher order DEMUX using lower order DEMUX

| Given DEMUX | To be implemented | Required |
|---|---|---|
| 1 : 2 | 1 : 4 | 1 + 2 = 3 |
| 1 : 2 | 1 : 8 | 1 + 2 + 4 = 7 |
| 1 : 2 | 1 : 16 | 1 + 2 + 4 + 8 = 15 |
| 1 : 2 | 1 : 64 | 1 + 2 + 4 + 8 + 16 + 32 = 63 |
| 1 : 4 | 1 : 16 | 1 + 4 = 5 |

**Example - 4.23** Consider the logic circuit given below:




The minimized expression for $F$ is

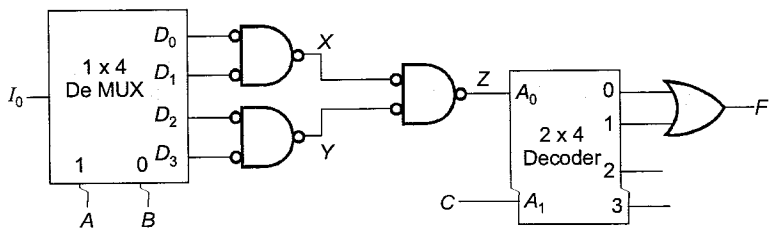
(a) $\bar{C}$

(b) $I_0$

(c) $C$

(d) $\bar{I}_0$

***Solution: (a)***



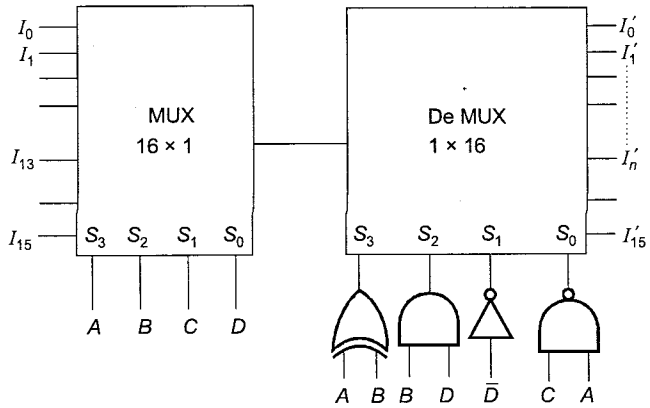

$$X = \overline{\bar{D}_0\bar{D}_1}\, I_0 = (D_0 + D_1)I_0 = (\bar{A}\bar{B} + \bar{A}B)I_0 = \bar{A}I_0$$

$$Y = \overline{\bar{D}_2\,\bar{D}_3}\, I_0 = (D_2 + D_3)I_0 = (A\bar{B} + AB)I_0 = AI_0$$

$$Z = (\overline{\bar{X}\cdot\bar{Y}}) = X + Y = \bar{A}\cdot I_0 + AI_0 = I_0$$

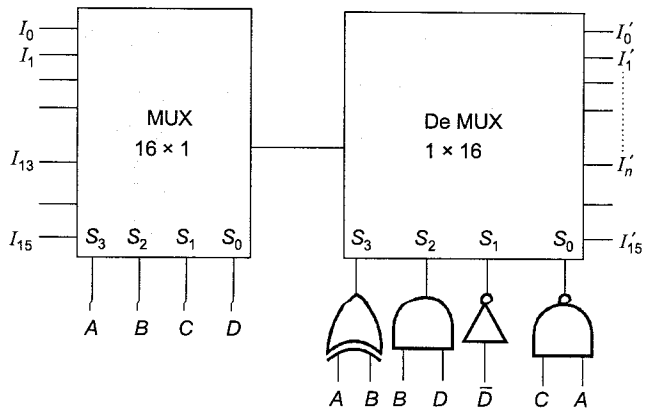
$$F = (\bar{Z}\bar{C} + Z\bar{C}) = \bar{C}(\bar{I}_0 + I_0) = \bar{C}$$

**Example - 4.24** Consider the logic circuit given below:




Input at line $I_{13}$ in 16 × 1 MUX corresponds to output at line $I'_n$ of 1 × 16 De MUX. The value of '$n$' is

***Solution:***




$$I_{13} \rightarrow A \quad B \quad C \quad D$$
$$1 \quad 1 \quad 0 \quad 1$$
$$A' = A \oplus B = 1 \oplus 1 = 0$$
$$B' = B \cdot D = 1 \cdot 1 = 1$$
$$C' = \overline{\overline{D}} = D = 1$$
$$D' = \overline{CA} = \overline{0.1} = 1$$
$$I_n \rightarrow A' \quad B' \quad C' \quad D'$$
$$0 \quad 1 \quad 1 \quad 1$$
$$(A' \; B' \; C' \; D') = 7 \Rightarrow n = 7$$

## 4.3.6 Encoders

- An encoder is a combinational circuit that performs the inverse operation of a decoder.
- It has '$2^n$' input lines, and '$n$' output lines. Out of $2^n$ input lines only one is activated at a given time and produces an $n$ bit output code, depending upon which input is activated. Like decoder, encoder also does not have any selection lines.
- The Figure 4.38 shows the general block diagram of an encoder with $m$ number of inputs and $n$-number of outputs.

  Where,

$$m = 2^n \text{ or } n = \log_2 m$$

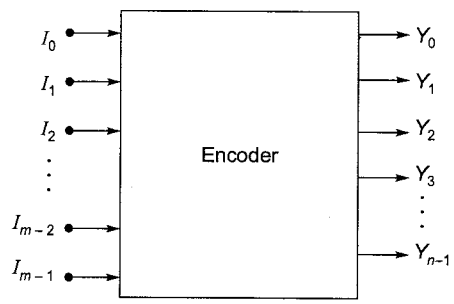



***Figure-4.38*** *Block diagram of an encoder*

- Encoder is used to convert other codes to binary such as,
  1. Octal to binary encoder (8 × 3 line)
  2. Decimal to BCD encoder (10 × 4 line)
  3. Hexadecimal to binary encoder (16 × 4 line)

### Octal to Binary Encoder

An octal to binary encoder (8 line to 3 line) has eight inputs and it produces three outputs corresponding to the activated input. Figure 4.39 shows the block diagram of octal to binary encoder and Table 4.14 shows the truth table for octal binary encoder.

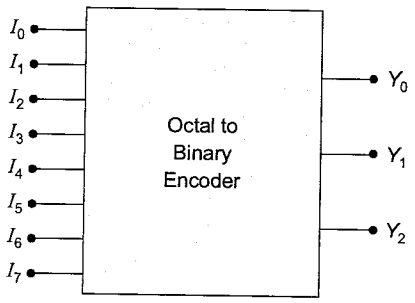



***Figure-4.39*** *Block diagram of octal to binary encoder*

| $I_7$ | $I_6$ | $I_5$ | $I_4$ | $I_3$ | $I_2$ | $I_1$ | $I_0$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

**Table-4.14** *Truth table for 8 line to 3 line encoder*

from the truth table, it is clear that $Y_2$ is '1' if any of the digits $I_4$, $I_5$ or $I_6$ or $I_7$ is a '1'.

Therefore, $Y_2 = I_4 + I_5 + I_6 + I_7$

Similarly, $Y_1 = I_2 + I_3 + I_6 + I_7$

and $Y_0 = I_1 + I_3 + I_5 + I_7$

As we see that $I_0$ is not present in any of the expression. So, $I_0$ is a don't care.

- From the logical expression a logic circuit for 8 line to 3 line encoder can be constructed as shown in Figure 4.40.

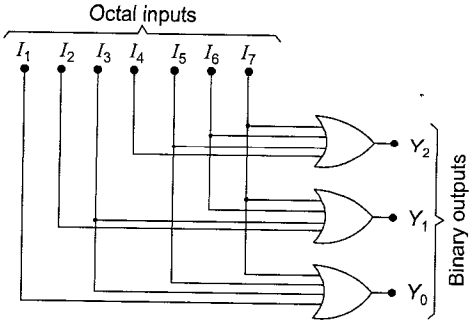



***Figure-4.40*** *Logic diagram of 8 line to 3 line encoder*

**NOTE:** The encoder contains OR gate.

### Decimal to BCD Encoder

This encoder has ten inputs (0 to 9), and four outputs corresponding to BCD codes. The Figure 4.41(a) shows a basic 10 line to 4 line encoder. However the truth table is shown in Table 4.15.

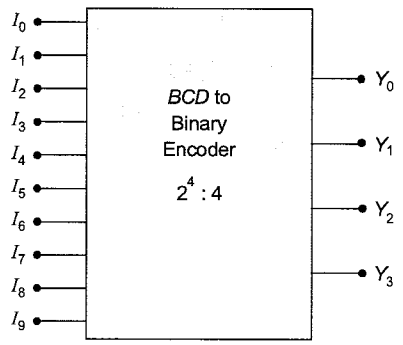



***Figure-4.41(a)*** *Block diagram of Decimal to BCD encoder*

| $I_9$ | $I_8$ | $I_7$ | $I_6$ | $I_5$ | $I_4$ | $I_3$ | $I_2$ | $I_1$ | $I_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 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 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

**Table-4.15** *Truth Table decimal to BCD encoder*

from the truth table we can determine the expression for outputs as.

$$Y_3 = I_8 + I_9; \qquad Y_2 = I_4 + I_5 + I_6 + I_7$$
$$Y_1 = I_2 + I_3 + I_6 + I_7; \qquad Y_0 = I_1 + I_3 + I_5 + I_7 + I_9$$

It is clear that, there is no explicit input for a decimal '0'. The BCD outputs are 0000 when the BCD inputs 1 – 9 are all '0'.

Based on the logical expression, the logic circuit for Decimal to BCD encoder can be constructed as shown in Figure 4.41(b).

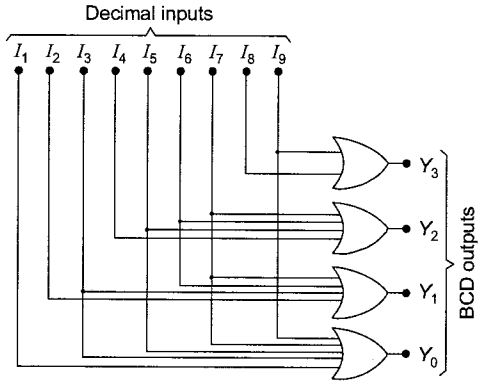



***Figure-4.41(b)*** *Logic diagram for decimal to BCD encoder*

### 4.3.7 Priority Encoder

- The encoders discussed above will operate correctly, provided that one and only one decimal input is 'high' at any given time. However in some practical systems, two or more decimal inputs may inadversely become 'high' at the same time. Thus, to eliminate this drawback (when more than one input is activated at a time) a modified version of simple encoder is used called "priority encoder".
- A priority encoder is a logic circuit that includes the priority functions to ensure that when two or more inputs are equal to '1' or high at the same time, the input having the highest priority will take precedence.

**Four Input Priority Encoder**

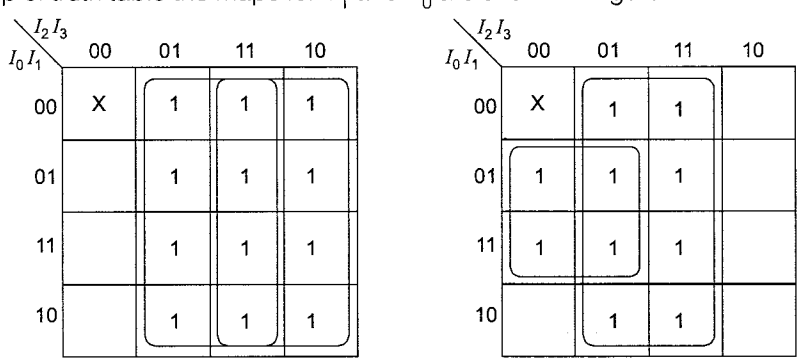
The truth table of a 4-bit priority encoder is given in Table 4.16. The inputs are designated as $I_0$, $I_1$, $I_2$, and $I_3$ and in addition to the two outputs $Y_1$ and $Y_2$ the circuit has a third output designated as $X$; this is activated when one or more inputs are 'high'. If all the inputs are '0' or 'low' there are no valid input and $X = 0$. The other two outputs are not inspected when $X = 0$ and are specified as don't care condition.

**Table-4.16** *Truth table for four input priority encoder*

| $I_0$ | $I_1$ | $I_2$ | $I_3$ | $Y_1$ | $Y_0$ | $X$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | X | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| X | 1 | 0 | 0 | 0 | 1 | 1 |
| X | X | 1 | 0 | 1 | 0 | 1 |
| X | X | X | 1 | 1 | 1 | 1 |

According to the truth table shown in Table 4.16 the higher the subscript numbers, the higher the priority of the input. Input $I_3$ has the highest priority. So regardless of the values of other inputs, when the input is high the output for $Y_1$ $Y_0$ = 11. Now $I_2$ has the next priority thus, when $I_3 = 0$ and $I_2 = 1$ regardless of other lower priority inputs shown as don't care, the outputs $Y_1$ $Y_0$ = 10 similarly for $I_1$ it is required that $I_2$ and $I_3$ should be 'low' and $I_0$ is don't care and outputs $Y_1$ $Y_0$ = 01 and lastly for $I_0$ (lowest priority) all the other inputs are 'low' and outputs $Y_1$ $Y_0$ = 00.

With the help of truth table the maps for $Y_1$ and $Y_0$ are shown in Figure.




(a) $Y_1 = I_2 + I_3$ (b) $Y_0 = I_0 + I_1\overline{I_2}$

***Figure-4.42*** Maps for a priority encoder

The simplified expressions for $Y_1$ and $Y_0$ is

$$Y_1 = I_2 + I_3$$

$$Y_0 = I_0 + I_1\overline{I_2} \quad \text{and} \quad X = I_0 + I_1 + I_2 + I_3$$

The implementation of logic circuit is shown in Figure 4.43.

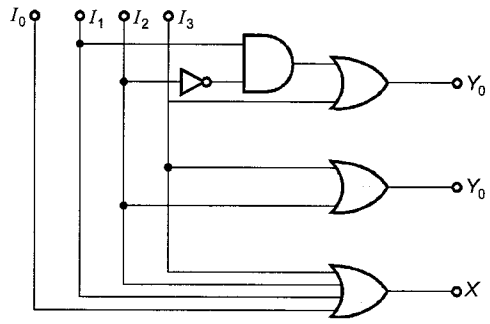



***Figure-4.43*** *Logic Diagram of Four Input Priority Encoder*

### 4.3.8 Parity Bit Generators/Checkers

- For error detection and correction codes, Exclusive-OR functions are very useful in system. Binary data, when transmitted and processed, is susceptible to noise that can alter its 1's to 0's and 0's to 1's. To detecting such errors during transmission of binary information, an additional bit called 'partity bit' is added to the data bits. Thus, a parity bit is an extra bit included with a binary message to make the number of 1's either even or odd. 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 parity generator. The circuit that checks the parity in the receiver is called a parity checker.
- In order to check or generate the proper parity bit in a given code word, the basic principle used is, "the module sum of an even number of 1's is always a '0' and the modulo sum of an odd number of 1's is always a 1". Therefore in order to check for an error, all the bits in the received word are added. If the modulo sum is a '0' for an odd parity system or a '1' for every parity system an error is detected.
- The logic diagram for an even parity and odd parity generator is shown in Figure 4.44(a) and (b) respectively.

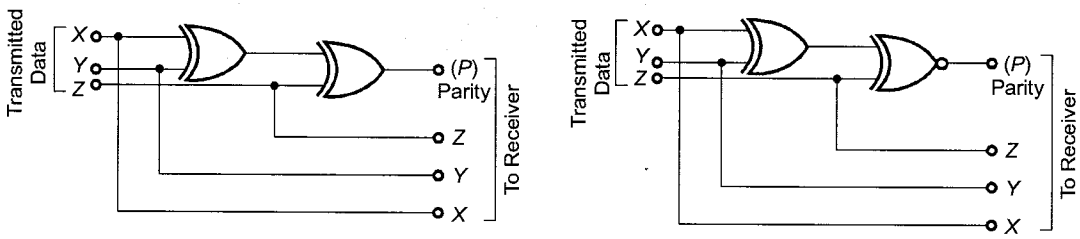



***Figure-4.44 (a)*** *Even parity generator* ***(b)*** *Odd parity generator*

- The transmitted data, together with the parity bit are applied to the parity checker circuit at the receiver to check for the possible errors in the transmission as shown in Figure 4.45 (a) and (b) respectively.

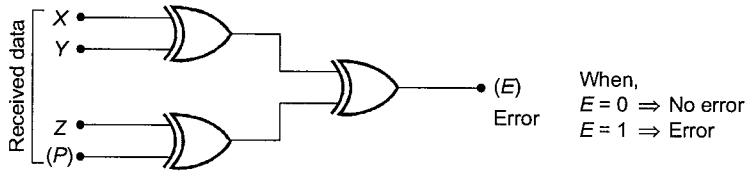



***Figure-4.45 (a)*** *Even parity checker*

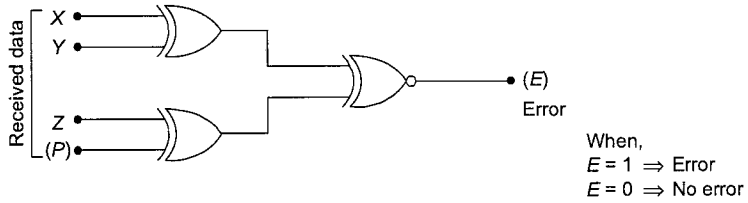



***Figure-4.45 (b)*** *Odd parity checker*

**NOTE**

If the input ($P$) in Figure 4.45 (a) is connected to logic '0' and the output is marked with $P$, then the same circuit can be used for both even parity generation as well as even parity checking, because $Z \oplus 0 = Z$, causing the value of $Z$ to pass through the gate unchanged.

## 4.4 Hazards

- **Hazard** is a phenomenon that may cause the digital circuit to malfunction. These are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays. Such a transient is also called a 'glitch' or a false 'spike', which is caused by hazardous behaviour of the logic circuit.
- Hazards occurs in combinational as well as sequential circuits. In combinational circuits they may cause a temporary false output value, where as in sequential circuits they may result in a transition to a wrong stable state. It is therefore necessary to check for possible hazards, determine whether they can cause improper operation so that step can be taken to eliminate their effect.
- **Hazards are classified as:**
  (*i*) Static hazard (*ii*) Dynamic hazard (*iii*) Essential hazard

**Static Hazard**

Static hazard occur in combinational circuits and can be eliminated by using redundant gates. Static hazard is further classified as:

1. Static -1 hazard 2. Static -0 hazard

The circuit in Figure 4.46 (a) shows the occurrence of hazard. Assume that all the three inputs initially equal to '1'. Now consider a change in $B$ from '1' to '0'. Then the output of gate-1 changes to '0' and that of gate-2 changes to '1', leaving the output $Y$ = '1' regardless of the changing variable, the output may momentarilty go to '0' if the propagation delay of the inverter is taken into account. The delay in the inverter may cause the output of gate-1 to change to '0' before the output of gate-2 change to 1. In that case, both input of the gate-3 are momentarily equals to '0', causing the output $Y$ to go to '0', for a short interval. Such a condition where the output is expected to be at '1', regardless of the changing variable, the false '0' level for a short interval will cause static-1 hazard.

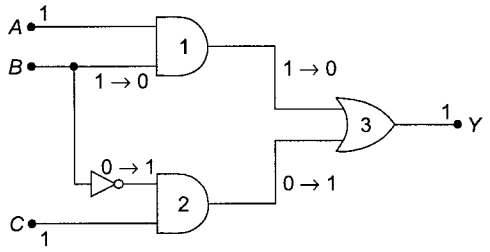



***Figure-4.46(a)*** *AND-OR circuit containing static hazard*

If the output is expected to be '0' regardless of the changing variable, the false '1' level for the short interval is called a static '0' hazard. The circuit shown in Figure 4.46 (b) is a NAND implementation. Here gate-1 and 2 are NAND gates and their outputs are the complement of the outputs of corresponding AND gates in Figure 4.46 (a). When *B* changes from '1' to '0', both inputs of gate-3 may be equal to '1', causing the output to produce a momentary change to '0', when it should have stayed to '1'.

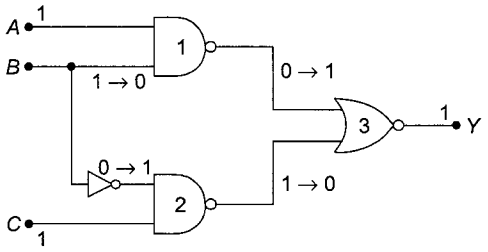



***Figure-4.46 (b)*** *NAND circuit containing static Hazard*

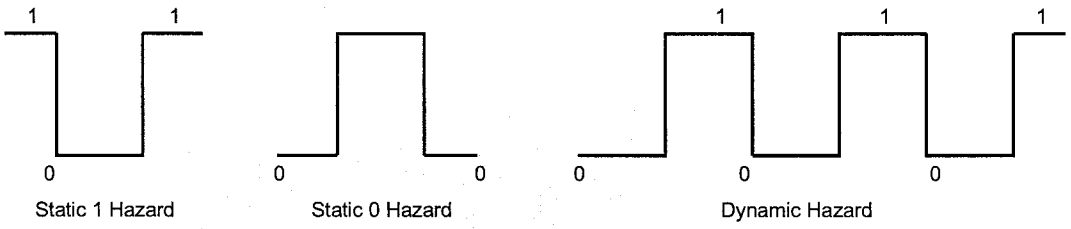



***Figure-4.46 (c)*** *Types of Hazards*

**Dynamic Hazard**

When the output changes several times then it should change from 1 → 0 or 0 → 1 only once, it is called dynamic hazard.

Dynamic hazard occur when the output changes for two adjacent input combinations while changing, the output should change only once; but it may change three or more times in short intervals because of different delays in several paths. Dynamic hazards occur only in multilevel circuit.

Figure 4.46 (c) illustrates three types of hazards. When a circuit is implemented in SOP form with AND-OR gates or with NAND gates, the removal of static-1 hazard guarantees that no static-0 hazard OR dynamic hazards will occur.

**Essential Hazard**

Essential hazard is the another type of hazard occur in asynchronous sequential circuit. It is called by unequal delays along two or more paths, that originated from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause such a hazard. Essential hazard cannot be corrected by adding redundant gates as in static hazards. However the problem that they can be corrected by adjusting the amount of delay in the effected path.

## Summary

- BCD code is used in calculators, counters, digital voltmeters, digital clocks etc.
- Minimum number of NAND/NOR gate required for the implementation of H.A/H.S is '5'.
- Minimum number of 2 × 1MUX required for the implementation of a H.A/H.S is '3'.
- Minimum number of decoders required for the implementation of H.A/H.S is a (2 : 4 line) plus an OR gate.
- Half Adder + Inverter = Half Subtractor
- Full Adder = 2 Half Adder + OR gate
- Full subtractor = 2 Half subtractor + OR gate
- Minimum number of NAND/NOR gates required for the implementation of F.A./F.S. is '9'.
- Number of AND gates required for an $N$ bit look ahead carry circuit is $\frac{N(N+1)}{2}$, and number of OR gates required for look ahead carry circuit is $N$.
- A look ahead carry adder is the fastest adder.
- To implement $2^n$ : 1 MUX by using 2 : 1 MUX, the total number 2 : 1 MUX required is $(2^n - 1)$.
- A 'decoder' with an enable input can function as a DEMUX.
- DEMUX contains AND gates.
- 2 × 4 decoder may act like a 1 : 4 DEMUX and vice-versa.
- Decoder and DEMUX circuits are almost same.
- Decoder contains AND gates or NAND gates.
- In priority encoder more than one input line can be high but only for the highest priority input line, binary output is available at output.
- Encoder contains OR gates
- Relation between the parity (check) bits and information bits.

  $$2^C - 1 \geq C + I$$

  where, $C$ = Number of check bits

  $I$ = Information bits

  $C + I$ = Word length
- Hamming codes are used for error detection and correction for a single bit only.
- If the minimum Hamming code distance is $m$, then the number of errors to be corrected is less than $m/2$.
- The type of display used in calculators is a 7 segment LED/LCD display.
- Static-1 hazard with contact networks is called a tie set hazard.
- Static-0 hazard with contact networks is called a cut set hazard.

## Student's Assignments 1

**Q.1** Binary message bits are coming serially. They are to be taken in blocks of four binary digits $a_3$ $a_2$ $a_1$ $a_0$ and coded into 7-bit binary word as $a_3$ $a_2$ $a_1$ $a_0$ $c_2$ $c_1$ $c_0$, where $c_2$, $c_1$ and $c_0$ satisfy the following equations, $a_3 + a_2 + a_1 + c_2 = 0$; $a_3 + a_2 + a_0 + c_1 = 0$; $a_3 + a_1 + a_0 + c_0 = 0$

(*i*) Draw the complete logic circuit for generating $c_2$, $c_1$ and $c_0$ giving the details of S/P converter and parity generator circuits. Explain the working of the circuit.

(*ii*) If an additional parity bit $C_p$ is to be generated making total number of bits in a block (part a) equal to eight so that number of 1's in the block are even, give the logic circuit for generation of $C_p$ using minimum number of Ex-ORs.

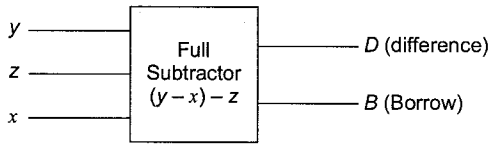
**Q.2** Implement the function
$f(A, B, C, D) = \Sigma(0, 1, 5, 7, 10, 14, 15)$
using an appropriate multiplexer.

**Q.3** Write the truth table for the following in Boolean expression, and realize it using only an 8 : 1 multiplexer. $f = \overline{AB} + BD + AB\overline{C}$

**Q.4** Design a logic circuit to generate (a) an even parity bit (b) an odd parity bit for a 3 bit binary input.

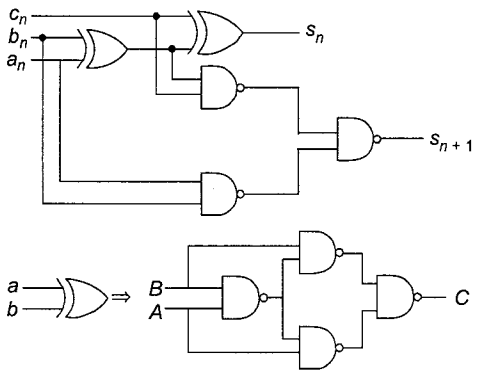
## Student's Assignments 2

**Q.1** Find the Boolean expression 'B' for the digital circuit below




(a) $xy + yz + zx$ (b) $\bar{x}y + yz + z\bar{x}$
(c) $x\bar{y} + \bar{y}z + zx$ (d) $xy + y\bar{z} + \bar{z}x$

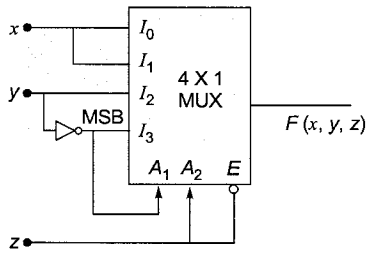
**Q.2** A full adder circuit is shown below:




The propagation time for each NAND GATE is 15 nsec then the time required for each addition is given by

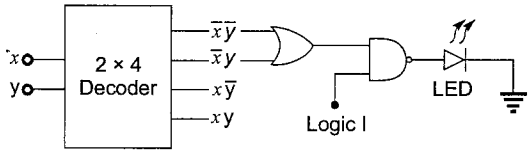
(a) 45 nsec (b) 90 nsec
(c) 125 nsec (d) 180 nsec

**Q.3** For the 4 × 1 MUX shown below the Boolean expression $F(x, y, z)$ is




(a) $xy\bar{z}$ (b) $(xy\bar{z} + xyz + z\bar{y})$
(c) $xy\bar{z} + xyz$ (d) None of these

**Q.4** A LED is connected to a combinational circuit as shown below:

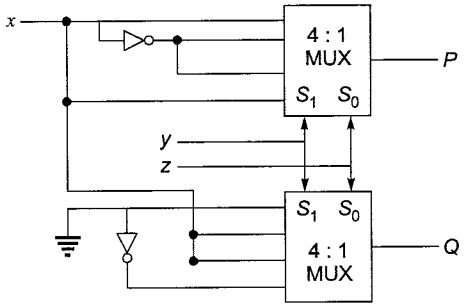



A LED emits light when its positive terminal is connected to a higher potential (equivalent to logic 1) and its negative terminal is connected to a lower potential (equivalent to logic 0).

For what values of input x and y, the LED will emit light?

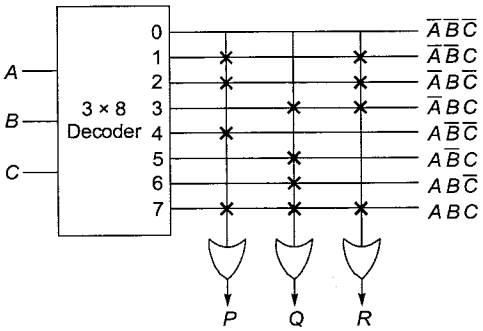
(a) $x = 1$ and $y = 0$
(b) $x = 0$ and $y = 1$
(c) $x = 1$ and independent of the value of $y$
(d) $y = 1$ and independent of the value of $x$

**Q.5** The multiplexer circuit functions as



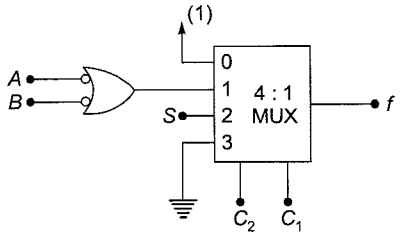

(a) Full adder
(b) Full subtractor
(c) Two output comparator
(d) Non arithmetic combinational logic

**Q.6** Consider the 8 × 3 ROM shown below:




Which of the following statement is correct?
(a) The ROM can be used as full subtractor with $R$ as the Difference and $P$ as the Borrow.
(b) The ROM can be used as a full subtractor with $P$ as the Difference and $Q$ as the Borrow.
(c) The ROM can be used as a full adder with R as the Sum and $Q$ as the Carry.
(d) The ROM can be used as full adder with $P$ as the Sum and $Q$ as the Carry.

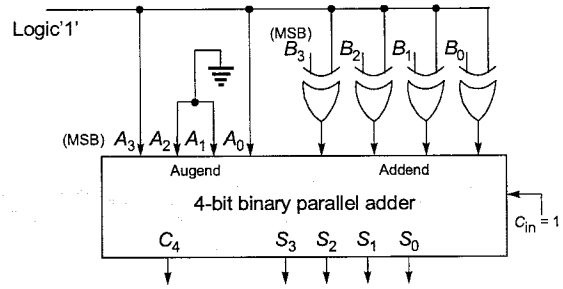
**Q.7** Consider the following circuit:




Which one of the following gives the function implemented by the MUX-based digital circuit?

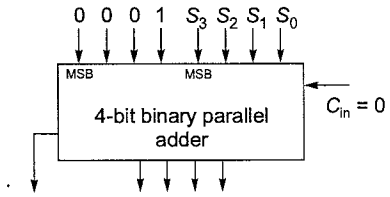
(a) $f = C_2 \cdot \overline{C}_1 \cdot S + \overline{C}_2 \cdot C_1 \cdot (\overline{A} + \overline{B})$
(b) $f = \overline{C}_2 \cdot \overline{C}_1 + C_2 \cdot C_1 + C_2 \cdot \overline{C}_1 \cdot S + \overline{C}_2 \cdot C_1 \cdot \overline{AB}$
(c) $f = \overline{AB} + S$
(d) $f = \overline{C}_2 \cdot \overline{C}_1 + C_2 \cdot \overline{C}_1 \cdot S + \overline{C}_2 \cdot C_1 \cdot \overline{AB}$

**Q.8** Consider the digital circuit shown below. A single digit decimal number ($B$) is converted into its 4 bit binary equlivalent ($B_3 B_2 B_1 B_0$) and then applied to the addend bits of the adder as shown below:




(*i*). The output $S_3\, S_2\, S_1\, S_0$ is the
(a) 8's complement of $B$
(b) 9's complement of $B$
(c) 10's complement of $B$
(d) 11's complement of $B$

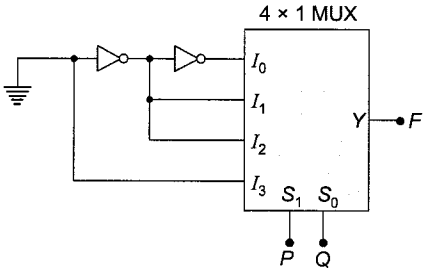
(*ii*). If $C_4 = 1$ and the outputs $S_3\, S_2\, S_1\, S_0$ are given to the addend bits of the 4-bit binary parallel adder as shown below:




then output of the circuit is
(a) 9's complement of $B$
(b) 10's complement of $B$
(c) 11's complement of $B$
(d) 12's complement of $B$

**Q.9** The number of select lines required in a single input and '256' output DEMUX is
(a) 8  (b) 16
(c) 32  (d) 64

**Q.10** The logic function implemented by the circuit below is (ground implies a logic "0")




(a) $F = \text{AND}(P, Q)$ (b) $F = \text{OR}(P, Q)$
(c) $F = \text{XNOR}(P, Q)$ (d) $F = \text{XOR}(P, Q)$

**Q.11** We consider the addition of two 2's complement numbers $b_{n-1}\, b_{n-2} \ldots b_0$ and $a_{n-1}\, a_{n-2} \ldots a_0$. A binary adder for adding unsigned binary numbers is used to add the two numbers. The sum is denoted by $c_{n-1}\, c_{n-2} \ldots c_0$ and the carry out by $c_{out}$, which one of the following options correctly identifies the overflow condition?

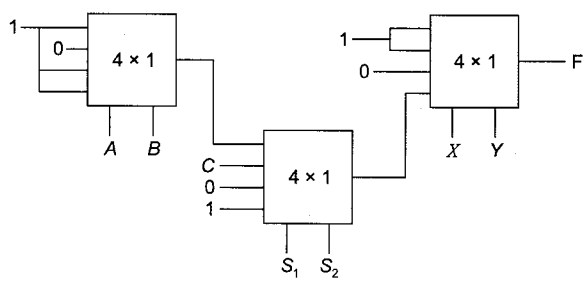
(a) $c_{out}\left(\overline{a_{n-1} \oplus b_{n-1}}\right)$

(b) $a_{n-1} b_{n-1} \overline{c_{n-1}} + \overline{a_{n-1}}\ \overline{b_{n-1}}\ c_{n-1}$

(c) $c_{out} \oplus c_{n-1}$

(d) $a_{n-1} \oplus b_{n-1} \oplus c_{n-1}$

**Q.12** What is the output of multiplexer shown below:




(a) $\overline{X}\,\overline{Y} + \overline{X}Y + XY S_1 S_2\, A\overline{B} + XY S_1 S_2 A\overline{B}$
$+ XY\, S_1 \overline{S_2}\, AB$

(b) $\overline{X}\,\overline{Y} + XY\overline{S_1}\,\overline{S_2}\,\overline{A}\,\overline{B} + XY\overline{S_1}\,\overline{S_2} A\overline{B}$
$+ XY + \overline{S_1}\,\overline{S_2} AB + \overline{S_1} S_2 C + XY S_1 S_2 + \overline{X}Y$

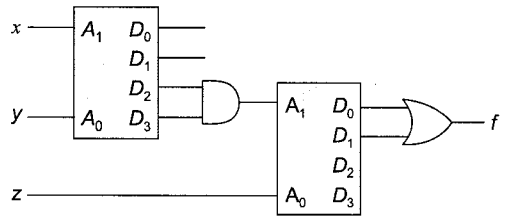
(c) $\overline{X}\,\overline{Y} + XY\overline{S_1} S_2 C + XYS_1\overline{S_2} A\overline{B} + XY\overline{S_1}$

(d) $S_1\overline{S_2} + A\overline{B}\,\overline{C} + \overline{X}\,\overline{Y} + XY\overline{S_1} S_2 C$

**Q.13** If carry propogation delay is 5Δ in full adder, then multiplication of 8 bit numbers using array co-multiplier takes (Assume AND gate delay = 2Δ)

(a) 72 Δ (b) 74 Δ
(c) 70 Δ (d) 82 Δ

**Q.14** A logic circuit consists of two 2 × 4 decoder as shown




The output of the decoder are as follows
$D_0 = 1$ when $A_0 = 0, A_1 = 0$
$D_1 = 1$ when $A_0 = 1, A_1 = 0$
$D_2 = 1$ when $A_0 = 0, A_1 = 1$
$D_3 = 1$ when $A_0 = 1, A_1 = 1$
The value of $f(x, y, z)$ is

(a) 0 (b) $Z$
(c) $\overline{Z}$ (d) 1

***Answer Key:***

| | | | | |
|---|---|---|---|---|
| **1.** (c) | **2.** (b) | **3.** (a) | **4.** (c) | **5.** (a) |
| **6.** (d) | **7.** (d) | **8.** ***(i)*** (b) & ***(ii)*** (b) | | **9.** (a) |
| **10.** (d) | **11.** (b) | **12.** (b) | **13.** (a) | **14.** (d) |

CHAPTER 05

# Sequential Logic Circuits

## Introduction

All the digital systems designed and analyzed so far are based upon combinational logic where the output levels at any instant of time depend only upon the levels present at the inputs at that time. Such systems are said to be memoryless systems. However there are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received, which cannot be accomplished using combinational circuits. These applications require outputs to be generated, they are not only dependent on the level present at the inputs at that time, but also on the state of circuit. i.e. on the prior input level conditions. The past history is provided by feedback from the output back to the input. Thus, sequential circuits are made up of combinational circuits and memory elements.

*Figure (5.1)* shows the block diagram of a sequential circuit. It consists of a combinational circuit, which accepts digital signals from external inputs and from the outputs of memory elements connected in feedback path and generates signals for external outputs and for the inputs to memory element. The *Table (5.1)* the comparison between combinational and sequential circuit.

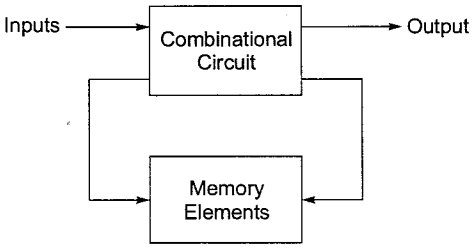



***Figure-5.1*** *: Block diagram of a sequential circuit*

**Table-5.1** *Comparison of combinational and sequential circuit*

| Combinational Circuits | Sequential Circuits |
|---|---|
| 1. The output variables at any instant of time depends only on the present input variables. | 1. The output variables at any instant of time depends not only on the present input variables but also on the past history of the system. |
| 2. These circuits do not require any memory element, hence called memory less system. | 2. To store the past history of the input variables, memory unit is required. |
| 3. Combinational circuits are faster. | 3. Sequential circuits are slower. |
| 4. They are easy to design. | 4. They are comparatively harder to design |

- The sequential circuits are classified as synchronous sequential circuits and asynchronous sequential circuits depending on the timing of their signals.

(*i*) **Synchronous Sequential Circuit**

- The change in input signals can affect memory element upon activation of clock signals.
- The maximum operational speed of clock depends on time delays involved.
- In this circuit, memory elements are "clocked flip-flops".
- It is easier to design.
- It is generally "edge triggered".

(*ii*) **Asynchronous Sequential Circuit**

- The change in input signals can affect memory element at any instant of time.
- Because of absence of clock, this circuit can operate faster than synchronous circuit.
- In this circuit, memory elements are either "unclocked flip-flops" or time delay elements.
- More difficult to design.
- It is generally "Level triggered".

**NOTE:** Clock is a periodically recurring pulse. It is generated by a pulse generator.

## 5.1 Latches and Flip-Flops

- A basic memory cell is a circuit that stores one bit of information. This one bit memory element is called flip-flop. The flip-flop is made up of an assembly of logic gates. Even though a logic gate by itself has no storage capacity, several logic gates can be connected together in ways that permit information to be stored. Flip-flops are the basic building blocks of most sequential circuit.
- A flip-flop, known more formally as a bistable multivibrator, has two stable states. It can remain in either of the states indefinitely, and the state can be changed by applying proper triggering signal.
- Flip flop is also called a binary or one-bit memory.
- *Figure (5.2)* shows the general symbol used for a flip-flop. The flip-flop has two outputs, labelled as $Q$ and $\bar{Q}$. The $Q$ is the normal output of the flip-flop and $\bar{Q}$ is the inverted output.

### 5.3.2 Conversion of S-R Flip-Flop into T Flip-Flop

- *Figure 5.16 (a)* shows the characteristic table of T flip-flop and the excitation table of S-R flip-flop.

| $T$ | $Q_n$ | $Q_{n+1}$ | $S$ | $R$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | x |
| 0 | 1 | 1 | x | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

***Figure-5.16 (a)***

- Using K-map, we get the expression of $S$ & $R$ in terms of $T$

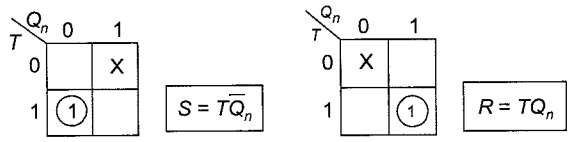



***Figure-5.16 (b)***

- Therefore, the circuit diagram for conversion of S-R flip-flop into $T$ flip-flop is

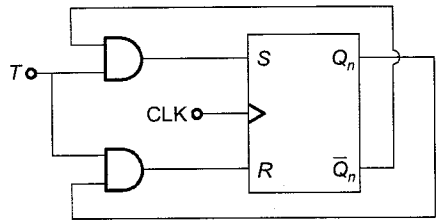



***Figure-5.16 (c)***

### 5.3.3 Conversion of J-K Flip-Flop into D Flip-Flop

- Here the available flip-flop is J-K FF and the desired flip-flop is $D$ FF. The *Figure 5.17(a)* shows the characteristic table of $D$ flip-flop with excitation table of J-K flip-flop.

| $D$ | $Q_n$ | $Q_{n+1}$ | $J$ | $K$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | x |
| 0 | 1 | 0 | x | 1 |
| 1 | 0 | 1 | 1 | x |
| 1 | 1 | 1 | x | 0 |

***Figure-5.17 (a)***

- The K-map for $J$ and $K$ is shown in *Figure 5.17 (b)*.

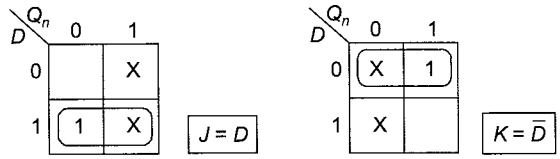



***Figure-5.17 (b)***

- The circuit diagram of the conversion of J-K flip-flop into $D$ flip-flop is shown in *Figure 5.17 (c)*.

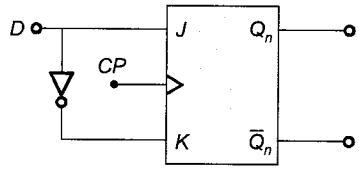



***Figure-5.17 (c)***

- Similarly the conversion of S-R flip-flop to *D* flip-flop is obtained with

$$\boxed{S = D} \quad \text{and} \quad \boxed{R = \overline{D}}$$

### 5.3.4 Conversion of J-K Flip Flop into T Flip Flop

- The *Figure 5.18 (a)* shows the characteristic table of *T* flip-flop (desired) and the excitation table of J-K flip-flop (available).

| $T$ | $Q_n$ | $Q_{n+1}$ | $J$ | $K$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | x |
| 0 | 1 | 1 | x | 0 |
| 1 | 0 | 1 | 1 | x |
| 1 | 1 | 0 | x | 1 |

***Figure-5.18 (a)***

- The K-map for *J* and *K* is shown in *Figure 5.18 (b)*.

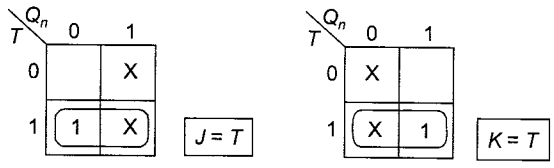



***Figure-5.18 (b)***

- The circuit diagram for the conversion of J-K flip-flop to *T* flip-flop is shown in *Figure 5.18 (c)*.

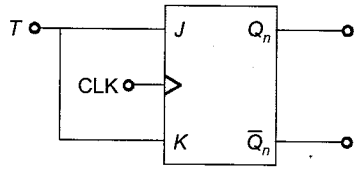



***Figure-5.18 (c)***

- Similarly D flip-flop can be converted into *T* flip-flop as: $\boxed{D = T \oplus Q_n}$

## 5.4 Applications of Flip-Flops

There are large applications of flip-flops some of the basic application are:

(*i*) Serial and parallel data storage
(*ii*) Data transfer
(*iii*) Serial to parallel converter
(*iv*) Parallel to serial converter
(*v*) Latch
(*vi*) Counters
(*vii*) As bounce elimination switch
(*viii*) Frequency division

**Example-5.4** An $X$-$Y$ flip flop, whose Characteristic Table is given below is to be implemented using a J-K flip flop

| $X$ | $Y$ | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | $Q_n$ |
| 1 | 0 | $\overline{Q}_n$ |
| 1 | 1 | 0 |

This can be done by making

(a) $J = X, K = \overline{Y}$  (b) $J = \overline{X}, K = Y$

(c) $J = Y, K = \overline{X}$  (d) $J = \overline{Y}, K = X$

***Solution: (d)***

$X$-$Y$ Truth Table

| X | Y | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | $Q_n$ |
| 1 | 0 | $\overline{Q}_n$ |
| 1 | 1 | 0 |

$J$-$K$ Truth Table

| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\overline{Q}_n$ |

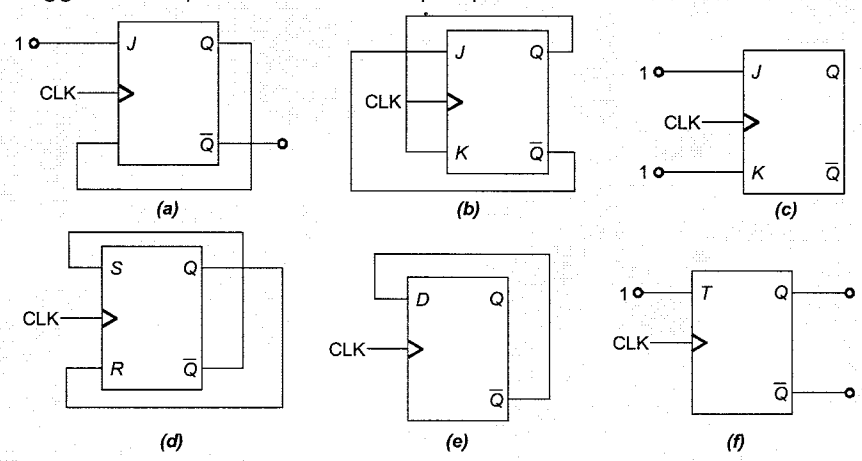
Excitation table

| Q(t) | Q(t+1) | J | K | X | Y |
|---|---|---|---|---|---|
| 0 | 0 | 0 | x | x | 1 |
| 0 | 1 | 1 | x | x | 0 |
| 1 | 0 | x | 1 | 1 | x |
| 1 | 1 | x | 0 | 0 | x |

To make ($X - Y$) FF using ($J - K$) FF, ($J$) should be ($\overline{Y}$) and ($K$) should be ($X$).

## Summary

- "Reset" and "Clear" are one and the same. It is used for setting the contents of FF or memory to ZERO.
- "Preset" is the state of a FF when $Q = 1$ and $\overline{Q} = 0$.
- "Set" is the state of a FF when $Q = 1$.
- Race-around condition occurs in J-K-FF to store 1-bit of information.
- Race-around condition always arises in "Asynchronous circuits".
- A Master-slave FF consists of an S-R-FF followed by a T-FF.
- The frequency is always halved at the output of any FFs whose behaviour is same as TOGGLED-FF.
- The time required to change the voltage level from 10% to 90% is known as rise time ($\tau_r$) where as the time required to change the voltage level from 90% to 10% is known as fall time ($\tau_f$).
- The pulse width on a clock is always measured at its 50% voltage level.
- The phase shift between rectangular clock waveform is referred as 'skew' and the time delay between the two clock pulse is called 'clock skew'.
- The toggle mode operation of various flip-flops are:



## Student's Assignments 1

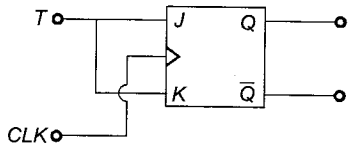
**Q.1** What do you mean by Latch? How it is different from flip-flops? Make the logical diagram of S-R Latch by using NAND and NOR gates with appropriate Truth Table.

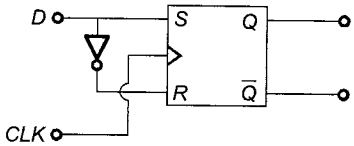
**Q.2** An *AB* flip-flop has four operations as:
Clear to '0', No change, Compliment and Set to '1', when the inputs *A* and *B* are 00, 01, 10 and 11 respetively.
(*i*) Tabulate the characteristic table and drive the characteristic equation.
(*ii*) Show that the *AB* flip-flop can be converted to *D* flip-flop.

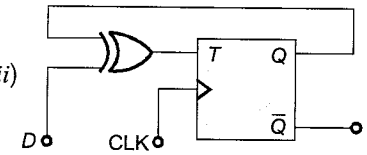
## Student's Assignments 2

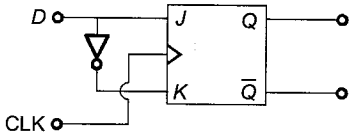
**Q.1** Consider an IC-FFs and choose the wrong statements regarding its Hold Time ($T_H$) and Setup Time ($T_S$):
P : $T_S$ and $T_H$ should be maximum.
Q : Only $T_H$ should be in maximum range.
R : $T_S$ and $T_H$ both should be minimum.
S : $T_S$ should be maximum but $T_H$ should be minimum.
(a) Only *R* (b) *P*, *Q* and *S*
(c) *P* and *Q* (d) Only *Q*

**Q.2** Which of the following circuit connection represents the D-FF?

(*i*)



(*ii*)


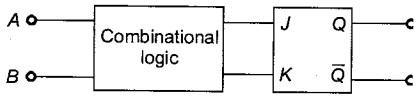

(*iii*)



(*iv*)



(a) (*ii*) and (*iii*) (b) (*ii*), (*iii*) and (*iv*)
(c) (*i*), (*ii*) and (*iii*) (d) (*ii*) and (*iii*)

**Q.3** Consider a circuit realisation of a combinational logic block as shown in figure to obtain the following truth table:

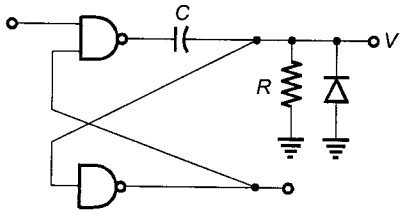
| A | B | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $\bar{Q}_n$ |
| 0 | 1 | 1 |
| 1 | 0 | $Q_n$ |
| 1 | 1 | 0 |




The combinational logic involves
(a) Only EX-OR gates
(b) AND and NOT gates
(c) NOT and EX-NOR gates
(d) OR and NAND gates

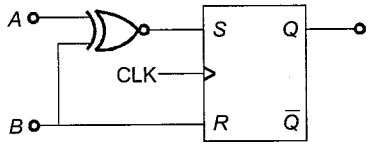
**Q.4** Which of the following flip-flop can not be converted to D-type (delay) flip-flop.
(a) S-R flip-flop
(b) J-K flip-flop
(c) Master slave flip-flop
(d) none of these

**Q.5** The circuit shown below is a



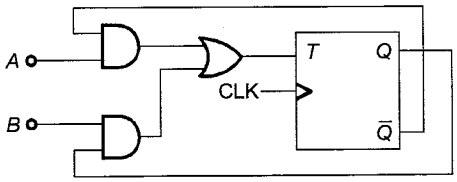

(a) Bistable multivibrator
(b) Astable multivibrator
(c) Monostable multivibrator
(d) Time-delay circuit

**Q.6** An $AB$ flip-flop is constructued from an S-R Flip-flop as shown in figure. The expression for next state $Q^+$ is



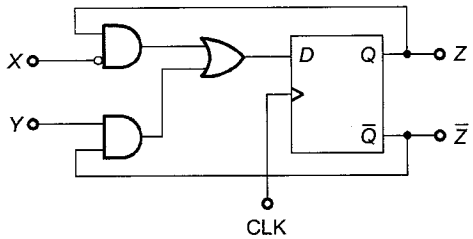

(a) $\overline{A}\overline{B}+AQ$  (b) $\overline{A}\overline{B}+\overline{B}Q$
(c) Both $A$ and $B$  (d) None of above

**Q.7** What is represented by digital circuit given below?




(a) An S-R flip-flop with $A = S$ and $B = R$
(b) A J-K flip-flop with $A = K$ and $B = J$
(c) A J-K flip-flop with $A = J$ and $B = K$
(d) An S-R flip-flop with $A = R$ and $B = S$

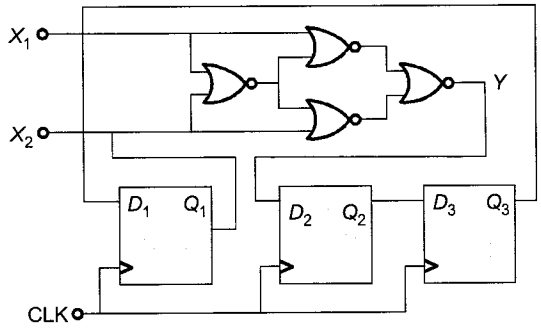
**Q.8** A sequential circuit using $D$ flip-flop and logic gates is shown in the figure where $X$ and $Y$ are the inputs and $Z$ is the output. The circuit is




(a) S-R FF with input $X = R$ & $Y = S$
(b) S-R FF with input $X = S$ & $Y = R$
(c) J-K FF with input $X = J$ & $Y = K$
(d) J-K FF with input $X = K$ & $Y = J$

**Q.9** For a flip-flop with provisions of preset and clear
(a) preset and clear operations are performed simultaneously.
(b) while clearing, preset is disabled.
(c) while presetting, clear is disabled.
(d) both (b) and (c) are correct.

**Q.10** The correct input-output relationship between $Y$ and $(X_1, X_2)$ is




(a) $Y = X_1 + X_2$  (b) $Y = X_1 X_2$
(c) $Y = X_1 \oplus X_2$  (d) $Y = \overline{X_1 \oplus X_2}$

**Q.11** The outputs $Q$ and $\overline{Q}$ of a master-slave flip-flop are connected to its $R$ and $S$ inputs respectively. Its output $Q$ when clock pulses are applied will be
(a) fixed 0 or 1
(b) permanently 0
(c) permanently 1
(d) complementing with every clock pulse

**Q.12** The output $Q_n$ of a J-K flip-flop is 1. It changes to 0 when a clock pulse is applied. The inputs $J$ and $K$ are respectively
(a) 0 and x  (b) 1 and x
(c) x and 0  (d) x and 1

**Q.13** A master-slave flip-flop is triggered
(a) when the clock input is at HIGH logic level
(b) when the clock input is at LOW logic level
(c) when the clock input makes a transition from LOW to HIGH
(d) when a pulse is applied at the clock input

***Answer Key:***

| | | | | |
|---|---|---|---|---|
| **1.** (b) | **2.** (b) | **3.** (c) | **4.** (d) | **5.** (c) |
| **6.** (c) | **7.** (c & a) | **8.** (d) | **9.** (d) | **10.** (d) |
| **11.** (d) | **12.** (d) | **13.** (d) | | |

CHAPTER 06

# Registers

## Introduction

- A register is a digital circuit with two basic functions i.e. data storage and data movement. It is basically a group of flip-flops logically connected to perform various functions.
- To store a group of *N*-bit word, the number of flip-flops required is '*N*' (one for each bit).
- We can also say that a register is a group of binary storage cells suitable for holding binary information. In addition to the flip-flops, a register may have combinational gates that perform certain data processing tasks. Thus a register consists of a group of FFs and gates that effect their transition.
- The simplest possible register is the one that consists of only flip-flop without any external gate, as shown in *Figure (6.1)*.

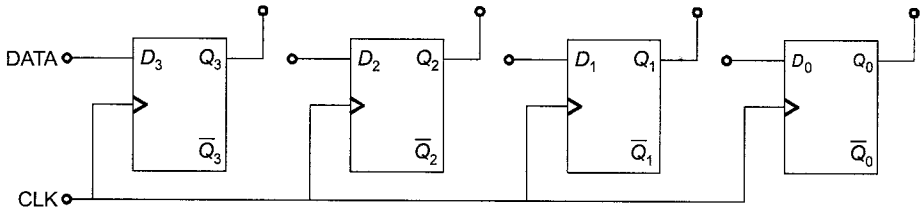



***Figure-6.1*** *General block diagram of a register*

- The data can be entered in SERIAL (one bit at a time) or in PARALLEL form (all the bits simultaneously) and can be retrieved in the serial or parallel form.

***Remember:*** Data in serial form is called temporal code & data in parallel form is called spatial code.

## 6.1 Shift Register

- A register capable of shifting the binary information entered into it from an external binary source is called the "shift register".
- It is a sequential circuit mainly used to store or shift binary data either to the right (called right shift register) or to the left (called left shift register).

- The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop, connected to the input of next flip-flop.
- All flip-flops receive a common CLK pulse which causes shift from one state to the next.
- In shift register each CLK pulse shifts the contents of register one bit position to the RIGHT or LEFT.

### 6.1.1 Classification of Shift Registers

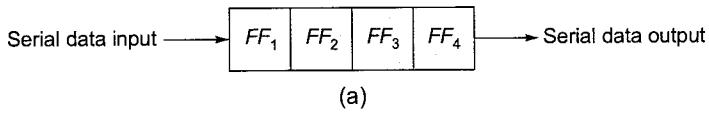
Shift registers can have a combination of serial and parallel shifted inputs and outputs including

(*i*) Serial IN, serial OUT shift register (SISO)
(*ii*) Serial IN, parallel OUT shift register (SIPO)
(*iii*) Parallel IN, serial OUT shift register (PISO)
(*iv*) Parallel IN, parallel OUT shift register (PIPO)

The 'serial input' determines, what goes into the left flip-flop during the shift, whereas the 'serial output' is taken from the output of the right most FF prior to the application of a pulse.

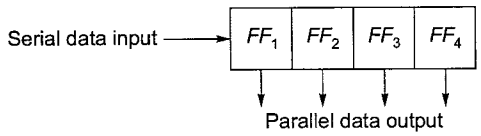
*Figure (6.2)* shows the classification of various registers.

- SISO



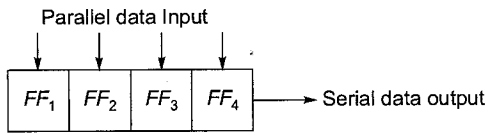

(a)

- SIPO



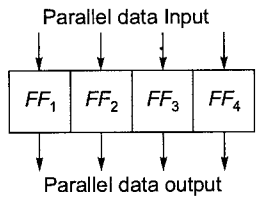

(b)

- PISO




(c)

- PIPO




(d)

***Figure-6.2*** *Classification of various shift registers*

**Serial IN Serial OUT Shift Register (SISO)**

- The serial IN serial OUT shift register accepts the data serially, one bit at a time on a single input line. It produces the stored binary information on its single output line in serial form.
- *Figure 6.3 (a)* shows the logic diagram of a 4 bit SISO register using *D* flip-flops. Let the binary information 1101 is applied to the input.

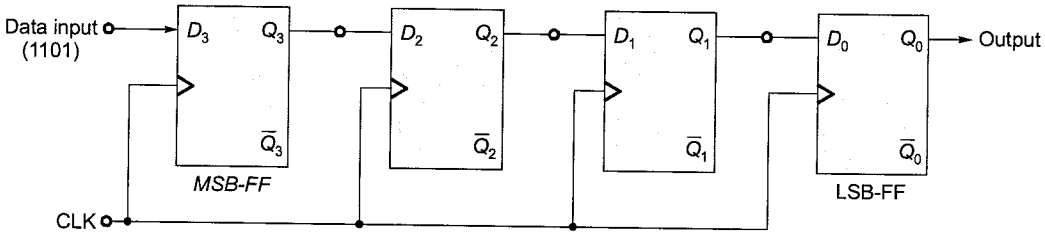



***Figure-6.3 (a)*** Right shift SISO register

- Since shift register is reset, so initially all the flip-flop outputs are zero i.e. 0000.
- Data 1101 is applied to the input line. Therefore in the right shift SISO register, *LSB* data is applied at the *MSB FF*($D_3$).

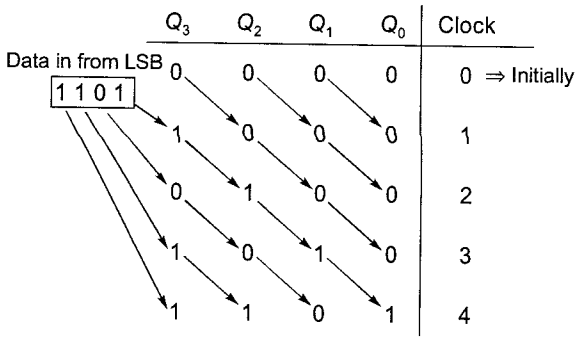



***Figure-6.3 (b)***

It is clear that the data 1101 stores from LSB in the right shift way after 4 clock pulses.

- Similarly *Figure 6.3 (c)* shows the logic diagram for left shift SISO shift register.

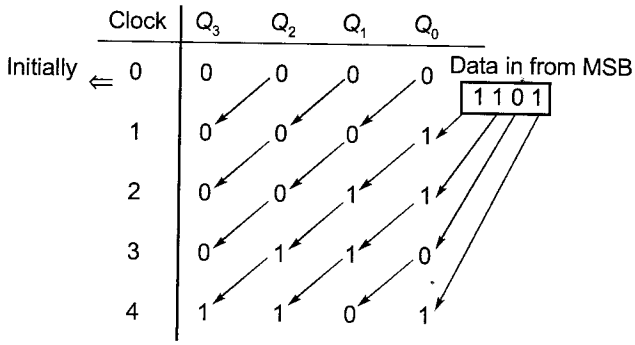



***Figure-6.3 (c)***

It is clear that the data 1101 stores from MSB in the left shift way after 4 clock pulses.

- In '$n$' bit register, to enter '$n$' bit data, it requires '$n$' clock pulses in serial form.
- If '$n$' bit data is stored in SISO register then output is taken serially for this it requires $(n-1)$ clock pulses.
- SISO register is used to provide '$n$' clock pulses delay to the input data.
- If '$T$' is the time period of clock pulse, then delay provided by SISO is $nT$.

**Bi-Directional Shift Register**

Such register which are capable of shifting the information (data) to right and left both is called "Bi-directional shift register".

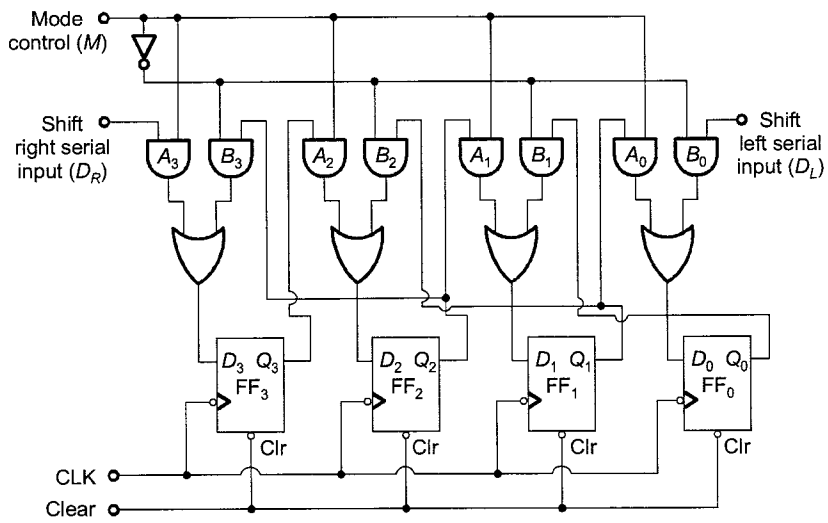



***Figure-6.4 :*** *Bi-directional shift register*

**Operation**

- When the Mode control $M = 1$, all the '$A$' AND gates are enabled and the data at $D_R$ is shifted to the right when clock pulses are applied.
- When $M = 0$, the $A$ gates are inhibited and $B$ gates are enabled allowing the data at $D_L$ to be shifted to the left.
- $M$ should be changed only when CLK = 0, otherwise the data stored in the register may be altered.

**Serial IN Parallel OUT Shift Register (SIPO)**

- *Figure 6.5(a)* shows, a 4 bit serial IN parallel OUT shift register. In this shift register, data is entered in serial form but output is taken in parallel form. Therefore, once the data is stored each bit is available on its respective output lines simultaneously.

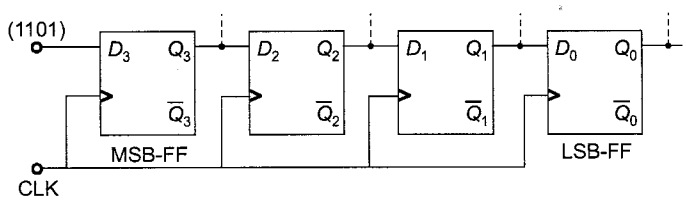



***Figure-6.5 (a)*** *SIPO shift register*

- For the data input of 1101 the table of operation is listed in *Figure 6.5 (b).*

Input Data 1101

| $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | Clock | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | ⇒ Initially |
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 1 | 2 | |
| 0 | 1 | 1 | 0 | 3 | |
| 1 | 1 | 0 | 1 | 4 | |

***Figure-6.5 (b)***

- For storing $n$-bit serial input data number of CLK pulses required = $n$.
- For $n$-bit parallel output data to be stored the number of CLK pulses required = 0.

**Parallel IN Serial OUT Shift Register (PISO)**

- In PISO shift register the data bits are entered simultaneously into the respective flip-flops and the shifted data is available at the output serially.
- The *Figure 6.6* shows the 4 bit PISO shift register.

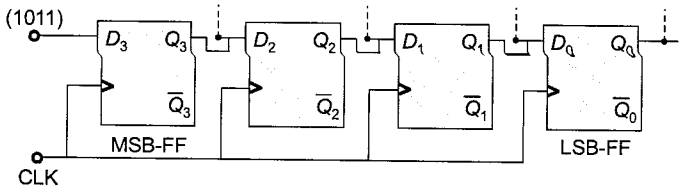



***Figure-6.6*** *PISO shift register*

- To store parallel data of $n$-bit it requires 1 CLK pulse.
- To store serial output data of $n$-bit the number of CLK pulses required are $(n-1)$.

**Parallel IN Parallel OUT Shift Register**

- *Figure 6.7* shows the 4 bit parallel in parallel out shift register. In this type of shift register, whole of the data appears simultaneously along with all the flip-flops inputs and outputs.

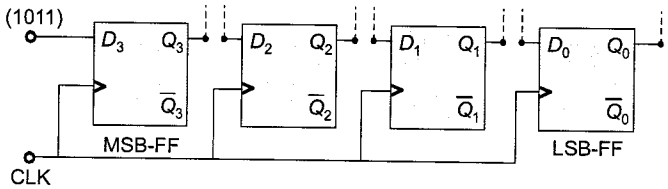
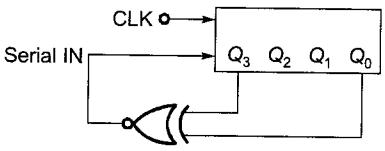



***Figure-6.7*** *PIPO shift register*

- For parallel IN data the number of clock pulse required is 1.
- For parallel OUT data the number of CLK pulse required is 0.

**Example - 6.1** A four bit serial in parallel out shift register is used with a feedback as shown in below figure. The shifting sequence is $Q_3 \rightarrow Q_2 \rightarrow Q_1 \rightarrow Q_0$. If the output is initially 0000, the number of clock pulses after which the output will repeat itself is




***Solution:***

From the given serial IN parallel OUT circuit the next state can be obtained as

| $CP$ | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 |

Thus, the number of clock pulses required is = 6.

### 6.1.2 Difference Between Serial Data Transfer and Parallel Data Transfer

**Table-6.1**

| Serial Data Transfer | Parallel Data Transfer |
|---|---|
| 1. For complete transfer of n bits of data it requires *n* CLK pulses. | 1. During a single CLK pulse, all the data is transferred simultaneously. |
| 2. It is slower. | 2. It is relatively faster. |
| 3. They are having less inter-connections between sending register and receiving register. | 3. They requires relatively more interconnections. |
| 4. The circuit is simple. | 4. The circuit is relatively complex. |

### 6.1.3 Applications of Shift Registers

The primary uses of shift registers are temporary data storage and bit manipulations. In addition to this, other common applications are given below:

**Time-Delay**

- A SISO shift register may be used to introduce time delay "$\Delta t$" in digital signals given by:

$$\boxed{\Delta t = N \times T = N \times \frac{1}{f_c}}$$

where, $N$ = Number of FFs
$T$ = Time period of CLK pulse
$f_c$ = CLK frequency

- The amount of delay can be controlled by the "$f_c$" or number of FFs in the SR.

**Data Conversion**

- Data in the serial form can be converted into parallel form by using SIPO-SR.
- Data in the parallel form can be converted into serial form by using PISO-SR.

**Ring Counter**

- If the output of the shift register is connnected back to the input then an injected pulse will keep circulating, this circuit is referred to as the "Ring counter".

**Sequence Generator**

- A circuit which generates a prescribed sequence of bits, in synchronism with a CLK, is referred to as a "sequence generator".

**Arithmetic Operation**

- Shift registers are used to perform various arithmetic operations e.g. "serial adder" adds two binary numbers.

**Example-6.2** Which of the following capabilities are available in a Universal Shift Register?

1. Shift left
2. Shift right
3. Parallel load
4. Serial add

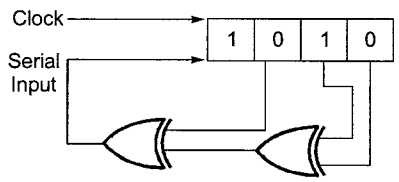
Select the correct answer from the codes given below :

(a) 2 and 4 only
(b) 1, 2 and 3
(c) 1, 2 and 4
(d) 1, 3 and 4

***Solution: (b)***

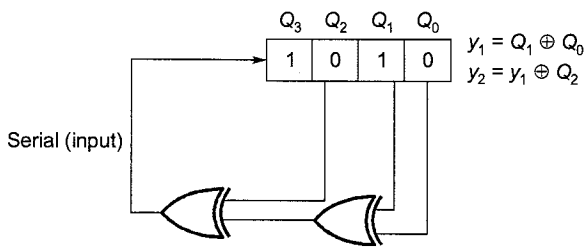
Universal shift register features parallel inputs, parallel outputs. It performs both functions shift left and shift right.

**Example-6.3** The shift register shown in figure is initially loaded with the bit pattern 1010. Subsequently the shift register is clocked, and with each clock pulse the pattern gets shifted by one bit position to the right. With each shift, the bit at the serial input is pushed to the left most position (*MSB*). After how many clock pulses will the content of the shift register become 1010 again?




(a) 3
(b) 7
(c) 11
(d) 15

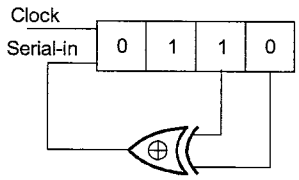
***Solution: (b)***




$$Q_3(t+1) = Q_0(t) \oplus Q_1(t) \oplus Q_2(t)$$

| CLK pulse | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 |
| 2 | 1 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 0 | 0 |
| 7 | 0 | 1 | 0 | 0 |
| | 1 | 0 | 1 | 0 |

**Example- 6.4** The initial contents of the 4-bit serial-in-parallel-out, right shift, shift register shown in the given figure is 0110. After three clock pulses are applied, the contents of the shift register will be

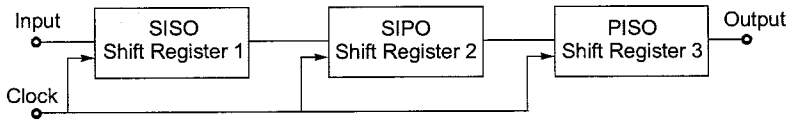



(a) 0000
(b) 0101
(c) 1010
(d) 1111

***Solution : (c)***

After 1 clock pulse, contents are 1011.
After 2 clock pulses, contents are 0101.
After 3 clock pulses, contents are 1010.

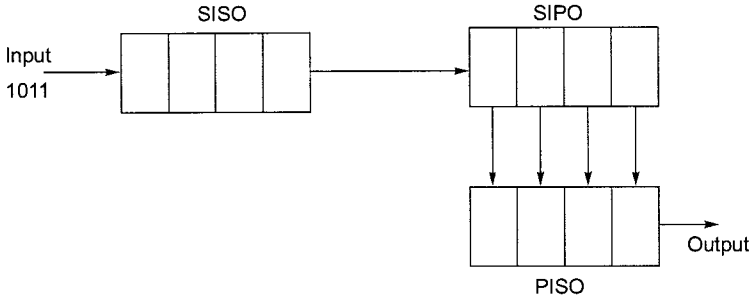
**Example - 6.5** Three 4 bit shift registers are connected in cascade as shown in figure below. Each register is applied with




A 4 bit data 1011 is applied to the shift register 1. The minimum number of clockpulses required, to get same output data as the input data with the same clock pulse, will be

(a) 11 (b) 12
(c) 9 (d) 7

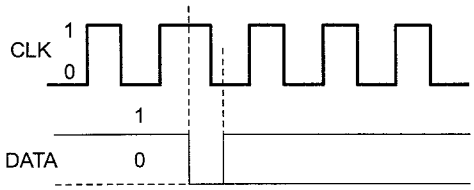
***Solution : (b)***




| | SISO | SIPO | PISO |
|---|---|---|---|
| Initially | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |
| 1. | 1 0 0 0 | 0 0 0 0 | 0 0 0 0 |
| 2. | 1 1 0 0 | 0 0 0 0 | 0 0 0 0 |
| 3. | 0 1 1 0 | 0 0 0 0 | 0 0 0 0 |
| 4. | 1 0 1 1 | 0 0 0 0 | 0 0 0 0 |
| 5. | | 1 0 0 0 | 0 0 0 0 |
| 6. | | 1 1 0 0 | 1 0 0 0 |
| 7. | | 0 1 1 0 | 1 1 0 0 |
| 8. | | 1 0 1 1 | 0 1 1 0 |
| 9. | | | 1 0 1 1 |
| 10. | | | × 1 0 1 |
| 11. | | | × × 1 0 |
| 12. | | | × × × 1 |

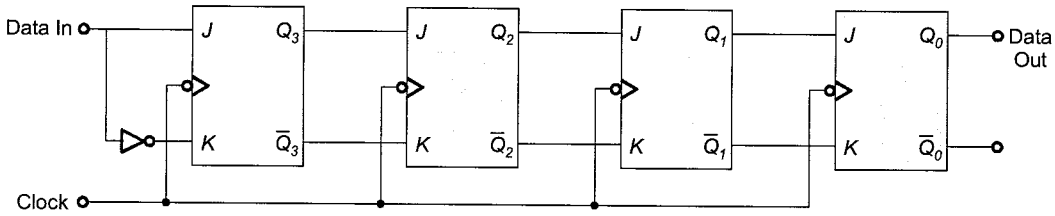
Minimum number of clock pulses required = 12.

**Example - 6.6** Draw the diagram of a 4-bit shift register using JK flip-flops. Each flip-flop triggers on the negative going transition. Draw the output waveforms for all flip-flops when the input data and clock signals are as shown.

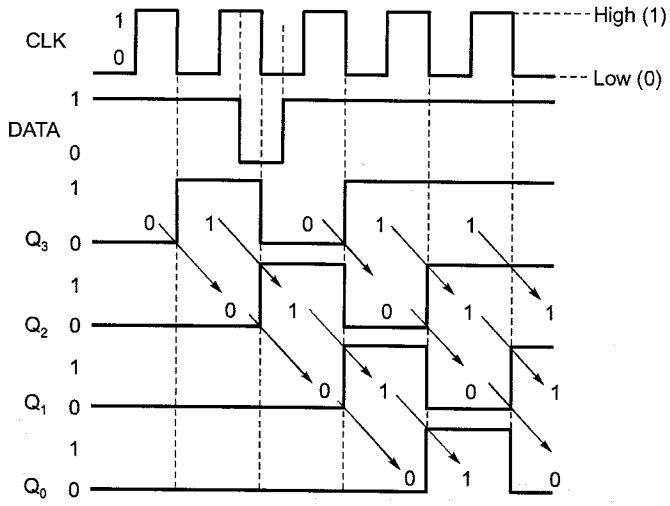


***Solution:***

The diagram of a 4-bit shift register using JK flip-flops is shown below:




The output waveforms of all flip-flops for the given data are shown below




## Summary

- All shift registers are JK-FFs.
- "PIPO" register is a storage register made up with D-FFs.
- "PIPO" register is not a shift register.
- Each Shift Left operation multiply the data by 2 and each Shift Right operation divides the data by 2.
- A register can always replace a Latch, but the converse should be done with caution to make sure that outputs from a latch never go to other FFs inputs that are triggered with same CLK pulse.
- The transfer of new information (data) into a register is referred to as "Loading" the register.
- A group of FFs sensitive to the pulse duration is usually called "gated latch" and a group of FFs sensitive to the pulse transition is called a 'register'.
- In storage registers mostly D flip-flops are used.
- The FFs hold binary information and the gate controls WHEN and HOW, new information is transferred into the register.
- To convert temporal code into spatial code, we use SIPO register, while to convert spatial code into temporal code use PISO register.

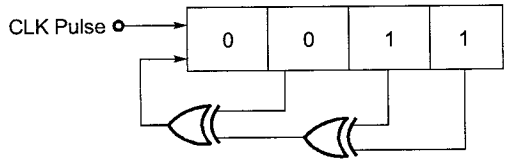
## Student's Assignments 1

**Q.1** What do you understand by the Bi-directional shift register and universal shift register? Give its function table and draw the logical circuit diagram of a 4-bit bi-directional shift register.

**Q.2** What is the difference between serial and parallel transfer?
Explain how to convert serial data to parallel data and vice-versa.

## Student's Assignments 2

**Q.1** The shift register shown below is initially loaded with binary bit stream 0011. If clock pulses are applied continuously and after some clock pulses the loaded data is repeated in shift register. During this, which pulse is responsible for the highest decimal data to be stored in the shift register?



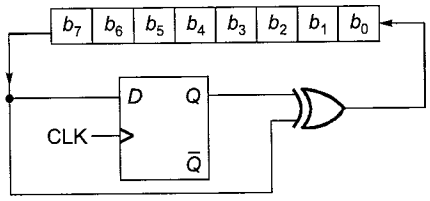

(a) $4^{th}$ (b) $5^{th}$
(c) $6^{th}$ (d) $2^{nd}$

**Q.2** A SISO shift registers may be used to introduce time delay $\Delta t = 200$ ns in digital signals. Assume this shift registers is manufactured by '$N$' number of $D$-flip-flops having clock frequency 50 MHz. The required value of '$N$' is
(a) 10 (b) 100
(c) 9 (d) 5

**Q.3** Data can be changed from spatial code to temporal code and viceversa by using
(a) ADCs and DACs
(b) Shift registers
(c) Synchronous counter
(d) Timers

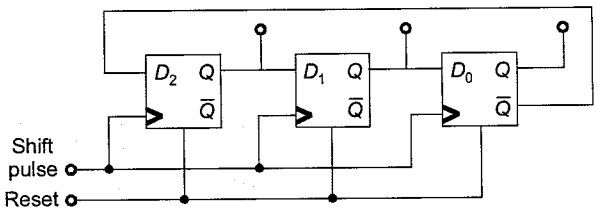
**Q.4** Shifting a register to the left by one bit position is equivalent to
(a) division by 2 (b) multiplication by 2
(c) addition of 2 (d) subtraction of 2

**Q.5** An 8-bit left shift register and $D$-flip-flop shown in figure is synchronized with same clock. The $D$ flip-flop is initially cleared.




The circuit acts as
(a) Binary to 2's complement converter
(b) Binary to Gray code converter
(c) Binary to 1's complement converter
(d) Binary to Excess- 3 code converter

**Q.6** A three-bit shift register is shown in the given figure.




To have the content '000' again, the number of clock pulses required should be
(a) 3 (b) 6
(c) 8 (d) 16

**Q.7** A universal register
(a) accepts serial input
(b) accepts parallel input
(c) gives serial and parallel outputs
(d) is capable of all of the above

***Answer Key:***

**1.** (b) **2.** (a) **3.** (b) **4.** (b)
**5.** (b) **6.** (b) **7.** (d)

CHAPTER 07

# Counters

## Introduction

- A counter is a circuit that counts the number of occurrences of an input (in terms of positive or negative edge transitions in the case of binary input). Counter circuits are primarily constituted of flip-flops which along with the combinational elements are used for generation of control signals. The flip-flops are interconnected such that their combined state at any time is the binary equivalent of the total number of pulses that have occurred upto that time. Thus a counter is used to count pulses.
- Each count, a binary number is called a state of a counter. Hence, a counter counting in terms of $n$-bits (n flip-flops) has $2^n$ different states.
- The number of different states in counting sequence is also known as the modulus of the counter. Thus, for '$n$' flip-flops counter will have '$2^n$' different states and then the counter is said to be "MOD-2" counter.

  The MOD number represents the frequency division obtained from the last flip-flop. It would be capable of counting upto $(2^n - 1)$ before returning to zero state.

  Therefore, If $M$ = Total number of states

  and $n$ = Total number of FFs

  Then, $\boxed{M \le 2^n}$

  If $M = 2^n$; Binary counter

  $M < 2^n$; Non-binary counter
- Depending upon the manner in which the flip-flops are triggered, counters can be divided into two major categories:

  (*i*) Asynchronous counter (Ripple/series counter).

  (*ii*) Synchronous counter (Parallel counter).

  The comparison between synchronous and asynchronous counter is listed in *Table (7.1)*.

**Table-7.1**

| Synchronous Counter | Asynchronous Counter |
|---|---|
| 1. All the flip-flops are triggered simultaneously with the same clock. | 1. Different FFs are triggered with different clock. |
| 2. Operation is faster. | 2. Operation is slower. |
| 3. Any required sequence can be designed. | 3. Will operate only in a fixed count sequence. |
| 4. No decoding error occurs. | 4. Decoding error occurs due to '$t_{pd}$'. |
| 5. Designing is complex as the number of states increases. | 5. Designing is easy even for more number of states. |
| 6. e.g. Ring counter, Johnson counter. | 6. e.g. Ripple UP counter, Ripple DOWN counter. |

- **Up/Down Counter:** If a counter counts in such a way that the decimal equivalent of the output increases with successive clock pulses, is called 'UP counter'and if it decreases, it is called 'Down counter'. An Up/Down counter can also be designed which can count in any direction depending upon the control input.

## Application of Counters

- To count the number of CLK Pulses.
- To count the number of items in industry.
- Works as a "Frequency divider".
- Used in time measurement.
- For distance measurement in RADAR system.
- In Analog to Digital converter (ADC).
- In measurement of PRI (Pulse Repetition Interference).

**Remember**

- In "MOD-N Counter", if applied input frequency is "$f$", then the output frequency is $f/N$.
- If two counters are cascaded with MOD-M followed by MOD-N, then number of overall states of combined counter is ($M \times N$) and counter is called "MOD-MN" counter.

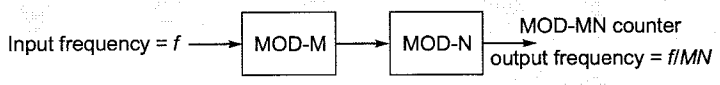



***Figure-7.1***

**Example-7.1** A certain J-K flip-flop has $t_{pd} = 12\ ns$, the largest 'MOD' counter that can be constructed from cascading these FFs and still operating upto 10 MHz is

***Solution:***

$$\text{Time period of clock pulse} = \frac{1}{10\,\text{MHz}} = 10^{-7}\text{sec.}$$

$$\text{Propagation delay of each FF} = 12 \times 10^{-9}\ \text{sec.}$$

$$\therefore \quad \text{Number of FF required} = \frac{t_{CLK}}{t_{pd}} = \frac{10^{-7}}{12 \times 10^{-9}} = 8.33$$

Therefore, the number of FF required = 8 (As MOD $\le 2^N$)

and Modulus of counter = $2^8 = 256$

## 7.1 Asynchronous/Ripple Counters

- In a ripple counter, there is no clock or source of synchronizing the pulses, however, the state change still occurs due to pulses at clock input of the flip-flops.
- Here, the first flip-flop is clocked by external clock pulse and each successive flip-flop is clocked from the previous output.

### 7.1.1 3-Bit Ripple Counter

- *Figure* 7.2 (a) shows a 3-bit binary ripple counter which consists of a series connection of complementing J-K flip-flops, with the output of each flip-flop connected to the clock pulse (*CP*) input of the next higher order flip-flop.
- The flip-flop holding the *LSB* receives the incoming clock pulses.

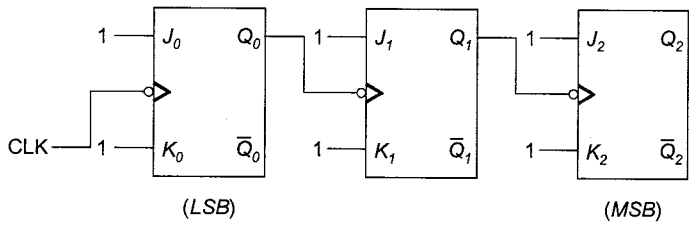



***Figure-7.2 (a)*** *Logic diagram of a 3-bit ripple counter*

**Operation**

- It is assumed that the flip-flops change their state on the negative going edge of clock pulse.
- Note that all the flip-flops are considered in toggled mode.
- $Q_0$ will change its state in every clock pulse.
- $Q_1$ will change its state when $Q_0$ will change from 1 to 0.
- $Q_2$ will change its state when $Q_1$ will change from 1 to 0.
- Depending upon the operation, the transition table is shown in *Table 7.2*.

**Table-7.2** *Transition table/truth table*

| CLK | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|
| 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 |
| 8 | 0 | 0 | 0 |

- Initially all FFs are set to zero.
- The maximum possible state = 8 (from 0 to 7).
- The timing diagram of 3 bit ripple counter is shown in *Figure* 7.2 (b).

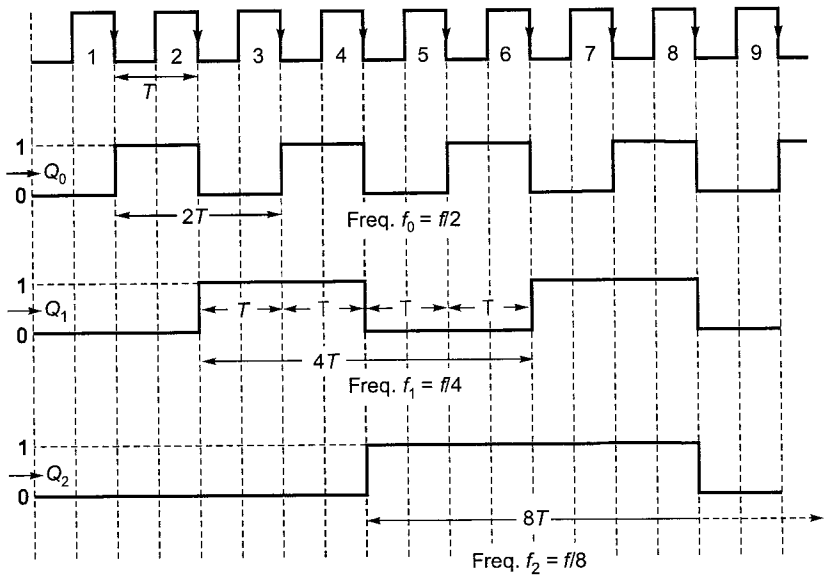



***Figure-7.2 (b)***

- If the input clock frequency is 'f' then the output frequency = $f/8$.
- For an '$N$' bit ripple counter, if the propagation delay of each flip-flop is $t_{pd}$, then the period of clock is given by

$$\boxed{T_{CLK} \geq N \cdot t_{pd}}$$

or

$$\boxed{f_{CLK} \leq \frac{1}{Nt_{pd}}}$$

and

$$\boxed{f_{max} = \frac{1}{Nt_{pd}}}$$

- For determination of Up/Down counter:

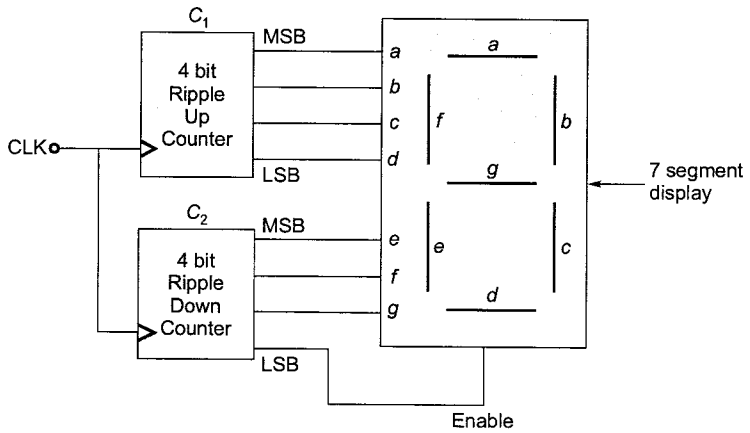
| Trigger Applied | CLK is given by | Access as |
|---|---|---|
| Negative edge | $Q$ | Up counter |
| Positive edge | $Q$ | Down counter |
| Negative edge | $\overline{Q}$ | Down counter |
| Positive edge | $\overline{Q}$ | Up counter |

- The major disadvantage of a ripple counter is the error due to the propagation delay of flip-flops i.e. $t_{pd}$ which, is known as decoding error or transient state. To overcome the decoding error in ripple counter we may use "strobe signal".
  Thus, with strobe signal

$$\boxed{T_{CLK} \geq Nt_{pd} + T_s}$$

  where, $T_s$ = Delay of AND gate (used to give strobe signal)
- In the counter discussed above all the states are used to get the output and therefore called as binary ripple counter.

**Example-7.2** Consider the circuit given below




If Enable = 0 ; 7 segment display 11 ($b = c = e = f = 1$)

Enable = 1 ; 7 segment display data according to Inputs

Initially both the counter were cleared. After 78 clock pulses the data displayed on the 7 segment display is _____.

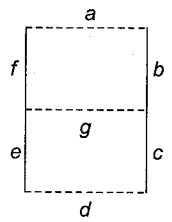
***Solution:***

After 78 clock pulse

Output of counter $C_1 = (1110)_2 = (14)_{10}$

Output of counter $C_2 = (0010)_2 = (2)_{10}$

7 segment display




| $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | Enable |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

⇒ Data displayed on 7 segment is '11'

## 7.1.2 Asynchronous Inputs

- In the clocked flip-flops discussed in previous chapters the inputs to *S-R*, *J-K*, *D* and *T* are called synchronous inputs because their effect on the output of the flip-flop is synchronized with the clock input.
- There are most IC flip-flops available, which have one or more asynchronous inputs. These asynchronous inputs affect the flip-flop output independently from the synchronous inputs and clock input.
- These inputs can be used to SET the flip-flop (1) or RESET the flip-flop (0) at any instant of time regardless of the conditions at other inputs.
- There are mainly two types of asynchronous inputs labelled-PRESET (PRE) and CLEAR (CLR).
- An active HIGH level on PRESET input will set the FF and active HIGH level on CLEAR input will RESET the FF. Similarity an active LOW level on PRESET input will SET the FF and on CLEAR input will RESET the FF as shown in *Figure* 7.3.

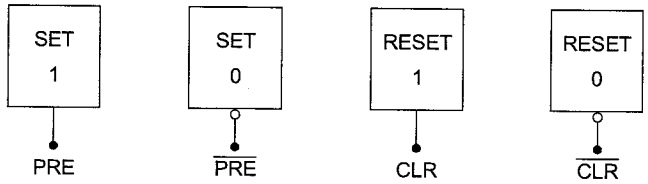
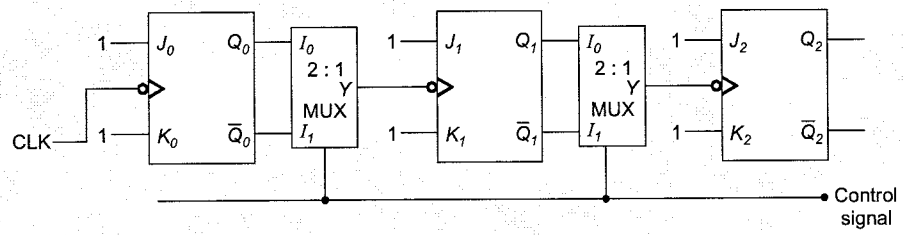



***Figure-7.3***

**Do You Know ?**

A ripple counter can be made to work as an up/down counter using 2 : 1 MUX and a control (signal) as shown in figure.




If, Control = 0 ; clock applied with $Q$ ∴ Up counter

If, Control = 1 ; clock applied with $\overline{Q}$ ∴ Down counter

### 7.1.3 Non Binary Counter

- For a non binary counter $M < 2^n$.
- *Figure 7.4* shows a mod-10 counter or a **decade** counter. Here the total number of flip-flops required is 4 thus the number of used state = 10 and the number of unused state = 6.

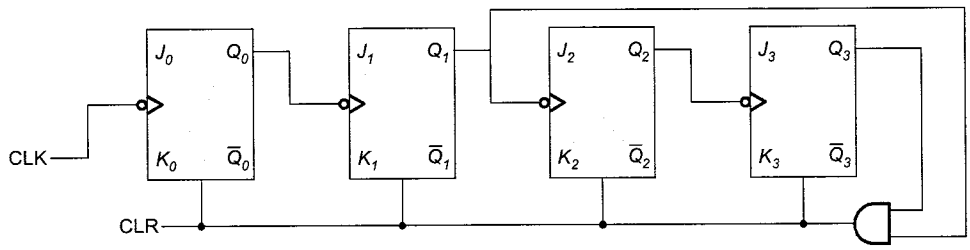



***Figure-7.4*** *Decade counter*

- In order to design a non binary decade counter a logic gate is used which detects 10 stage from 0000 to 1001 and as soon as 1010 appears it clears all the flip-flops as shown in *Figure* 7.4.
- *Table* 7.3 shows the transition table of a decade counter.

**Table-7.3** *Truth table*

| CLK | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |

- It is clear from the truth table that as soon as 1010 appears, inputs to the AND gate becomes 11 which resets the flip-flop with 'CLR' = 1.
- The output frequency of MOD-10 counter = $f/10$.
- If there is no feedback present at $Q_3$ then the output frequency = $f/16$.
- When decade counter counts from 0 to 9 then it is known BCD counter.

**NOTE**

For making non binary counter, if 'CLR' is present and CLK is connected with output $Q$, then we use AND gate, (*Figure* 7.4). Similarly

$$\text{CLR} \Rightarrow Q \Rightarrow \text{AND Gate}$$

$$\text{CLR} \Rightarrow \bar{Q} \Rightarrow \text{NOR Gate}$$

$$\overline{\text{CLR}} \Rightarrow Q \Rightarrow \text{NAND Gate}$$

$$\overline{\text{CLR}} \Rightarrow \bar{Q} \Rightarrow \text{OR Gate}$$

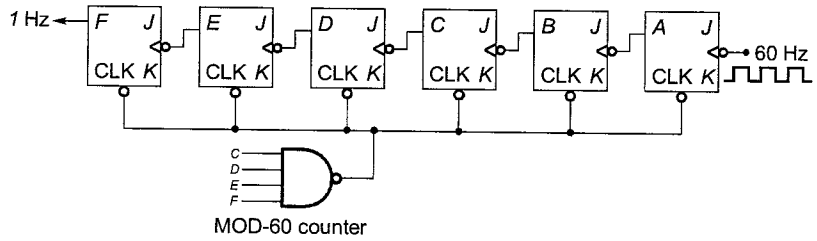
**Example-7.3** MOD-60 counter was needed to divide the 60-Hz line frequency down to 1 Hz. Construct an appropriate MOD-60 counter.

***Solution:***

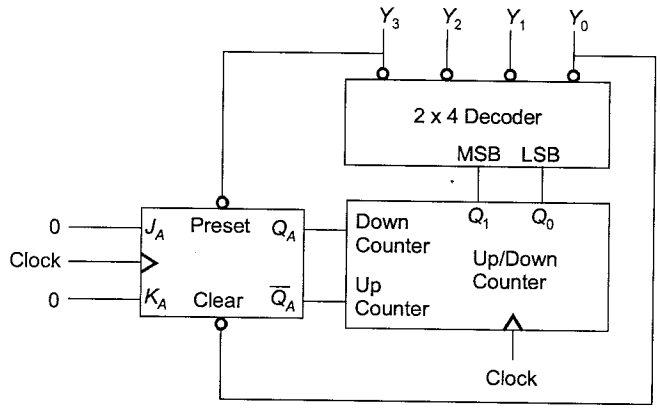
$$\text{MOD} \le 2^N$$

$$60 \le 2^N; \qquad \text{so, } N = 6$$

Hence, we need six FFs. The counter is to be cleared when it reaches the count of sixty (111100). Thus, the outputs of flip-flops $C$, $D$, $E$ and $F$ must be connected to the NAND gate. The output of flip-flop $F$ will have a frequency of 1 Hz.




**Example-7.4** Consider the circuit given below




Assuming the initial value of counter output ($Q_1$, $Q_0$) as zero, the counter output for 8 clock pulses in decimal form is

(a) 0, 3, 2 ,1, 0, 1, 2, 3
(b) 0, 1, 2, 3, 2, 1, 0, 1
(c) 0, 1, 2, 3, 3, 2, 1, 0
(d) 0, 3, 2, 1, 0, 0, 1, 2

***Solution: (b)***

Initially, $Q_0 = Q_1 = 0$

$\Rightarrow$ $Y_0 = 0$

$\Rightarrow$ *J-K* flip-flop is cleared

$Q_A = 0$ $\bar{Q}_A = 1$

As clock pulse is applied counter starts up counting

As counter reaches $Q_1 = 1$, $Q_0 = 1$ after 3 clock pulses, *J-K* flip flop is preset.

$\Rightarrow$ $Q_A = 1$ $\bar{Q}_A = 0$

$\Rightarrow$ Counter starts down counting until $Y_0$ is low and this repeats.

So, Output $Q_1 Q_0$ in decimal form is

0, 1, 2, 3, 2, 1, 0, 1....

## 7.2 Synchronous Counters

- The problem encountered with ripple counters are caused due to the accumulated FFs propagation delay. In other words, the FFs do not change states simultaneously in synchronism with the input pulses. These limitations can be overcome with the use of synchronous counters or parallel counters in which, all the flip-flops are triggered simultaneously by the CLK input pulses. The maximum operating frequency for this counter will be significantly higher than that of the corresponding ripple counter.
- The synchronous counters are classified as:
  - (*i*) Shift register counters
    1. Ring counter
    2. Twisted ring counter/Johnson counter
  - (*ii*) Series carry counter
  - (*iii*) Parallel carry counter

### 7.2.1 Shift Register Counters

- One of the applications of shift register is that they can be arranged to form different types of counters. Shift register counters are obtained from serial-in serial-out (SISO) shift register by providing feedback from the output of the last FF to the input of the first FF.
- The most widely used shift register counters are:
  1. Ring counter.
  2. Twisted ring counter.

**Ring Counters**

- It is the simplest shift register counter.
- It is also called "end carry counter".
- The *Figure* 7.5 (a) shows the logic diagram of a four bit ring counter using *D* FF. Its state diagram and transition table is shown in *Figure* 7.5 (b) *and* (c) respectively.

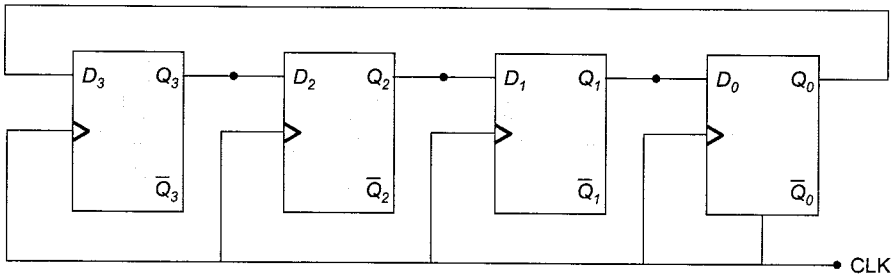
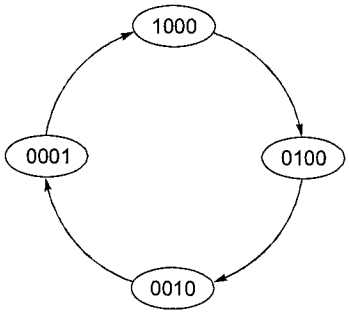



***Figure-7.5 (a)*** *Logic diagram of a ring counter*

- It is clear from the logic diagram that the *Q* output of each state is connected to *D* input of the next state. However the output $Q_0$ of *LSB FF* is connected back to the input $D_3$ of *MSB FF* such that an array of *FFs* is arranged in a ring and, hence, the name 'ring counter'.
- In this only one bit is high and is made to circulate around the register as long as clock pulses are applied.




(b) State diagram

| CLK | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 0 | 4-state |
| 2 | 0 | 1 | 0 | 0 | |
| 3 | 0 | 0 | 1 | 0 | |
| 4 | 0 | 0 | 0 | 1 | |
| 5 | 1 | 0 | 0 | 0 | 4-state |
| 6 | 0 | 1 | 0 | 0 | |
| 7 | 0 | 0 | 1 | 0 | |
| 8 | 0 | 0 | 0 | 1 | |
| 9 | 1 | 0 | 0 | 0 | |

(c) Truth table

***Figure-7.5 (b) and (c)***

- As the sequence repeats itself after four clock pulses, thus, the number of distinct states in the ring counter is equal to the number of FFs used in the counter.
  Therefore, with n flip-flops, there are n-states present in ring counter.

  Hence, $\boxed{\text{MOD-M} = n}$

- With '*n*' flip-flops, maximum count possible in ring counter is $(2^{n-1})$.
- The timing diagram is shown in *Figure* 7.5 (d).

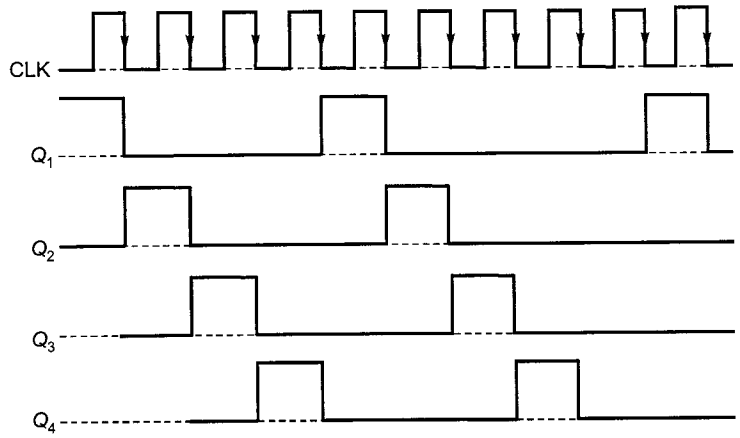



***Figure-7.5(d)*** *Timing diagram of a 4-bit ring counter*

- Thus, the number of unused states in ring counter is $(2^n - n)$.

**Advantages**

Decoding is very easy in ring counter, because there is no aid of extra circuits.

**Disadvantages**

Outputs are not symmetric.

**Self-Starting Ring Counter**

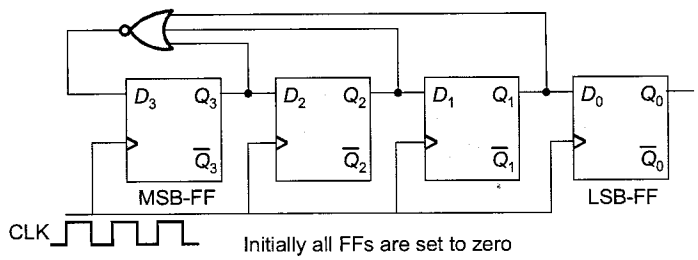



***Figure-7.6 (a)*** *Self starting ring counter*

| Clock | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 0 | |
| 2 | 0 | 1 | 0 | 0 | |
| 3 | 0 | 0 | 1 | 0 | 4-states |
| 4 | 0 | 0 | 0 | 1 | |
| 5 | 1 | 0 | 0 | 0 | |
| 6 | 0 | 1 | 0 | 0 | |
| 7 | 0 | 0 | 1 | 0 | 4-states |
| 8 | 0 | 0 | 0 | 1 | |
| 9 | 1 | 0 | 0 | 0 | |

***Figure-7.6 (b)***

- In 4-bit ring counter the used states = 4 and the unused input = $2^4 - 4 = 12$.
  In any counter if CLK frequency is "$f$" the FFs output frequency is "$f/N$" (where $N$ = No. of states).

**Applications of Ring Counter**

- In analog to digital converter
- In stepper motors
- In controlled signal generation such as interrupts

**Twisted Ring Counter (Johnson Counter)**

- The Johnson counter is a SISO shift register obtained by providing feedback from the inverted output of the last FF to the input of the first flip-flop.
- It is also know as
  - (*i*) Twisted ring counter
  - (*ii*) Switch tail ring counter
  - (*iii*) Mobies counter
  - (*iv*) Creeping counter
  - (*v*) Walking counter
- *Figure* 7.7 (a) shows the logic diagram of a 4-bit Johnson counter. It is clear that the $Q$ output of each stage is connected to the $D$ input of the next stage, however the output $\overline{Q}$ of the last stage is connected to the $D$ input of the first stage, therefore, the name 'twisted ring counter'.

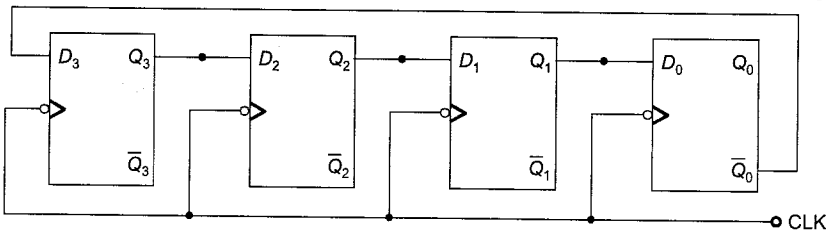
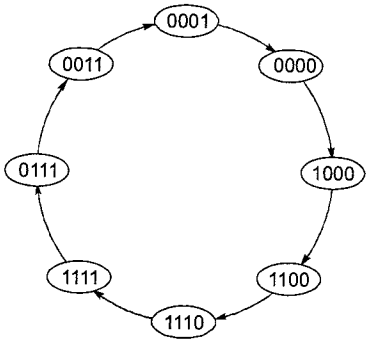
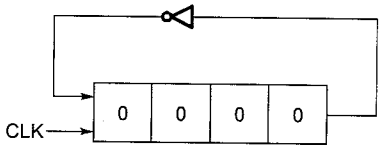



***Figure-7.7 (a)*** *Twisted ring counter*

- The state diagram and the transition table is shown in *Figure* 7.7 (b) and (d) respectively. Let initially all the FFs are reset.




**(b)**




**(c)**

| CLK | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 0 |

**(d)**

***Figure-7.7*** *(b), (c) and (d)*

- It is clear from the truth table, that the sequence is repeated after every eighth clock pulse. Thus for '$n$' FFs there are $2n$ possible states.

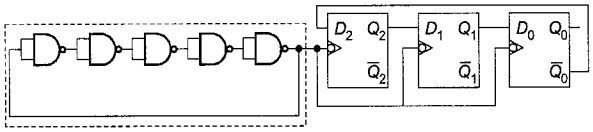
$$\boxed{\text{MOD-M} = 2n}$$

- The total number of unused state = $(2^n - 2n)$.

- In normal "Johnson counter" with '$n$' flip-flops and the input frequency is '$f$' then the output frequency is $f/2n$.
- When a counter enters into an unused state, it will persist in moving from one unused state to another and will never find its way to a used state.
- Thus, counter is said to be suffered from the problem of "LOCK OUT". This problem can be overcome by adding a Gate.
- In a 'counter' if a feedback connection is used, the number of possible states will decrease.

**NOTE:** In Johnson counter to decode each state two input AND gate or NOR gate is used.

**Example-7.5** Consider the digital circuit shown in the given figure




Clock Generator

The average propagation delay of each NAND gate in the clock generator circuit is 10 ns. Calculate the frequency of the signal at $Q_0$.

***Solution:***

Here the clock generator is a ring oscillator circuit

$$\therefore \quad f_{CLK} = \frac{1}{2Nt_{pd}}$$

$N$ = Number of logic gates

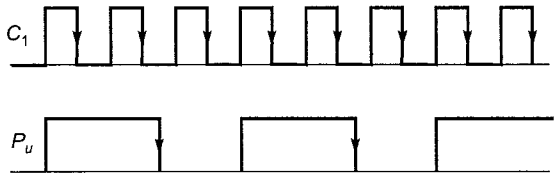
$t_{pd}$ = Propagation delay of each logic gate

$$f_{CLK} = \frac{1}{2\times5\times10\times10^{-9}} = 10\text{MHz}$$

The clocked sequential circuit is a 3-bit Johnson counter

$$\therefore \quad f_{Q_0} = \frac{f_{CLK}}{2\times3} = \frac{10\text{ MHz}}{6} = 1.66\text{ MHz}$$

**Example-7.6** The square wave $C_1$ shown in figure is given to the clock input of a 4-bit binary Up/Down counter whose Up/Down input is fed with the pulse train $P_u$. The counter is a negative edge triggered one. The counter starts with 0000 and will reach 0000 again at the




(a) 15th clock pulse
(b) 16th clock pulse
(c) 44th clock pulse
(d) 48th clock pulse

***Solution: (c)***

Here in every $P_u$ pulse there are 3 clock pulses. For initial two clock pulses counter will work as up counter as $P_u$ is high and for 3rd clock pulse it will work as down counter so in every three clock pulse it will be incremented by only 1 (2 increment and one decrement), so after $14 \times 3 = 42$ clock pulses counter goes to 14. After that $P_u$ goes high for two clock pulses and counter will be incremented by 2 is $14 + 2 = 16$ i.e. equivalent to 0000. So after $42 + 2 = 44$ clock pulses counter will be go to 0000 again.

**Example-7.7** The 14-bit timer is loaded with the counter value of 07D0 H. The timer input is connected to a clock with a frequency of 800 KHz. The timer is programmed to produce a continuous signature wave output. The frequency of the square wave output is

(a) 400 kHz
(b) 800 kHz
(c) 400 Hz
(d) 2000 kHz

***Solution: (c)***

$(07D0)H \rightarrow (0 \times 16^3 + 7 \times 16^2 + 13 \times 16^1 + 0 \times 16^0) = (2000)_{10}$

$$\text{frequency of square wave generated} = \frac{800 \times 10^3}{2000} = 400\text{ Hz}$$

**Example-7.8** 12 MHz clock frequency is applied to a cascaded counter of modulus-3 counter, modulus-4 counter and modulus-5 counter. What are the lowest output frequency and the overall modulus, respectively?

(a) 200 kHz, 60
(b) 1 MHz, 60
(c) 3 MHz, 12
(d) 4 MHz, 12

***Solution: (a)***

In cascade the resulting mode of counter $= M \times N \times O = 3 \times 4 \times 5$

$$\text{Lowest output frequency} = \frac{12 \times 10^6}{3 \times 4 \times 5} = 200 \times 10^3 = 200\text{ kHz}$$

$$\text{Overall modulus} = 3 \times 4 \times 5 = 60$$

### 7.2.2 Synchronous Series Carry Counter

- A 4-bit (MOD-16) series carry counter is shown in *Figure* 7.8 (a). The basic principle of operation of this synchronous counter is inputs of the flip-flops are connected in such a manner that only those FFs that are supposed to toggle on a given CLK will have $T = 1$.

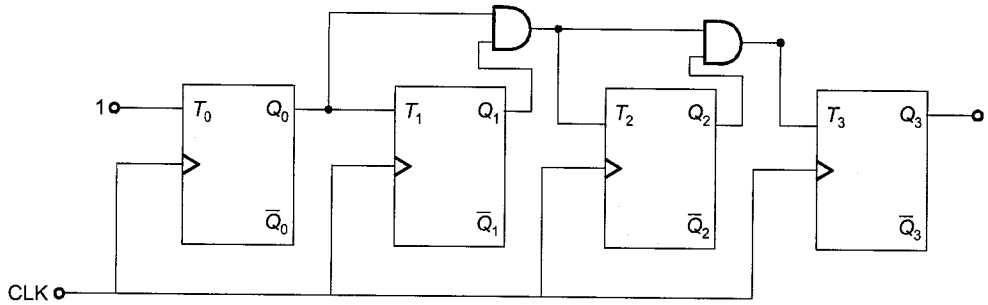



***Figure-7.8 (a)*** *Series carry counter*

- The circuit shown above is a synchronous counter. In this counter
  1. $Q_0$ toggles for every CLK pulse is applied.
  2. $Q_1$ toggles when $Q_0 = 1$ and CLK pulse is applied.
  3. $Q_2$ toggles when $Q_1 = 1$ and $Q_0 = 1$ and CLK pulse is applied.
  4. $Q_3$ toggles when $Q_2 = Q_1 = Q_0 = 1$ and CLK pulse is applied.
- The truth table, depending upon the operation is shown in *Table* 7.4.

**Table-7.4** *Truth table for series carry count*

| CLK | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 |
| 16 | 0 | 0 | 0 | 0 |

- In order to get synchronous down counter, $\bar{Q}$ output of each stage is connected to the input of the next stage.
- Most important advantage of this counter is that it reduces the 'decoding error'.
- The total delay for synchronous series counter is

$$\boxed{T_{CLK} \geq t_{pd} + (n-2)\, t_{pd} \text{ (AND-Gate)}}$$

As the number of bit increases, the propagation delay of FFs and the propagation delay of AND gate also increases, this will increase the total delay of the circuit. Thus, in order to overcome this difficulty synchronous parallel carry counter is used.

For synchronous parallel carry counter.

$$\boxed{T_{CLK} \geq t_{pd} + t_{pd} \text{ (AND-Gate)}}$$

***Remember:*** Speed of synchronous parallel carry counter > speed of synchronous series carry counter > speed of ripple carry counter.

**Example-7.9** (*i*) Determine $f_{max}$, if $t_{pd}$ for each FF is 50ns and $t_{pd}$ for each AND gate is 20ns. Compare this with $f_{max}$ for a MOD-16 ripple counter.

(*ii*) What has to be done to convert this counter to MOD-37?

(*iii*) Determine $f_{max}$ for the MOD-32 parallel counter.

***Solution:***

(*i*) The total delay that must be allowed between input clock pulses is equal to FF $t_{pd}$ + AND gate $t_{pd}$. Thus, $T_{clock} \geq 50 + 20 = 70$ns, and so the parallel counter has

$$f_{max} = \frac{1}{70\text{ns}} = 14.3 \text{ MHz (parallel counter)}$$

A MOD-16 ripple counter uses four FFs $t_{pd} = 50$ns. Thus, $f_{max}$ for the ripple counter is

$$f_{max} = \frac{1}{4 \times 50\text{ns}} = 5 \text{ MHz (ripple counter)}$$

(*ii*) A fifth FF must be added, since $2^5 = 32$. The CLK input of this FF is also tied to the input pulses. Its *J* and *K* inputs are fed by the output of a four-input AND gate whose inputs are *A*, *B*, *C* and *D*.

(*iii*) $f_{max}$ is still determined as in (a) regardless of the number of FFs in the parallel counter. Thus, $f_{max}$ is still 14.3 MHz.

## 7.3 Synchronous Counter Design

Synchronous counters for any given count sequence and modulus can be designed in the following way.

- Identify the number of flip-flops, number of inputs and number of outputs required.
- Construct the state table with the help of present state, next state and excitation table of the FFs.
- Prepare K-map for each flip-flop input in terms of flip-flop outputs as the input variable, obtain and minimize the logical expressions.
- Connect the circuit using flip-flops and other gates corresponding to the minimized expressions.

**Example-7.10** Design a 3-bit synchronous counter using J-K Flip-flops.

***Solution:***

The number of Flip-flops required is 3. Let the flip-flops be $FF_0$, $FF_1$ and $FF_2$ and their inputs and outputs are given below:

| Flip-flop | Input | Output |
|---|---|---|
| $FF_0$ | $J_0, K_0$ | $Q_0$ |
| $FF_1$ | $J_1, K_1$ | $Q_1$ |
| $FF_2$ | $J_2, K_2$ | $Q_2$ |

| Present state | | | Next state | | | Flip-flop inputs | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $FF_0$ | | $FF_1$ | | $FF_2$ | |
| $Q_2$ | $Q_1$ | $Q_0$ | $Q'_2$ | $Q'_1$ | $Q'_0$ | $J_0$ | $K_0$ | $J_1$ | $K_1$ | $J_2$ | $K_2$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | x | 0 | x | 0 | x |
| 0 | 0 | 1 | 0 | 1 | 0 | x | 1 | 1 | x | 0 | x |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | x | x | 0 | 0 | x |
| 0 | 1 | 1 | 1 | 0 | 0 | x | 1 | x | 1 | 1 | x |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | x | 0 | x | x | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | x | 1 | 1 | x | x | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | x | x | 0 | x | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 | x | 1 |
| 0 | 0 | 0 | | | | | | | | | |

The count sequence and the required inputs of flip-flops are given in table. The inputs to the flip-flops are determined by using K-MAP.

| $Q_0$ \ $Q_2Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 |
| 1 | x | x | x | x |

$J_0 = 1$

| $Q_0$ \ $Q_2Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | x | x | x | x |
| 1 | 1 | 1 | 1 | 1 |

$K_0 = 1$

| $Q_0$ \ $Q_2Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | x | x | 0 |
| 1 | 1 | x | x | 1 |

$J_1 = Q_0$

| $Q_0$ \ $Q_2Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | x | 0 | 0 | x |
| 1 | x | 1 | 1 | x |

$K_1 = Q_0$

| $Q_0$ \ $Q_2Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | x | 0 |
| 1 | 0 | 1 | x | x |

$J_2 = Q_0Q_1$

| $Q_0$ \ $Q_2Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | x | x | 0 | 0 |
| 1 | x | x | 1 | 0 |

$K_2 = Q_0Q_1$

Consider one column of the counter state at a time and start from the first row.

⇒ We consider $Q_0$. Before the first pulse is applied, $Q_0 = 0$ and it is required to be 1 at the end of the first clock pulse.

⇒ Therefore, to achieve this condition, the values of $J_0$ and $K_0$ are 1 and x respectively (from the excitation table). These are entered in the table in the row corresponding to 0 pulse.

⇒ When the second clock pulse is applied $Q_0$ is to change from 1 to 0, therefore, the required inputs are

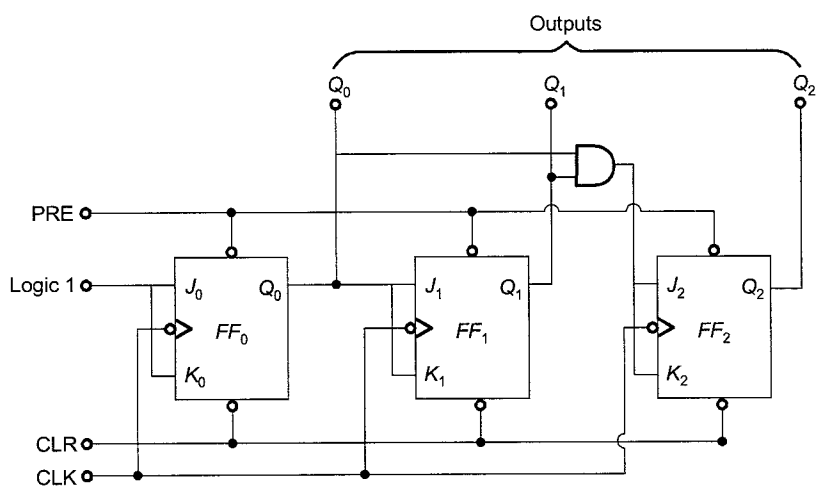
$$J_0 = x,\ K_0 = 1$$

⇒ In a similar manner inputs of each flip-flop are determined.

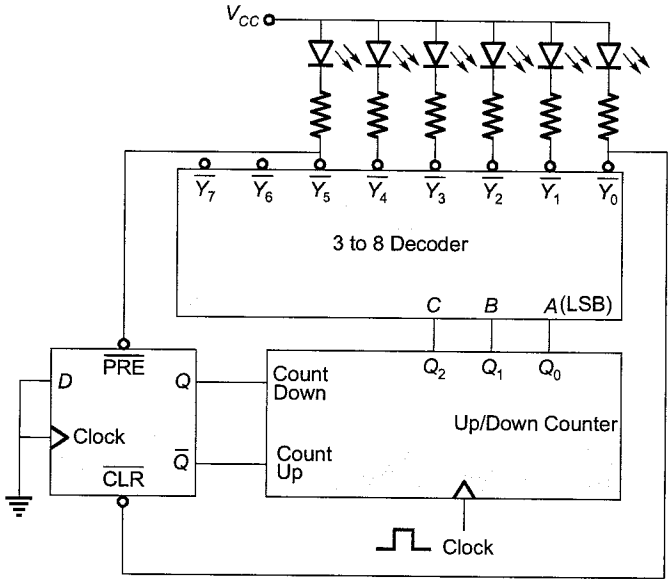
⇒ Now, we prepare the $K$-maps with $Q_2$, $Q_1$ and $Q_0$ as input variables and flip-flop inputs are output variables as:

$$J_0 = 1, \qquad K_0 = 1$$
$$J_1 = Q_0, \qquad K_1 = Q_0$$
$$J_2 = Q_0\,Q_1, \qquad K_2 = Q_0\,Q_1$$

Now the final circuit of 3-bit synchronous counter is,



**Example-7.11** The circuit below shows an up/down counter working with a decoder and a flip-flop. Preset (PRE) and Clear (CLR) of the flip-flop are asynchronous active-low inputs.




Assuming that the initial value of counter output ($Q_2$ $Q_1$ $Q_0$) as zero, the counter outputs in decimal for 12 clock cycles are (1 → Up, 0 → Down)

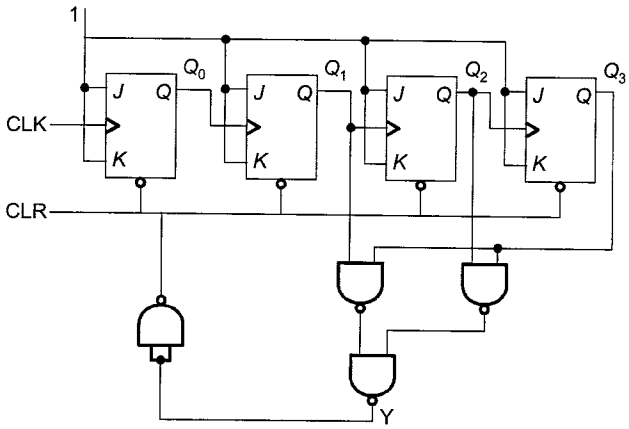
(a) 0, 1, 2, 3, 4, 4, 3, 2, 1, 1, 2, 3, 4
(b) 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0
(c) 0, 1, 2, 3, 4, 5, 5, 4, 3, 2, 1, 0, 1
(d) 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0, 1, 2

***Solution: (d)***

Initially $Q = 0$ and count up ($\overline{Q} = 1$) is active so it started counting up and when it reaches to 5 then decoder output at pin 5 becomes 0 and preset will be active and it will set $Q$ and it will make the counter mode down and count becomes 4, then 3 then 2 then 1 then 0, as soon as it reaches 0, decoder output at pin 0 is low and clear is active and $Q$ goes to 0 and $\overline{Q} = 1$ so up is active and it counts 1, 2,.....

so sequence is 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0, 1, 2.

**Example-7.12** Consider the counter circuit shown below:



(*i*) In the above figure, $Y$ can be expressed as

(a) $Q_3(Q_2 + Q_1)$ (b) $Q_3 + Q_2 Q_1$

(c) $\overline{Q_3(Q_2 + Q_1)}$ (d) $\overline{Q_3 + Q_2 Q_1}$

(*ii*) The above circuit is a

(a) Mod-8 counter (b) Mod-9 counter

(c) Mod-10 counter (d) Mod-11 counter

***Solution:***

(*i*) (a)

$$Y = \overline{(\overline{Q_1 Q_3}) \cdot (\overline{Q_2 Q_3})} = \overline{(\overline{Q_1 Q_3})} + \overline{(\overline{Q_2 Q_3})} = Q_1 Q_3 + Q_2 Q_3 = Q_3(Q_1 + Q_2)$$

(*ii*) (c)

To reset the counter output $Y$ must be one.

$$Y = Q_3(Q_1 + Q_2)$$

| $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

When the counter output $Q_3 Q_2 Q_1 Q_0 = 1010$ then the output Y will give output 1 and it will be given to inverter, so it will reset the counter and again counting will start and it will not go for further combinations. So, MOD of the counter is '10'.

## 7.4 State Diagram and State Table

- A state diagram is a pictorial representation of the relationship between the present state, the input, the next state, and the output of a sequential circuit. However, for the implementation of sequential circuit, the information contained in the state diagram is to be translated into state table. Thus, the state table is a tabular representation of the state diagram.
- Consider the sequential circuit shown in *Figure 7.10 (a)*.

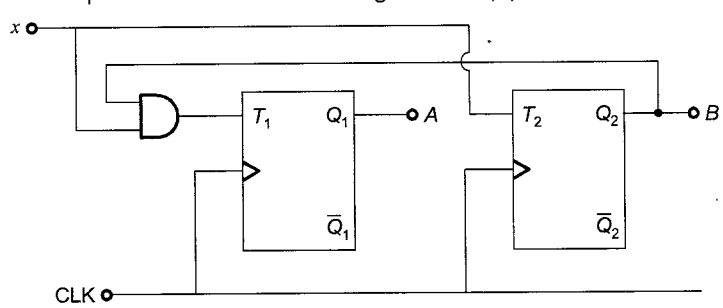



***Figure-7.10 (a)*** *Logic diagram*

The circuit consists of two T FFs with external input $x$. A set of next state equation can be written as

$$(Q_{n+1})_1 = Q_1 \oplus T_1 = Q_1 \oplus xQ_2$$

$$(Q_{n+1})_2 = Q_2 \oplus T_2 = Q_2 \oplus x$$

- The corresponding state table is shown *Table 7.5*.

**Table-7.5** *State table*

| Present State | | Input | Next state | | Required excitation | |
|---|---|---|---|---|---|---|
| $Q_1$ | $Q_2$ | $x$ | $(Q_{n+1})_1$ | $(Q_{n+1})_2$ | $T_1$ | $T_2$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 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 |

- From the state table, the state diagram may be drawn as shown in *Figure 7.10 (b)*.

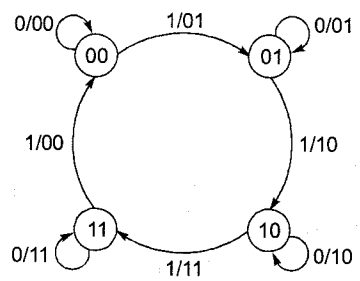



***Figure-7.10 (b)*** *State diagram*

- Here, each encircled binary number represents a particular state of the flip-flop outputs. Each directed line which indicates the transition between the two state is labelled with two binary numbers separated by a 'slash'. The first number represent the input during present state and the second number represents the next state.

## 7.4.1 State Diagram of Various Flip-Flops

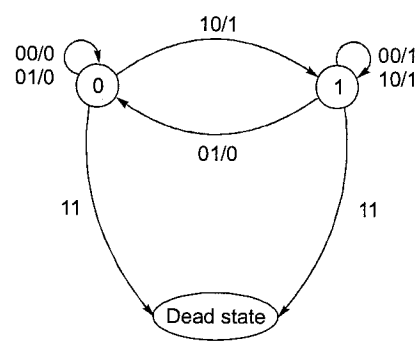
### S-R Flip-Flop

The state table and state diagram of the SR flip-flop are shown in *Figure 7.11 (a), and (b)* respectively. For SR flip-flop the next state equation is

$$Q_{n+1} = S + \bar{R}Q_n$$

| Present State | Input | | Next state |
|---|---|---|---|
| $Q_n$ | $S$ | $R$ | $(Q_{n+1})_1$ |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | x |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | x |

(a) State Table




(b) State Diagram

***Figure-7.11*** *(a) and (b)*

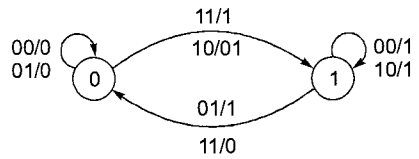
### J-K Flip-Flop

The state table and state diagram for JK flip-flop shown in *Figure 7.12 (a) and (b)* respectively. For JK flip-flop the next state equation is

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

| Present State | Input | | Next state |
|---|---|---|---|
| $Q_n$ | $J$ | $K$ | $(Q_{n+1})_1$ |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

(a) State Table




(b) State Diagram

***Figure-7.12*** *(a) and (b)*

### D Flip-Flop

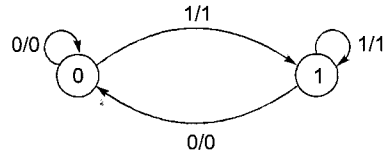
The state table and state diagram for *D* flip-flop are shown in *Figure 7.13 (a)* and (b) respectively.

For D flip-flop the next state equation is

$$Q_{n+1} = D$$

| Present State | Input | Next state |
|---|---|---|
| $Q_n$ | $D$ | $Q_{n+1}$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(a) State Table




(b) State Diagram

***Figure-7.13*** *(a) and (b)*

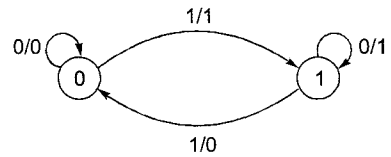
### T Flip-Flop

The state table and state diagram for T flip-flop are shown in *Figure 7.14 (a) and (b)* respectively. For *T* flip-flop the next state equation is

$$Q_{n+1} = Q_n \oplus T$$

| Present State | Input | Next state |
|---|---|---|
| $Q_n$ | $T$ | $Q_{n+1}$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(a) State Table




(b) State Diagram

***Figure-7.14*** *(a) and (b)*

## 7.5 Finite State Model/Machine

- The finite state machine is an abstract model that describes the synchronous sequential machine.
- In a sequential circuit, the output depends upon the present input as well as past history and therefore it would need an infinite capacity for storing them. Since it is impossible to implement machines with infinite storage capacity, finite state machines are used. Thus finite state machines are sequential circuits whose past history can affect their future behaviour in only a finite number of ways.
- That means finite state machines are machines with a fixed number of states. Every finite state machine has a finite number of memory devices.
- The block diagram of a finite state model is shown in *Figure 7.15*. Here $x_1, x_2 \cdots x_n$ are the inputs $y_1, y_2 \cdots y_m$ are the outputs, $z_1, z_2 \cdots z_K$ are state variables and $Z_1, Z_2 \cdots Z_K$ represents the next state.

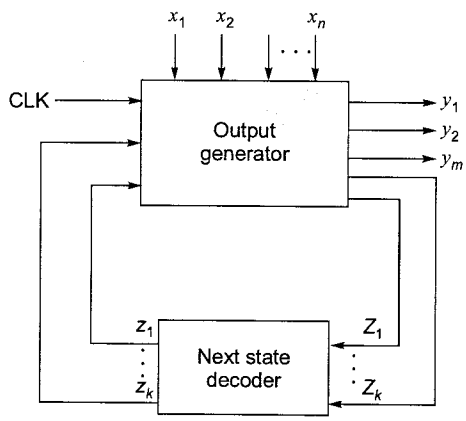



***Figure-7.15*** *Block diagram of a state model*

- With an *n*-state machine, we can generate a periodic sequence of less than or equal to *n*-states.
- Finite machines are of two types. The major difference between them is the way the output is generated. They are
  (*i*) Moore type model.
  (*ii*) Mealy type model.
- *Table (7.6)* shows a comparison between the Moore machine and the mealy machine.

**Table-7.6** *Comparison between Moore and Mealy model*

| Moore Machine | Mealy Machine |
|---|---|
| 1. The output is a function of present state only. | 1. The output is a function of present state as well as present input. |
| 2. The change in input does not affect the output of the circuit. | 2. The change in input may affect the output of the circuit. |
| 3. For implementing same function, it requires more number of states. | 3. It requires less number of states for implementing same function. |
| 4. The designing of Moore model is easy. | 4. The designing of Mealy model is complex. |

**Example-7.13** The output of a clocked sequential circuit is independent of the input. The circuit can be represented by

(a) Mealy model
(b) Moore model
(c) Either Mealy or Moore model
(d) Neither Mealy nor Moore model

***Solution: (b)***

## Summary

- The ring counter and Johnson counter are not use for counting purpose.
- In ring counter the phase shift between the generated waveform is $\frac{360°}{n}$, where '$n$' is the number of bits.
- "Lock out" problem occurs in non-binary counter.
- $J$, $K$, $S$, $R$, $D$ and $T$ are called synchronous input.
- Reset and Set are called asynchronous input.
- "Preset" always make the output to '1'.
- "Clear" always make the output to '0'.
- "Glitch" is an unwanted spike in the signal.
- For a MOD-10 Counter (DECADE), Johnson counter uses 5 FFs, ring counter uses 10 FFs and ripple counter uses 4 FFs.
- Synchronous counter is used for "High Frequency Application" and Asynchronous counter is used for "Low Frequency Application".
- MOD-16 counter could also be called a "divide-by-16 counter".
- Johnson counter represents a middle ground between ring counter and binary counter.
- A "ring counter" is a circular shift register with only one FF being set at any particular time; all others are cleared.
- Minimum number of J-K-FFs needed to construct a BCD counter is 4.
- The overall MOD of a counter with MOD M is connected in cascade with another counter of MOD $N$ is $(M \times N)$.
- In "MOORE state machines" present output depends only on present state.
- In "MEALY state machines" present output depends on present state and present external inputs.

## Student's Assignments 1

**Q.1** What is the delay time of 14 stage ripple counter and that of $N$ stage synchronous counter? Draw the logic diagram of a counter which can count both in up and down directions.

**Q.2** Draw the state diagram, logic diagram and timing diagram for a decimal BCD ripple counter. Explain the operation mentioning the conditions for state transition of each flip-flop.

**Q.3** A flip-flop has a 5 ns delay from the time the clock edge occurs to the time the output is complemented. What will be the maximum delay in 10 bit binary ripple counter that uses this FF. What is the maximum frequency at which the counter can operate reliably?

**Q.4** How many flip-flop will be complemented in a 10 bit binary ripple counter to reach the next count after 1111000111?

***Answer : (Conventional)***

**3.** (50 ns; 20 MHz)

**4.** 9

## Student's Assignments 2

**Q.1** The initial state of a Mod-20 counter is 10101. Counter design is in such a way that output of the logic gate (AND) is connected with preset of all the 5 FFs and FFs are triggered with (–)ve edge and clock connection is with complement of the FF output ($Q$). After 44 clock pulses, the state of the counter will be

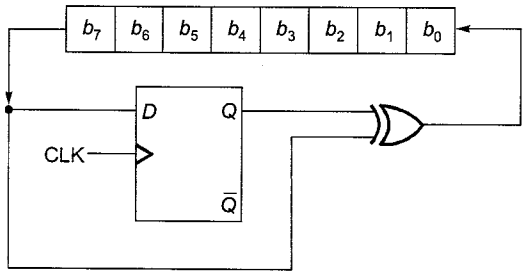
(a) 11001 (b) 10010
(c) 11000 (d) 10001

**Q.2** The state table of a latch with inputs $L$ and $M$ is given below. The expression for the next state $Q^+$ is

| $L$ | $M$ | $Q$ | $Q^+$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

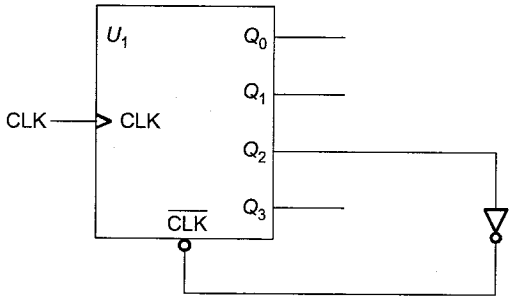
(a) $\bar{L}M + MQ$ (b) $L\bar{M} + LQ$
(c) $\bar{L}M + M\bar{Q}$ (d) $L\bar{M} + L\bar{Q}$

**Q.3** The 8-bit shift-left shift register and D flip-flop shown in the figure is synchronized with the same clock. The D flip-flop is initially cleared. The circuit acts as




(a) Binary to 2's complement converter
(b) Binary to Gray code converter
(c) Binary to 1's complement converter
(d) Binary to Excess - 3 Code converter

**Q.4** In figure $U_1$ is a 4-bit binary synchronous counter with synchronous clear. $Q_0$ is the LSB and $Q_3$ is the MSB of output
Circuit shown represents a

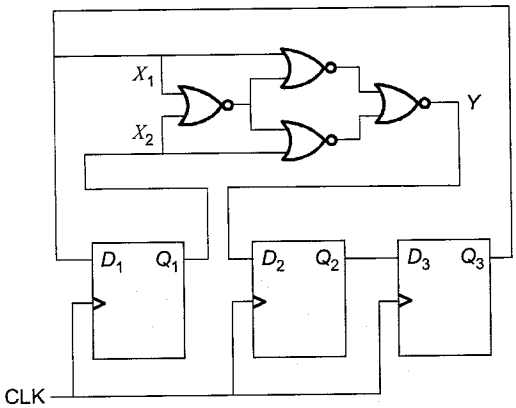



(a) MOD-2 Counter (b) MOD-3 Counter
(c) MOD-4 Counter (d) MOD-5 Counter

**Q.5** A certain J-K-FF has propagation delay ($t_{pd}$) = 12 ns. The largest Mod counter that can be designed from these FFs which still operate up to 10 MHz will be

(a) 8 (b) 9
(c) 10 (d) 256

**Q.6** For the circuit shown below:



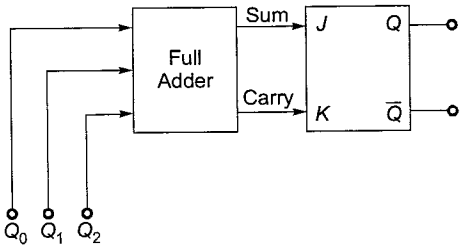

(*i*) The correct input-output relationship between $Y$ and ($X_1$, $X_2$) is

(a) $Y = X_1 + X_2$ (b) $Y = X_1 X_2$
(c) $Y = X_1 \oplus X_2$ (d) $Y = \overline{X_1 \oplus X_2}$

(*ii*) The $D$ flip-flops are initialized to $Q_1\ Q_2\ Q_3$ = 000. After 1 clock cycle, $Q_1\ Q_2\ Q_3$ is equal to

(a) 011 (b) 010
(c) 100 (d) 101

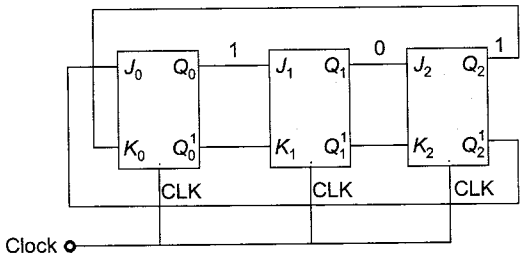
**Q.7** Just before the flip-flops of the counter are cleared (that is, when the counter is in the last used state), the flip flop outputs are made to pass through a full adder. The output of the full adder is then passed through a JK flip flop as shown below.




The JK flip flop enters into the

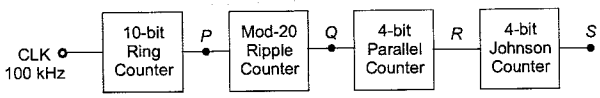
(a) Hold state (b) Set state
(c) Reset state (d) Toggle state

**Q.8** The three state Johnson-ring counter as shown above is clocked at constant frequency of $f_c$ from the starting state of $Q_0\ Q_1\ Q_2$ = 101. The frequency of outputs $Q_0\ Q_1\ Q_2$ will be




(a) $\frac{f_c}{8}$ (b) $\frac{f_c}{6}$
(c) $\frac{f_c}{3}$ (d) $\frac{f_c}{2}$

**Q.9** Consider a cascaded network having 4-sequential circuit block with input clock frequency is 100 kHz. The frequencies of the pulses at the points $P$, $Q$, $R$ and $S$ respectively are




(a) 10 kHz, 0.5 kHz, 31.25 kHz, 390 Hz
(b) 10 kHz, 5 kHz, 1.25 kHz, 390 Hz
(c) 10 kHz, 0.5 kHz, 31.25 Hz, 7.8 Hz
(d) 10 kHz, 0.5 kHz, 31.25 kHz, 3.9 Hz

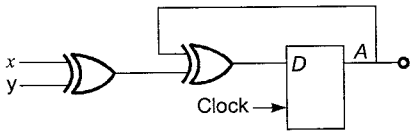
**Q.10** Which of the following measurements can be done using a counter?

1. Pulse duration
2. Interval between two pulses
3. Amplitude of the pulse
4. Rise time of a pulse

Select the correct answer from the codes given below:

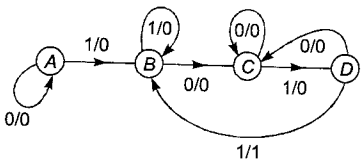
(a) 1 and 2 (b) 2 and 3
(c) 1 and 4 (d) 2 and 4

**Q.11** A finite state machine is given below. Here $x$, $y$ are inputs and '$A$' is the output. The FSM is



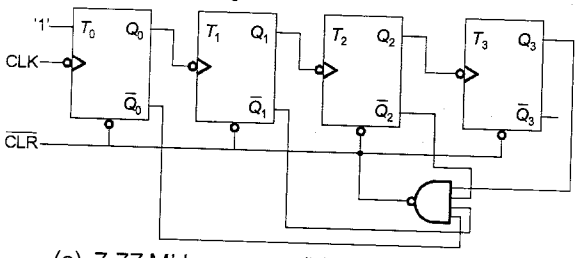
(a) a Moore machine with $A(t+1) = D$
(b) a Mealey machine with $A(t+1) = A \oplus x \oplus y$
(c) neither moore nor mealey machine
(d) a logarithmic state machine

**Q.12** Which of the following is correct regarding the above transition diagram?




(a) detects the sequence 1010 and is overlapping detector.
(b) detects the sequence 1011 and is non-overlapping detector.
(c) detects the sequence 1011 and is overlapping detector.
(d) detects the sequence 1010 and is non-overlapping.

**Q.13** If the clock frequency is 70 MHz, then the output frequency at $Q_3$ of the above counter is



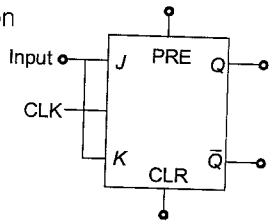

(a) 7.77 MHz (b) 11.66 MHz
(c) 10 MHz (d) 8.75 MHz

**Q.14** Consider one set of Mod-14 counter having input clock frequency 19 kHz. If another input clock frequency as first one is cascaded then final output clock frequency of the composite counter will be

(a) 1 kHz (b) 4.75 kHz
(c) 0.28 kHz (d) 2.1 kHz

**Q.15** The following configuration provides

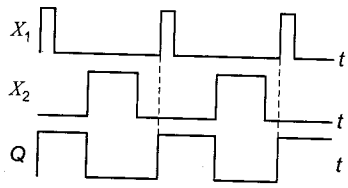
(a) Master slave action
(b) Delay gate
(c) Toggle switch
(d) None of these

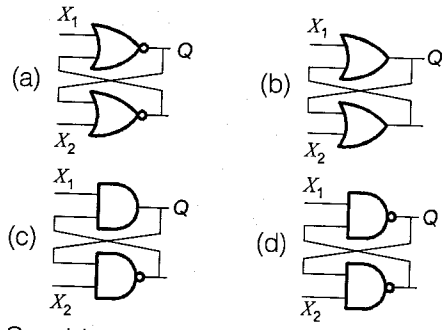



**Q.16** In a non-binary ripple counter, by using negative edge-triggered Toggled FFs, the input clock pulse is applied to

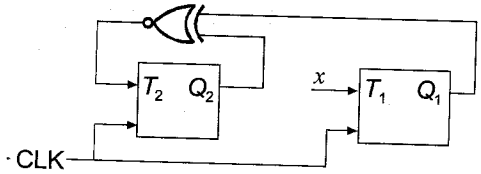
(a) $T$-inputs of all FFs
(b) $T$-inputs of one FF
(c) Clock input of one FF
(d) Clock input of all FFs

**Q.17** Select the circuit which will produce the given output Q for the input signals $X_1$ and $X_2$ given in the figure




**Q.18** Consider the partial implementation of a-2 bit counter using using T flip-flops following the sequence 0–2–3–1–0, as shown below:




To complete the circuit the input x should be

(a) $Q_2'$ (b) $Q_2 + Q_1$
(c) $(Q_1 \oplus Q_2)'$ (d) $Q_1 \oplus Q_2$

***Answer Key:***

| | | | | |
|---|---|---|---|---|
| **1.** (d) | **2.** (d) | **3.** (b) | **4.** (c) | **5.** (d) |
| **6.** (*i*) (d) & (*ii*) (b) | | **7.** (b) | **8.** (b) | **9.** (d) |
| **10.** (a) | **11.** (a) | **12.** (c) | **13.** (d) | **14.** (c) |
| **15.** (c) | **16.** (c) | **17.** (a) | **18.** (d) | |