Revised Edition

A Textbook of Digital Electronics

Dr. R.S. SEDHA

S. CHAND

A TEXTBOOK OF

DIGITAL ELECTRONICS

A TEXTBOOK OF

DIGITAL ELECTRONICS

[For the students of B.E./B.Sc.(Engg.) / B.Tech. in Electronics, Communication, Computer Science, Information Technology, Grade IETE, AMIE(I), B.Sc. of all Indian Universities, UPSC Engineering/Civil Services and Other Competitive Examinations]

Dr. R.S. SEDHA
Program Chair
B.E. (Hons), M. Tech. (IIT Delhi),
Ph.D. (U.K.), FIETE, MIEEE
School of Engineering
Republic Polytechnic
Tanglin Campus
1 Kay Siang Road
SINGAPORE–248922

S. CHAND & COMPANY LTD.
(AN ISO 9001 : 2000 COMPANY)
RAM NAGAR, NEW DELHI - 110 055

S. CHAND & COMPANY LTD.
(An ISO 9001 : 2000 Company)
Head Office: 7361, RAM NAGAR, NEW DELHI - 110 055
Phone: 23672080-81-82, 9899107446, 9911310888
Fax: 91-11-23677446
Shop at: **schandgroup.com;** e-mail: **info@schandgroup.com**

Branches :

AHMEDABAD : 1st Floor, Heritage, Near Gujarat Vidhyapeeth, Ashram Road, **Ahmedabad** - 380 014,
Ph: 27541965, 27542369, ahmedabad@schandgroup.com

BANGALORE : No. 6, Ahuja Chambers, 1st Cross, Kumara Krupa Road, **Bangalore** - 560 001,
Ph: 22268048, 22354008, bangalore@schandgroup.com

BHOPAL : 238-A, M.P. Nagar, Zone 1, **Bhopal** - 462 011, Ph: 4274723. bhopal@schandgroup.com

CHANDIGARH : S.C.O. 2419-20, First Floor, Sector - 22-C (Near Aroma Hotel), **Chandigarh** -160 022,
Ph: 2725443, 2725446, chandigarh@schandgroup.com

CHENNAI : 152, Anna Salai, **Chennai** - 600 002, Ph: 28460026, 28460027, chennai@schandgroup.com

COIMBATORE : Plot No. 5, Rajalakshmi Nagar, Peelamedu, **Coimbatore** -641 004, (M) 09444228242,
coimbatore@schandgroup.com

CUTTACK : 1st Floor, Bhartia Tower, Badambadi, **Cuttack** - 753 009, Ph: 2332580; 2332581,
cuttack@schandgroup.com

DEHRADUN : 1st Floor, 20, New Road, Near Dwarka Store, **Dehradun** - 248 001,
Ph: 2711101, 2710861, dehradun@schandgroup.com

GUWAHATI : Pan Bazar, **Guwahati** - 781 001, Ph: 2738811, 2735640 guwahati@schandgroup.com

HYDERABAD : Padma Plaza, H.No. 3-4-630, Opp. Ratna College, Narayanaguda, **Hyderabad** - 500 029,
Ph: 24651135, 24744815, hyderabad@schandgroup.com

JAIPUR : A-14, Janta Store Shopping Complex, University Marg, Bapu Nagar, **Jaipur** - 302 015,
Ph: 2719126, jaipur@schandgroup.com

JALANDHAR : Mai Hiran Gate, **Jalandhar** - 144 008, Ph: 2401630, 5000630, jalandhar@schandgroup.com

JAMMU : 67/B, B-Block, Gandhi Nagar, **Jammu** - 180 004, (M) 09878651464

KOCHI : Kachapilly Square, Mullassery Canal Road, Ernakulam, **Kochi** - 682 011, Ph: 2378207,
cochin@schandgroup.com

KOLKATA : 285/J, Bipin Bihari Ganguli Street, **Kolkata** - 700 012, Ph: 22367459, 22373914,
kolkata@schandgroup.com

LUCKNOW : Mahabeer Market, 25 Gwynne Road, Aminabad, **Lucknow** - 226 018, Ph: 2626801, 2284815,
lucknow@schandgroup.com

MUMBAI : Blackie House, 103/5, Walchand Hirachand Marg, Opp. G.P.O., **Mumbai** - 400 001,
Ph: 22690881, 22610885, mumbai@schandgroup.com

NAGPUR : Karnal Bag, Model Mill Chowk, Umrer Road, **Nagpur** - 440 032, Ph: 2723901, 2777666
nagpur@schandgroup.com

PATNA : 104, Citicentre Ashok, Govind Mitra Road, **Patna** - 800 004, Ph: 2300489, 2302100,
patna@schandgroup.com

PUNE : 291/1, Ganesh Gayatri Complex, 1st Floor, Somwarpeth, Near Jain Mandir,
Pune - 411 011, Ph: 64017298, pune@schandgroup.com

RAIPUR : Kailash Residency, Plot No. 4B, Bottle House Road, Shankar Nagar, **Raipur -** 492 007,
Ph: 09981200834, raipur@schandgroup.com

RANCHI : Flat No. 104, Sri Draupadi Smriti Apartments, East of Jaipal Singh Stadium, Neel Ratan Street,
Upper Bazar, **Ranchi** - 834 001, Ph: 2208761, ranchi@schandgroup.com

VISAKHAPATNAM: Plot No. 7, 1st Floor, Allipuram Extension, Opp. Radhakrishna Towers, Seethammadhara
North Extn., **Visakhapatnam** - 530 013, (M) 09347580841, visakhapatnam@schandgroup.com

First Edition 2004
Reprints 2008, 2009
Revised Enlarged Edition, 2010

ISBN : 81-219-2378-6 **Code :** 10A 260

By Rajendra Ravindra Printers Pvt. Ltd., 7361, Ram Nagar, New Delhi -110 055
and published by S. Chand & Company Ltd., 7361, Ram Nagar, New Delhi -110 055.

PREFACE TO THE SECOND EDITION

I feel extremely happy presenting the revised edition of this book to the students and faculty members of the universities, engineering colleges in India and abroad.

The present book has been thoroughly revised and 5 new chapters have been added to widen the scope of the book. These new chapters are:

1. D/A and A/D Converters,
2. Semiconductor Memories,
3. Programmable Logic Devices,
4. Fundamentals of Microprocessors,
5. Digital System Design using VHDL.

The chapter on **Fundamentals of Microprocessors** has been added because the 'brains' behind the most high-level digital systems is the microprocessors. The basic understanding of microprocessor software and hardware is imperative for the electronics engineer to design and troubleshoot digital systems. A new chapter on **Digital System Design using VHDL** has been added knowing the fact that with the advance of the semiconductor and communication industries, the use of **System-on-Chip** (SoC) has become an essential technique to decrease product costs. As a result, it is important for electronic engineers to develop a good understanding of the key stages of hardware description language (HDL) design flow based on cell based libraries of field-programmable gate array (FPGA) devices.

I will take this opportunity to share with faculty members about some teaching strategies that I have developed over the past 30 years. We all know that the students these days, have become very excited about learning digital electronics because of the expanding job opportunities for digital engineers. Students are also attracted to this subject area because of the availability of inexpensive digital IC's and CPLDS/FPGAs. This helps them to build digital circuits in the lab or at home at a lower cost. As a faculty member, we should give the students a greater opportunity for hands-on laboratory experience. An important feature of this book is that it gives a lot of information to build the circuits. The students should be encouraged to build circuits such as 5-V power supply, 50-Hz pulse generator, the cross-NAND switch debouncer, a digital clock with a seven segment LED display for displaying hours, minutes and seconds. The students should also be encouraged to do simulations using ORCAD, MultiSIM tools. These tools help the student to create a schematic of given circuit faster and test the functionality of the circuit.

I would like to share some thoughts with the students as well. As you know digital electronics is the foundation course for computers and microprocessor / microcontroller-based systems found in notebooks, mobile phones, home-entertainment systems, Microsoft X-box, Nintendo game systems, automobiles, industrial control systems, and many more. You are beginning your study of digital electronics at a good time. Technological advances made during the last 50 years have provided us with IC's that can perform complex tasks. Before you are finished studying this book, you will be designing, fabricating and testing the exciting circuits. The study of digital electronics also provides you the prerequisites for future studies in microprocessor and microcontroller based systems. The subject provides the job skills for you to become a computer technician, production test technician, or digital design technician or to fill any other positions related to computer and microprocessor / microcontroller-based systems. This book is written as a learning tool in simple language. There are several solved examples. At the end of every chapter, there are review questions and multiple choice questions. Work-out the solution of the unsolved problems to build up your understanding further. Take advantage of the simulation tools from Cadence, MultiSIM, Altera and Xilinx.

I wish to express my sincere thanks to my numerous fellow colleagues and students, both at home and abroad for patronizing this book and recommending it to their students and friends. I hope they will continue to patronize this book in the future also.

Any errors, omissions and suggestions for the improvements of this book, brought to my notice, will be thankfully acknowledged and incorporated in the next edition.

Singapore–248 922
DID : (65) 63768303
Fax : (65) 64151310
Website : http://www.rp.edu.sg

Dr. R.S. SEDHA

CONTENTS

1

INTRODUCTION

OUTLINE

Objectives

Upon completion of this chapter, you should be able to:

- Distinguish between analog and digital representations.
- Know advantages and limitations of digital techniques compared with analog.
- Understand the need for analog-to-digital converters (ADCs) and digital-to-analog converters (DACs).
- Show how voltage levels are used to represent digital quantities.
- Describe various parameters of pulse waveform such as, rise time, fall time, pulse width, frequency, period and duty cycle.
- Identify a timing diagram.
- Describe the major parts of a computer and understand their functions.
- Distinguish among microcomputers, microprocessors and microcontrollers.

1-1. Introduction

The term *digital* is derived from the way the computers perform operations, by counting digits. For many years, applications of digital electronics were confined to computer systems. In today's world, the term digital has become part of our everyday vocabulary because of the dramatic way that digital circuits and digital techniques have become so widely used in almost all areas of life: computers, automated machine control, robots, energy monitoring and control, inventory management, medical science and technology, transportation, entertainment and space exploration.

In a modern home, digital circuitry controls the appliances, alarm systems and heating/cooling systems. Under the control of digital circuitry and microprocessors, newer automobiles have added safety features, are more energy efficient, and are easier to diagnose and correct when malfunction arise. Effective control of heating, ventilating and air-conditioning can reduce energy bills significantly. More and more grocery stores are using the universal product code (UPC) to check out the total sale of grocery orders, as well as to control inventory and replenish stock automatically. The area of medical electronics uses digital thermometers, life-support systems and monitors. The digital electronics is used extensively in reproduction of music. Digital reproduction is less susceptible to electrostatic noise and therefore can reproduce music with greater *fidelity*.

Digital electronics evolved from the principle that transistor circuitry could easily be designed and fabricated to output one of the two voltage levels based on the levels placed at its inputs. The two distinct levels (usually 5 V and 0 V) are HIGH and LOW and can be represented by 1 and 0. Now we will begin our exciting educational journey in which we will discover the fundamental principles, concepts and operations that are common to all digital systems.

1-2. Numerical Representation

As a matter of fact, in science, technology, business and in all other fields of endeavor, we are constantly dealing with quantities. Quantities are measured, monitored, recorded, manipulated arithmetically, or in some other way utilized in most physical systems. It is extremely important, when dealing with various quanitities that we be able to represent their values efficiently and accurately.

Basically there are two ways of representing the numerical values of quantities namely **analog** and **digital.** Now we will discuss these quantities one by one in the following pages.

1-3. Analog Representation

The numerical representation in which a quantity is represented by a voltage, current or meter movement that is proportional to the value of that quantity is called **analog representation**. An example of an analog quantity is an automobile speedometer, in which the deflection of the needle is proportional to the speed of the auto. The angular position of the needle represents the value of auto's speed and the needle follows any changes that occur as the auto speeds up or slows down.

Another example of an analog quantity is the electric iron thermostat in which the bending of the bimetallic strip is proportional to the filament temperature. As the temperature changes gradually, the curvature of the strip changes proportionally.

Still another example of an analog quantity is found in the familiar audio microphone. In this device an output voltage is generated in proportion to the amplitude of the sound waves that impinge on the microphone. The variations in the output voltage follow the same variations as the input sound.

Analog quantities such as those mentioned above have an important characteristics: *they can vary over a range of values*. The automobile speed can have any value between zero and say 100 km per hour. Similarly the microphone output might be anywhere with in the range of zero to 20 mV.

1-4. Digital Representation

The numerical representation in which a quantity is represented by the symbols called *digits* is known as digital representation . As an example, consider the digital watch, which provides the time of the day in the form of decimal digits which represent hours, minutes and seconds. As we know, the time of the day changes continuously but the digital watch reading does not change continuously; rather it changes in steps of one per second. In other words, this digital representation of the time of the day changes in discrete steps as compared with the representation of time provided by an analog watch, where the dial reading changes continuously.

Let us summarize the major difference between analog and digital quantities:

Analog = Continuous
Digital = Discrete (*i.e.*, step by step)

Example 1-1. *Which of the following are analog quantities and which are digital*?

(*a*) *number of atoms in a sample of material*

(*b*) *pressure in a bicycle tire*

(*c*) *altitude of an aircraft*

(*d*) *current through a speaker*

(*e*) *timer setting on a microwave oven*

Solution.

(*a*)	number of atoms in a sample of material:	digital — because it is a fixed value.
(*b*)	pressure in a bicycle tire:	analog — because it could vary over a range of values.
(*c*)	altitude of an aircraft:	analog — because it could vary over a range of values.
(*d*)	current through a speaker:	analog — because it is a fixed value.
(*e*)	timer setting on a microwave oven:	digital — because it is a fixed value.

1-5. Digital and Analog Systems

It will be interesting to know that a digital system is a combination of devices that are designed to manipulate logical information or physical quantities which are represented in digital form. In other words, these quantities can take only discrete values. The devices are most often electronic but they can be mechanical, magnetic or pneumatic. Some of the more familiar digital systems include digital computers, digital camera, calculators, digital audio and video equipment, and the telephone (corded and cordless and cellular phones). It may be noted carefully that the telephone system is considered as the world's largest digital system.

On the other hand, an analog system contains devices that manipulate physical quantities that are represented in analog form. In an analog, system, the quantities can vary over a continuous range of values. For example, the amplitude of the output signal to a speaker in a CD player can have any value between zero and its maximum limit. Other common analog systems are audio amplifiers, magnetic tape (audio and video) recording and playback equipment and a simple light dimmer switch.

Fig. 1-1 shows a block diagram of a public address system used to amplify a sound that can be heard by a large audience. The diagram indicates that sound waves (which are analog in nature) are picked up by a microphone and converted to a small analog voltage called the audio signal. This voltage varies continuously as the volume and frequency of the sound changes and is applied to the input of the amplifier. The output of the amplifier (which is an increased reproduction of input voltage) goes to the speaker. The speaker changes the amplified audio signal back to sound waves picked up by the microphone.

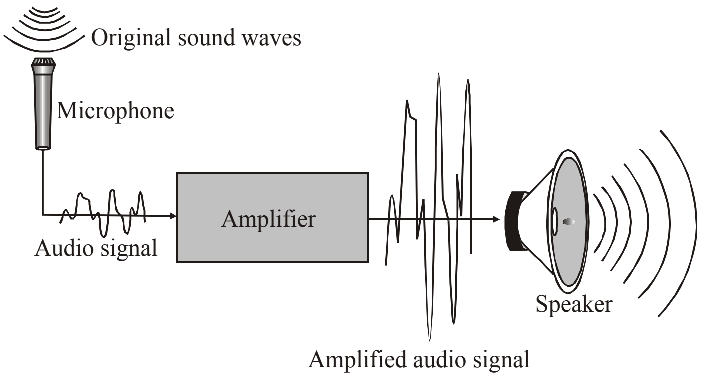


Fig. 1-1. *A basic public address system*

Let us now take an example of a system in which both digital and analog circuits are used. Fig. 1-2 illustrates the basic principle of a compact disk (CD) player. The diagram indicates that the music is stored on the compact disc in digital form. An optical system using laser diode picks up the digital data from the rotating disk. Then, this system transfers the data to the digital-to-analog converter (DAC). The DAC changes the digital data back to analog signal which is an electrical reproduction of the original music. The signal is amplified and sent to the speaker for the listener to enjoy.

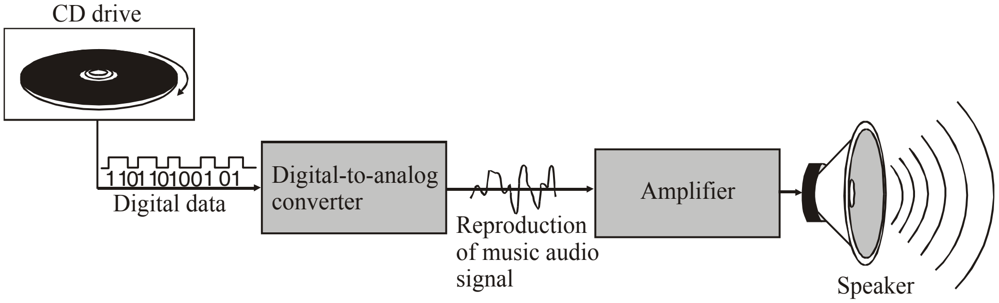

CD drive
1 1 0 1 1 0 1 0 0 1 0 1
Digital data
Digital-to-analog converter
Reproduction of music audio signal
Amplifier
Speaker

Fig. 1-2. *Illustrating the basic principle of a CD player.*

1-6. Advantages of Digital Techniques

A majority of applications in the field of electronics, as well as in most other technologies, use digital techniques to perform operations that were once performed by analog methods. Following are some of the reasons which are responsible for the shift from analog to digital technology.

1. **Digital systems are easier to design.** It is due to the fact that the circuits which are used in digital systems are switching circuits. In such circuits, exact values of voltage or current are not important, rather it is the range (HIGH or LOW) in which they fall.
2. **Storage of information is easy.** This is accomplished by special switching circuits that can latch onto information and hold it for a time as long as it is necessary.
3. **Greater accuracy and precision.** Digital systems have a capability to handle as many digits of precision as we need simply adding more switching circuits. On the other hand, in analog systems, precision is usually limited to three or four digits because the values of voltage and current are directly dependent on the circuit component values. Moreover it is affected by random fluctuations (or noise).
4. **Operation can be controlled by a program.** It is quite easy to design digital systems whose operation is controlled by a set of stored instructions called a program. With the progress in technology day by day, it is becoming even more easier. It is possible to program analog systems. But the variety and the complexity of the available operations are limited.

5. **Digital systems are less affected by noise.** Unwanted fluctuations (or noise) in voltage are not as critical in digital systems as it is in analog systems. It is due to the fact that in digital systems, the exact value of the voltage is not important, as long as the noise is not large enough to prevent us from distinguishing a HIGH voltage level from a LOW voltage level.
6. **More digital circuitry can be fabricated on integrated circuit (IC) devices.** With the tremendous advancement of integrated circuit (IC) technology, both the digital and analog circuitry has benefited. But the benefit is comparatively less in analog because of the relative complexity and its use of devices that cannot be economically integrated. The example of such devices are high-value capacitors, precision resistors, inductors and transformers. This problem has prevented analog systems from achieving the same degree of integration as that of digital systems.

1-7. Limitations of Digital Techniques

We have already discussed several advantages of digital techniques in the last article. Let us now look at the limitations also. There is only one major drawback with the digital techniques, *i.e.*, **The real-world problems are analog.** Thus most of the physical quantities in nature are analog. These quantities are usually the inputs and outputs that are being monitored and controlled by a system. For example, temperature, pressure, position, velocity, liquid level, flow rate etc., all are analog quantities.

In order to take advantage of digital techniques when dealing with analog quantities, we use the following three steps:

1. Convert the real-world inputs to digital form
2. Process the digital information
3. Convert the digital outputs back to real-world analog form.

In order to illustrate this concept further let us consider an example of a flow rate measurement and control system. Fig. 1-3 shows a block diagram for such a system. As indicated in the diagram, the flow rate (*i.e.*, an analog quantity) is measured and its value is converted to a digital quantity by an electronic device called analog-to-digital converter (ADC). Then the digital quantity is processed by a digital circuitry. Finally the digital quantity is converted back to an analog quantity by another electronic device called digital-to-analog (DAC) converter. This analog output is fed to a controller which takes some kind of action to adjust the flow rate.

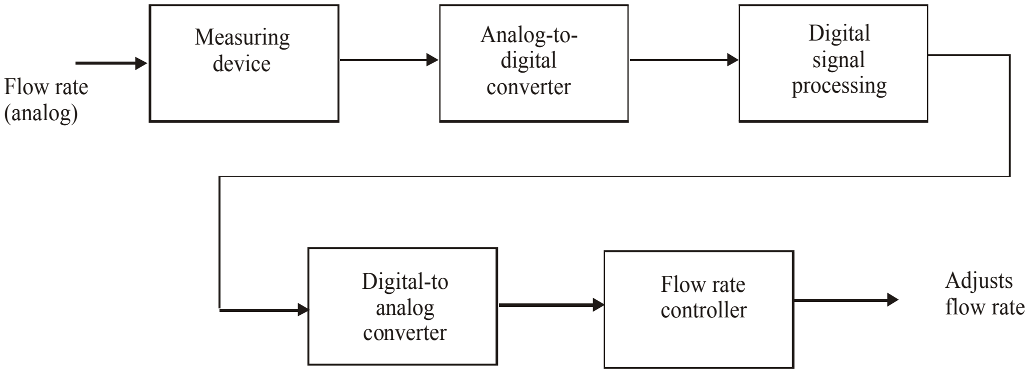

Flow rate
(analog)
Measuring device
Analog-to-digital converter
Digital signal processing
Digital-to analog converter
Flow rate controller
Adjusts flow rate

Fig. 1-3. *Illustrating a flow measurement and control system.*

It is evident from the above example, that there is a need for conversion between analog and digital forms of information in most of the control systems realized using digital techniques. This can be considered as a drawback because of the added complexity and expense. In addition to this, an extra time is required to perform these conversions. However, we must understand that, these factors are outweighed by the numerous advantages of using digital techniques.

1-8. The Future is Digital

Most of the advances in many areas of technology have taken place in the area of digital technology and many more will take place in future.

We have digital mobile phones with a variety of features in them. Digital audio transmission is quite common in the radio broadcast. The quality of the radio sound has improved drastically with the digital broadcast. The internet information at present contains multimedia information, i.e., text, graphics, video, audio, and animated images. The speed to down load the video files (which is considered to be low at present) will increase drastically in future. More devices in future will be having many more features than at present. A digital watch with a mobile phone may be a common thing. We have telephones that are able to receive, sort, and respond to incoming calls like a well trained secretary.

The rapid pace of these advances may even exceed the phenomenal growth we have been experiencing in the past few years-period in which we have seen the availability of computers, modems, CD-ROM drives, modem automobiles making use of microprocessor ICs teenagers with personal computers at home. In future we will have more computer power than our present home or office computer. Children in schools will be able to gather ideas and information and socialize with other children all over the world through the wide spread use of internet. When we watch television for an hour, what we see will have been delivered to home in less than a second and stored in our TVs (or computer's) memory for viewing at our convenience. Reading about a place 5,000 or 10,000 kilometers away may include the sensory experience of being there. In other words, digital technology will continue its high speed incursion into current areas of our lives as well as breaking new ground in ways we may not even have thought about. All we can do is try to learn as much as we can about this technology and enjoy the benefits of new digital systems to be introduced in future.

1-9. The Binary Digits

The binary number system (which you will study in chapter 2) makes use of two digits 1 and 0. These two digits are called **bits** .The term bit is a contraction of binary digit. In digital circuits, two different voltage levels are used to represent the two bits. A 1 is represented by the higher voltage, which is referred to as a HIGH. On the other hand, a 0 is represented by the lower voltage and is referred to as a LOW. This is called positive logic and will be used throughout the book. Thus,

$$\text{HIGH} = 1 \text{ and LOW} = 0$$

A much less common system in which a 1 is represented by a LOW Voltage and a 0 is represented by a HIGH is called negative logic.

In order to represent numbers, letters, symbols instructions and anything else required in a given application (to be implemented by a digital systems), we make use of groups of bits. Such groups of bits are called **codes**. There are several different types of codes used in the field of digital electronic systems. You will study about the different codes in Chapter 2.

1-10. Logic Levels

We have already discussed in the last article that a digital system makes use of bits (*i.e.*, 1 and 0) to represent numbers, letters, symbols, instructions etc. The voltage used to represent these bits (1 and a 0) are called logic levels. Ideally speaking, one voltage level represents a HIGH and the other voltage level represents a LOW. In a practical digital circuit, however, a HIGH can be any voltage between a specified minimum value and a specified maximum value. Likewise, a LOW can be any voltage between a specified minimum and a specified maximum.

Fig. 1-4 illustrates the general range of LOW and HIGH voltages for a digital circuit. The variable $V_{H\,(max)}$ represents the maximum HIGH voltage value, and $V_{H\,(min)}$ represents the minimum HIGH voltage value. Similarly the maximum LOW voltage value is represented by $V_{L\,(max)}$ and the minimum LOW voltage value is represented by $V_{L\,(min)}$. The range of voltages between $V_{L\,(max)}$ and $V_{H\,(min)}$ is a range of uncertainty. A voltage in the range of uncertainty can appear as either a HIGH or a LOW to a given circuit; one can never be sure. Therefore, the values in the uncertain range are not use in digital circuits. For example, the HIGH values for a certain type of digital circuit may range from 2 V to 5 V and LOW values may range from 0 V to 0.8 V. So, if a voltage of 3.9 V is applied, the circuit will accept is as a HIGH or binary 1. On the other hand, if a voltage of 0.5 V is applied, the circuit will accept it as a LOW or binary 0.

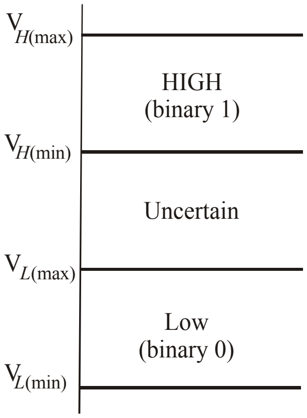


Fig. 1-4. *Illustrating logic levels in a digital circuit.*

1-11. Representation of Digital Waveforms

A digital waveforms consist of voltage levels that are changing back and forth between the HIGH and LOW states. These can also be described as the one that is composed of series of pulses, or a pulse train as shown in Fig. 1-5.

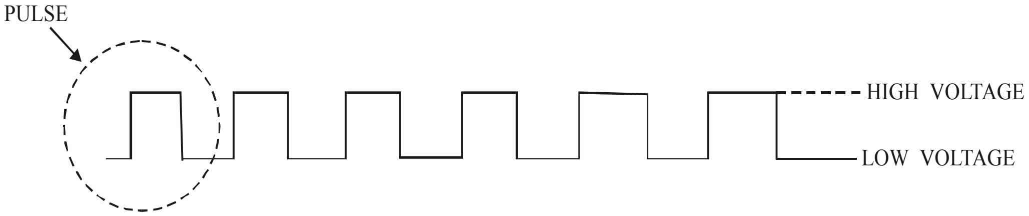


Fig. 1-5. *Representation of a digital waveform.*

Fig. 1-6 (a) and (b) shows the representation of a pulse separately. Notice that the pulse shown in Fig. 1-6(a) is referred to as a positive going pulse because this pulse is generated when the voltage (or the current) goes from its normally LOW level to its HIGH level. On the other hand, the pulse shown in Figure 1-6 (b) is generated when the voltage (or current) goes from its normally HIGH level to its LOW level and then back to its HIGH level. It may also be noted that a pulse has two edges: (1) a rising edge and (2) a falling edge. A rising edge is a one that occurs when a voltage (or a current) goes from its normally LOW level to its HIGH level. On the other hand, a falling edge is a one that occurs when a voltage (or current) goes from its HIGH level back to LOW level. The rising edge in a positive-going pulse is also known as leading edge and the falling edge as the trailing edge. The reason for this is that the leading edge occurs first at time 't_0' while the trailing edge occurs later at time 't_1'. The pulses in Fig. 1-6 are **ideal** because the rising and falling edges change in zero time *(instantaneously).* In actual practice, these transitions never occur instantaneously, although for most digital working, we can assume pulses to be an ideal one.

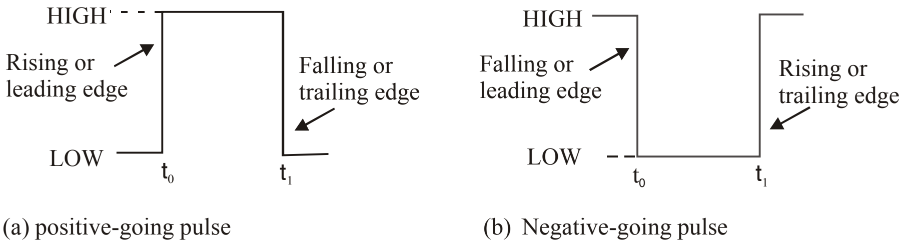


Fig. 1-6. *Representation of a digital waveform.*

1-12. The pulses Characteristics

We have already discussed in the last article about an ideal pulse. Now we shall discuss about a non-ideal pulse *i.e.* a pulse which is generated by the practical digital circuit. Such a pulse has the following important five characteristics:

1. Rise time
2. Fall time
3. Pulse width
4. Overshoot
5. Ringing

Fig. 1-7 shows a positive- going non-ideal pulse. The time required for the pulse to go from its LOW voltage level to its HIGH voltage level is called the **rise time** (t_r) while the time required for the transition from the HIGH voltage level to the LOW voltage level is called the **fall time** (t_f). In actual practice, it is common to measure rise time from 10% of the pulse amplitude *i.e.* (height from baseline) to 90% of the pulse amplitude and to measure the fall time from 90 % to 10% of the pulse amplitude, as indicated in Figure 1-7. It may be noted that the bottom 10% and the top 10% of the pulse are not included in the rise and fall times because of the non-linearities in the waveform in these areas.

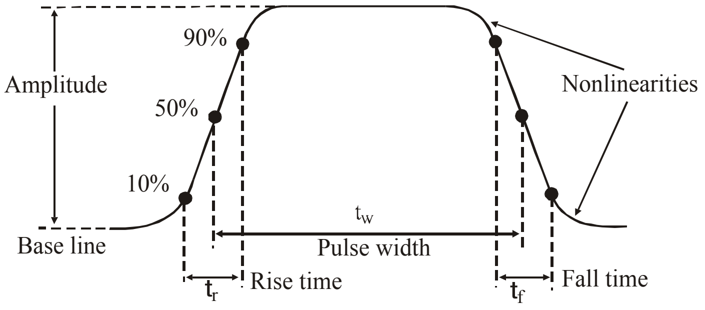


Fig. 1-7. *Non-ideal pulse Characteristics.*

The Pulse Width (t_w) is a measure of the duration of the pulse and is often defined as the time interval between the 50% points on the rising and falling edges, as indicated in Fig. 1-7.

Overshoot and Ringing. These are undesirable pulse characteristics. The overshoot could be positive and negative. The overshoot is caused by a capacitive effect in the circuit or measuring instrument that results in the voltage exceeding the normal HIGH and LOW levels for a short time on the rising and falling edges, as indicated in Fig. 1-8 (a). Ringing on the rising and falling edges of a pulse is actually an oscillation caused by capacitance and inductance in the circuit, as indicated in Fig. 1-8(b). Notice that the ringing dies out after a short time.

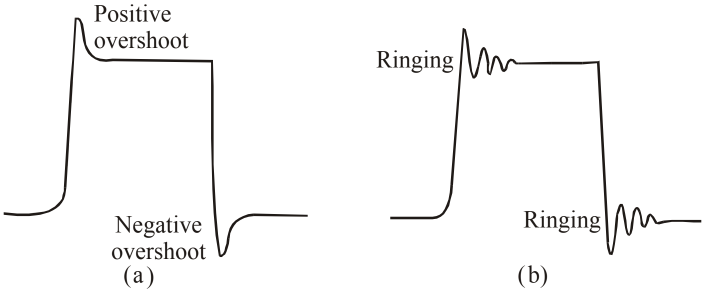
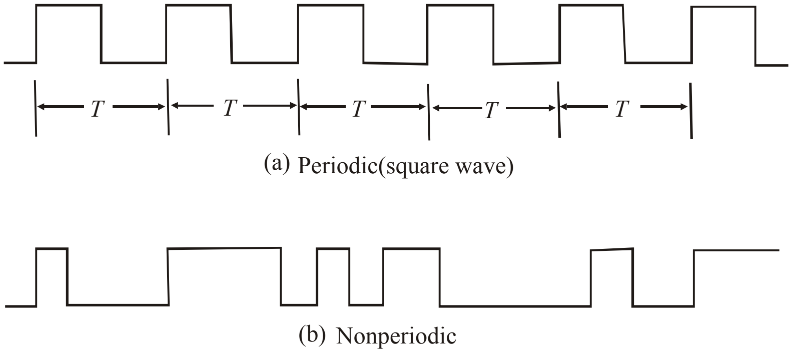


Fig. 1-8. *Overshoot and Ringing.*

1-13. Waveform Characteristics

We have already discussed in the Article 1-11 that the waveforms encountered in digital electronic systems are composed of series of pulses, or pulse trains. These can be classified as either periodic or non-periodic. A periodic pulse waveform is one that repeats itself at a fixed interval, called a period (T). The frequency (f) is the rate at which it repeats itself and is measured in hertz (Hz). A non-periodic pulse wave form, of course, does not repeat itself at fixed intervals and may be composed of pulses of differing pulse widths and/or differing time intervals between the pulses. An example of each type is shown in Fig. 1-9.



(a) Periodic(square wave)

(b) Nonperiodic

Fig. 1-9. *Digital Waveforms.*

The frequency (f) of a pulse waveform is the reciprocal of the period. The relationship between frequency and period is given by the equation:

$$f = \frac{\mathbf{1}}{T} \qquad \text{(Hz)}$$

$$\text{or} \qquad T = \frac{\mathbf{1}}{f} \qquad \text{(s)}$$

An important characteristic of a periodic digital waveform is its **duty cycle**. The duty cycle is defined as the ratio of the pulse width (t_w) to the period (T) expressed as a percentage, *i.e.*

$$Duty\ \ cycle = \frac{Pulse\ width}{Period} \times 100\%$$

$$= \frac{t_W}{T} \times 100\%$$

Example 1-2. *Figure* 1-10 *shows a portion of a periodic digital waveform. The measurements are in milliseconds. Determine the following*:

(*a*) *period* (*b*) *frequency* (*c*) *duty cycle*

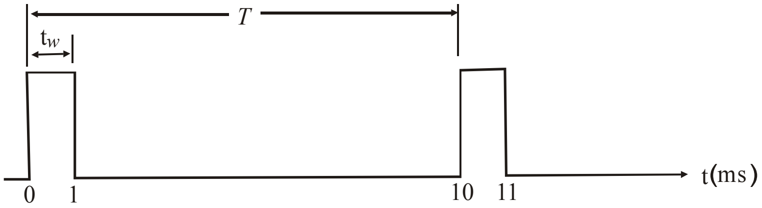


Fig. 1-10.

Solution. Given : A portion of the periodic digital waveform whose time period is indicated as 10 ms.

(*a*) *Period*

We know that time period of the waveform.

T= 10 ms **Ans.**

(*b*) *Frequency*

We know that frequency of the waveform,

$$f = \frac{1}{T} = \frac{1}{10 ms} = \frac{1}{10 \times 10^{-3}} = 100\,Hz \textbf{ Ans.}$$

(*c*) *Duty cycle*

We know that duty cycle of the waveform,

$$\text{Duty cycle} = \left(\frac{t_w}{T}\right) \times 100\%$$

$$= \left(\frac{1}{10}\right) \times 100\% = 10\% \textbf{ Ans.}$$

1-14. A Digital Waveform Carries Binary Information

It will be interesting to know that binary information handled by digital systems appears as waveforms that represent sequences of bits. When the waveform is HIGH, a binary 1 is present. Similarly when the waveform is LOW, a binary 0 is present. Each bit in a sequence occupies a defined time interval called a **bit time.**

In many digital systems, all waveforms are synchronized with a basic timing waveform called the **clock.** The clock is a periodic waveform in which each interval between pulses (the period) equals one bit time. Fig. 1-11 shows an example of a clock waveform.

Notice that, in this case, each change in level of waveform A occurs at the leading edge of the clock waveform. In other cases, level changes occur at the trailing edge of the clock. During each bit time of the clock, waveform X is either HIGH or LOW. These HIGHs and LOWs represent a sequence of bits as indicated. A group of several bits can be used as a piece of binary information such as a number or a letter.

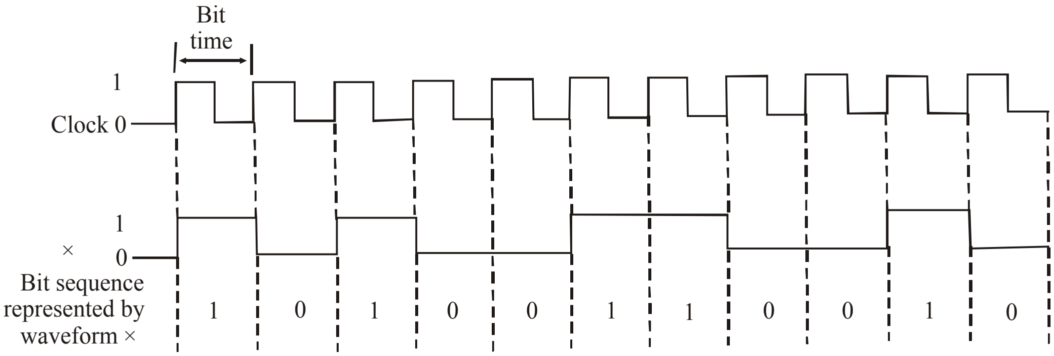


Fig. 1-11. *A clock waveform illustrating a synchronization with a waveform representation of a sequence of bits.*

1-15. Timing Diagram

A timing diagram is a graph of digital waveforms showing the proper time relationship of all the waveforms and how each waveform changes in relation to the others. Fig. 1-12 is an example of a simple timing diagram that shows how the clock waveform and waveform X are related.

A timing diagram can consist of any number of related waveforms. By looking at a timing diagram we can determine the states (HIGH and LOW) of all the waveforms at any specified point in time and the exact time that a waveform changed state relative to the other waveforms. Fig. 1-12 is an example of a timing diagram made up of four waveforms.

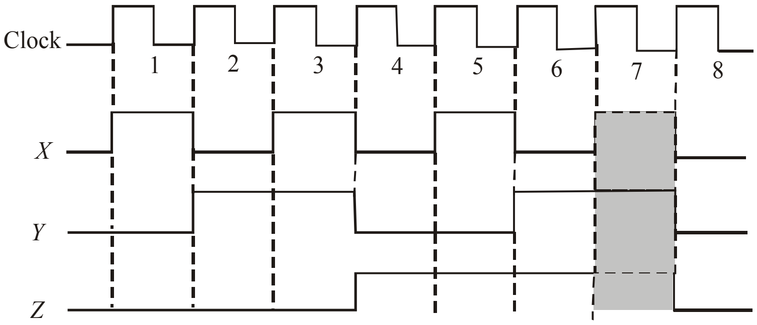


Fig. 1-12. *Example of a timing diagram.*

From this timing diagram we can see, that all the three waveforms *i.e.* (X, Y and Z) are HIGH only during bit time 7 and they all change back LOW at the end of bit time 7.

Example l-3. *Draw a timing diagram for a digital signal that continuously alternates between* 0.2 *V* (*binary* 0) *for* 2 *ms and* 4.4 *V* (*binary*) *for* 4 *ms.*

Solution. Given: a digital signal that alternates between 0.2 and 4.4 V.

We know that for a digital signal that continuously alternates between 0.2 V (binary 0) for 2 ms is indicated by a line at a LOW voltage and 4.4 V(binary) for 4 ms by a line at a HIGH voltage as shown in Fig. 1-13 (*a*) **Ans.**

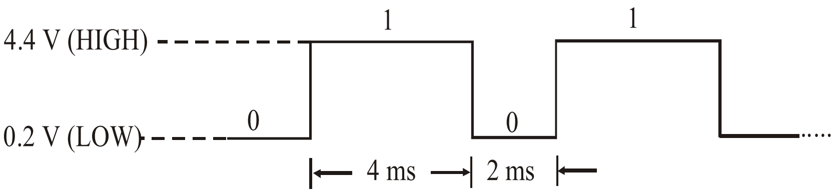


Fig. 1-13.

1-16. Data Transfer

The transmission of information in any digital system from one place to another is the most common operation in today's fast moving world. The information can be transmitted over a short distance of few centimeters on the same circuit board, or over a distance of several kilometers when an operator at a computer terminal in one city is communicating with a computer in another city. The information (called **data)** that is transmitted is in binary form. It is generally represented as voltages at the outputs of a sending circuit that are connected to the inputs of a receiving circuit.

There are two methods for the transmission of digital information namely parallel- and serial-data transfer. Both these methods are discussed one by one in the following pages.

1-17. Parallel Data Transfer

In this method of data transfer all the bits with in a group are sent out on separate lines at the same time. In other words, there is one line for each bit. To illustrate this point let us consider the following example. Suppose we wish to transfer a group of bits (called binary number) 10110 from a circuit (or a system) X to circuit (or a system) Y using parallel transmission. Then as shown in Fig. 1.14 (*b*), each bit of the binary number is represented by one of the circuit X outputs. The output X_4 of the circuit X is the most-significant bit (MSB) and the output X_0 is the least-significant bit (LSB). Notice that each of the circuit X outputs is connected to the corresponding input of circuit Y. So that all the five bits of information are transmitted simultaneously (*i.e.* in parallel).

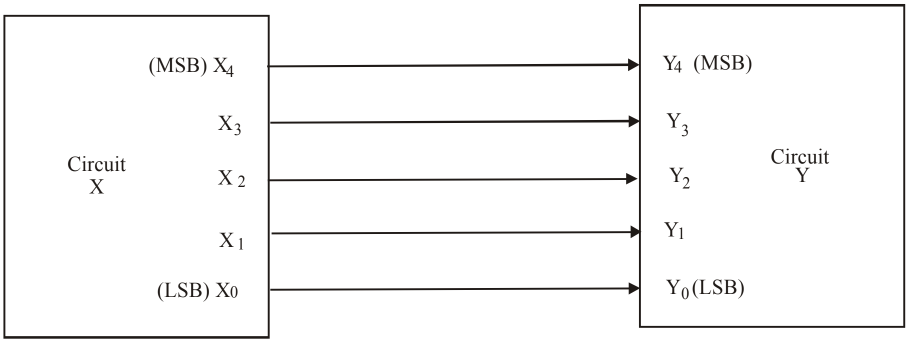


Fig. 1-14. *Illustrating parallel transmission of data from circuit X to circuit Y.*

1-18. Serial Data Transfer

In this method of data transfer, the bits within a group are sent one at a time along a single conductor (or single line). In other words, there is no separate lines as in parallel data transfer. Consider an example to illustrate the point. Suppose we wish to transfer the same binary number we used for parallel data transfer *i.e*. 10110. Fig. 1-15 shows how the bits of the number are transmitted serially from circuit X to circuit Y. Here the output of circuit X will produce a digital signal whose voltage level will charge at regular intervals in accordance with the binary number being transmitted. In this way, the information is being transmitted a bit at a time (serial) over the one conductor.

The timing diagram shown in Fig. 1-15 indicates how the signal level varies with time. During the first time interval, T_0, the signal is at the 0 level; during the interval T_1, the signal is at the level 1; and so on. It may be noted that the most-significant bit (MSB) of the number is transmitted first and the least-significant bit (LSB), at the end.

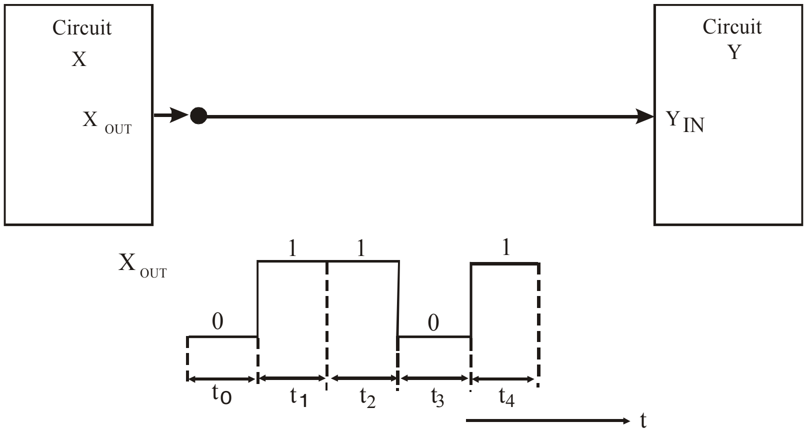


Fig. 1-15. *Illustrating serial transmission of data from circuit X to circuit Y.*

1-19 Comparison Between Parallel and Serial Data Transfer

We have already discussed in the Art. 1-17, that in parallel data transfer all the bits with in a group are sent out on separate lines at the same time. On the other hand, in serial data transfer, the bits within the group are sent one at a time along a single line.

The difference between the parallel and serial data transfer can be discussed on the basis of speed and circuit simplicity, some of the major differences is as given in table 1-1.

Table 1-1. *Comparison between parallel and serial data transfer.*

Sr. No.	*Parallel*	*Serial*
1.	In this method of data transfer number of lines required is equal to the number of bits to be transferred at one time.	In this method of data transfer, only one line is required for any number of bits to be transferred.
2.	This method is a fast way of data transfer as all the bits are transferred simultaneously.	This method is a slow way of data transfer as only one bit is transferred at a time.

The comparison between parallel and serial method of data transfer will be encountered many times in discussions through out the text in this book.

Example 1- 4. *Determine the total time required to serially transfer the eight bits contained in waveform A of Fig.* 1-16 *and indicate the sequence of bits. The left most bit is the first to be transferred. The* 100 *kHz clock is used as reference*.

(*a*) *What is the total time to transfer the same eight bits in parallel*?

(*b*) *Total time to transfer eight bits in parallel*.

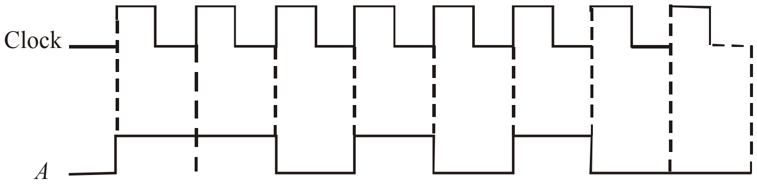


Fig. 1-16.

Solution. Given : Frequency of the clock, $f = 100$ kHz $= 100 \times 10^3$ Hz.

(*a*) *Total time required and sequence of bits*

Total time required

We know that the time period,

$$T = \frac{1}{f} = \frac{1}{100 \times 10^3} = 10 \times 10^{-6} \text{ s} = \mu\text{s}$$

Since each bit is to be transmitted serially requires one clock cycle (or 10 μs), therefore time required to transfer eight bits serially,

$$t_{\text{serial}} = 8 \times 10\ \mu\text{s} = 80\ \mu\text{s}\ \textbf{Ans.}$$

Sequence of bits

To determine the sequence of bits, let us examine the waveform in Fig. 1-16 during each bit time. If waveform A is HIGH during the bit, a 1 is transferred. If waveform A is low during the bit time, a 0 is transferred. The bit sequence is illustrated in Fig. 1-17. The left-most bit is the first to be transferred.

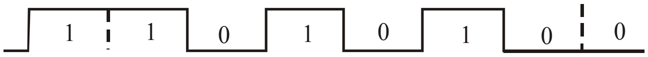


Fig. 1-17.

(*b*) *Total time to transfer eight bits in parallel.*

We know that in parallel data transfer, all the eight bits will be transmitted simultaneously. Therefore, the total time required to transfer eight bits in parallel,

$$t_{\text{parallel}} = 10\ \mu\text{s}\ \textbf{Ans.}$$

1-20. Digital Integrated Circuits

It will be interesting to know that most of the digital circuits used in modern digital systems are integrated circuits called ICs in short. There is a wide variety of ICs available in the market than their discrete-component counterparts.

There are several integrated-circuit fabrication technologies which are used to produce digital ICs. The most common fabrication technologies are TTL, CMOS, ECL and BiCMOS. Each differs in the type of circuitry used to provide the desired logic operation. For example, TTL (transistor-transistor logic) uses the bipolar transistor as its main circuit element, while CMOS (complementary metal-oxide-semiconductor) uses the enhancement-mode MOSFET as its principal circuit element. We will learn about the various IC technologies, their characteristics, and their relative advantages and disadvantages in Chapter 8.

1-21. Memory

When an input signal is applied to some electronic devices or circuits, the output somehow changes in response to the input, and when the input signal is removed the output returns to its original state. The circuits do not exhibit the property of memory, since their outputs revert back to normal. However there are certain types of devices and circuits that do have memory. When an input is applied to such a circuit, the output will change its state, but it will remain in the new state even after the input is removed. This property of retaining its response to a momentry input is called memory. Fig. 1-18 illustrates non-memory and memory operations.

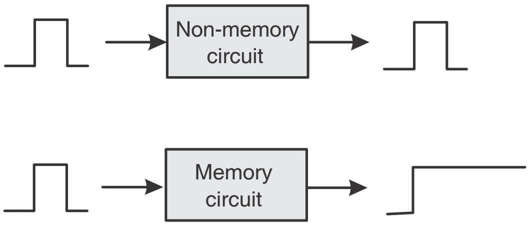


Fig. 1-18. *Comparison of non-memory and memory operations.*

Memory devices and circuits play a very important role in digital systems. It is because of the fact that they provide a means for storing binary numbers either temporarily or permanently, with the ability to change the stored information at any time. The various memory elements include magnetic types and those which utilize electronic latching circuits (called latches and flip-flops). The latches and flip-flop will be discussed later in the book.

1-22. Digital Computers

Digital techniques have found their way into innumerable areas of technology, but the area of automatic **digital computers** is by far the most notable and most extensive. Although digital computers affect some part of all of our lives, it is doubtful that many of us know exactly what a computer does. In simplest terms, a computer is a system of **hardware** that performs arithmetic operations, manipulates data (usually in binary form), and makes decisions.

For the most part, human beings can do whatever computers can do, but computers can do it with much greater speed and accuracy. This is in spite of the fact that computers perform all their calculations and operations one step at a time. For example, a human being can take a list of 10 numbers and find their sum all in one operation by listing the numbers one over the other and adding them column by column. A computer, on the other hand, can add numbers only two at a time, so that adding this same list of numbers will take nine actual addition steps. Of course, the fact that the computer requires only a microsecond or less per step makes up for this apparent inefficiency.

A computer is a faster and more accurate than people are, but unlike most people it must be given a complete set of instructions that tell it exactly what to do at each step of its operation. This set of instructions, called a **program** is prepared by one or more persons for each job the computer is to do. Programs are placed in the computer's memory until in binary-coded form, with each instruction having a unique code. The computer takes these instruction codes from memory one at a time and performs the operation called for by the code. Much more will be said about this later.

1-23. Major Parts of a Computer

There are several types of computer systems, but each can be broken down into the same functional units. Each unit performs specific functions, and all units function together to carry out the instructions given in the program. Fig. 1-19 shows the five major functional parts of a digital computer and their interaction. It may be noted that the solid lines with arrows represent the flow of data and information rule. The dashed lines with arrows represent the flow of timing and control signals.

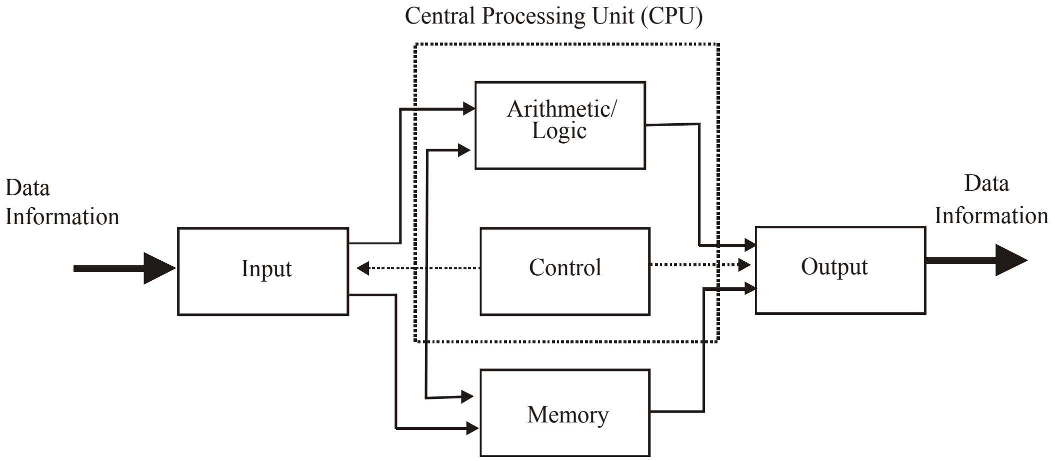


Fig. 1-19. *Major Parts of a digital computer indicating the flow of information among different parts.*

The major functions of each unit are:

1. **Input unit.** Through this unit a complete set of instructions and data is fed into the computer system and into the memory unit, to be stored until needed. The information typically enters the input unit from a keyboard or a disk.
2. **Memory unit.** The memory stores the instructions and data received from the input unit. It stores the results of arithmetic operations received from the arithmetic unit. It also supplies information to the output unit.
3. **Control unit.** This unit takes instructions from the memory unit one at a time and interprets them. It then sends appropriate signals to all the other units to cause the specific instruction to be executed.
4. **Arithmetic/logic unit.** All arithmetic calculations and logical decisions are performed in this unit, which can then send results to the memory unit to be stored.
5. **Output unit.** This unit takes data from the memory unit and prints out, displays, or otherwise presents the information to the operator (or process, in the case of process control computer).

1-24. Central Processing Unit (CPU)

As the diagram in Fig. 1-19 shows, the control and arithmetic/logic units are often considered as one unit called the Central Processing Unit (CPU). The CPU contains all the circuitry for fetching and interpreting instructions and for controlling and performing the various operations called for by the instructions.

1-25. Types of Computers

All computers are made up of the basic units described above, but they can differ as to physical size, operating speed, memory capacity, and computational power, as well as other characteristics. Computers are often classified according to physical size which often, although not always, is an indication of their relative capabilities. The three basic classifications, from smallest to largest, are: microcomputer, minicomputer (workstation), and mainframe. As microcomputers have become more and more powerful, the distinction between microcomputers has become rather blurred, and we have begun to distinguish only between small computers-those that can fit in an office or on a desktop or a lap-and large computers-those that are too big for any of those places. In this book we will be concerned mainly with microcomputers.

1-26. Microcomputers

A microcomputer is the smallest type of computer. It generally consists of several IC chips including a microprocessor chip, memory chips, and input/output interface chips along with input/output devices such as a keyboard, video display, printer, and disk drives. Microcomputers were developed as a result of tremendous advances in IC fabrication technology that made it possible to pack more and more digital circuits onto a small chip. For example, the microprocessor chip contains-at a minimum- all of the circuits that make up the CPU portion of the computer, that is, the control unit and the arithmetic/logic unit. The microprocessor, in other words is a "CPU on a chip".

1-27. Microcontrollers

Most of us are familiar with general purpose microcomputers such as the IBM PC and its clones and the Apple Macintosh, which are used in more than half of our homes and in almost all of our business. These microcomputers can perform a wide variety of tasks in a wide range of applications depending on the software (programs) they are running. There is a more specialized type of microcomputer called a microcontroller which is not a general-purpose computer. Rather, it is designed to be used as a dedicated or embedded controller which helps monitor and control the operation of a machine, a piece of equipment, or a process. Microcontrollers are microcomputers because they use a microprocessor chip as the CPU, but they are much smaller than general- purpose microcomputers because the input/output devices they normally use are much smaller. In fact, the input/output devices-as well as memory-are usually right on the same chip as the microprocessor. These single chip microcontrollers are employed in a wide variety of control applications such as appliance control, metal-working machines, VCR, automated teller machines, photocopiers, automobile ignition systems, antilock brakes medical instrumentation, and much more.

1-28. Simulation tools

PSPICE, Electronics Workbench (EWB) and Circuit Maker are some of the software systems that are used as simulation tools. These provides an accurate simulation of digital and analog circuit operation, along with simulation of instruments used by a technician to measure IC, component and circuit characteristics.

With these software, you have the ability and test most of the circuits presented in the text.

SUMMARY

In this chapter, we have learnt that:

1. There are two basic ways representing the numerical value of physical quantities: namely analog and digital.

2. Most of the quantities in the real world are analog, but digital techniques are generally superior to analog and most of the advances in future will be in the digital realm.

3. The digital level for 1 is commonly represented by a voltage of 5 V in digital systems. A voltage of 0 V is used for the 0 level.

4. The frequency of clock waveform is equal to the reciprocal of the waveforms period.

5. There are two ways to transfer digital information (*i*) parallel and (*ii*) serial. In parallel transfer, all the bits are transferred simultaneously hence it is faster. But we require several conductors for parallel data transfer. In serial transfer, the bits are transferred, one at a time, hence it is slower. But we require only two conductors for serial data transfer.

6. The main parts of a computer are the input, control, memory, arithmetic/ logic and output units.

7. The combination of the arithmetic/ logic unit and the control unit makes up the CPU (central processing unit).
8. A microcomputer usually has CPU that is fabricated on a single chip called microprocessor.
9. A microcontroller is a microcomputer specially designed for control engineering and instrumentation applications.

GLOSSARY

Analog representation. Representation of a quantity that varies over a continuous range of values.

Digital representation. Representation of a quantity that varies in discrete steps over a range of values.

Digital system. Combination of devices designed to manipulate physical quantities that are represented in digital form.

Analog system. Combination of devices designed to manipulate physical quantities that are represented in analog form.

Analog-to-digital converter (ADC). Circuit that converts an analog input to a corresponding digital output.

Digital-to-analog converter (DAC). Circuit that converts a digital input to a corresponding analog output.

Decimal System. A number system that uses 10 different digits or symbols to represent a quantity.

Binary System. A number system in which there are only two possible digit values, 0 and 1.

Bit: A digit in the binary number system.

Timing Diagram. Depiction of logic levels as related to time.

Digital/Logic Circuits. Any circuit that behaves according to a set of logic rules.

Parallel data transfer. Operation by which several bits of data are transferred simultaneously from one digital circuit to another.

Serial Data Transfer. Transfer of data from one digital circuit to another one bit at a time.

Memory: Ability of a circuit's output to remain at one state even after the input condition that caused that state is removed.

Digital computer. System of hardware that performs arithmetic and logic operations, manipulates data and makes decisions.

Program. Sequence of binary-coded instructions designed to accomplish a particular task by a computer; also the act of entering information into a programmable device (*e.g.* EPROM, PLD).

Input unit (IU). Part of a computer that facilitates the feeding of information into the computer's memory unit or ALU.

Memory unit (MU). Part of a computer that stores instructions and data received from the input unit, as well as results from the arithmetic/logic unit.

Control Unit (CU). Part of a computer that provides decoding of program instructions and the necessary timing and control signals for the execution of such instructions.

Arithmetic/Logic unit (ALU). Digital circuit used in computers to perform various arithmetic and logic operations.

Output unit (OU). Part of a computer that receives data from the memory unit or ALU and presents it to the outside world.

Control processing unit (CPU). Part of a computer which is composed of the arithmetic/logic unit (ALU) and control unit.

Microcomputer. System consisting of microprocessor IC, memory ICs and I/O interface ICs. In some cases all the devices are in one single IC.

Microprocessor. Large-scale integrated circuit that contains the central processing unit.

Microcontroller. Small microcomputer used as a dedicated controller for a machine, a piece of equipment or a process.

DESCRIPTIVE QUESTIONS

1. What are the advantages of digital techniques over analog?
2. What is the major limitation to the use of digital techniques?
3. How are the rise time and fall time of a pulse measured?
4. Knowing the period of a waveform, how do you find the frequency?
5. Explain what a clock waveform is?
6. What is the purpose of a timing diagram?
7. What is the main advantage of parallel transfer over serial transfer of binary data?
8. Explain, how a microprocessor is different from a microcomputer.
9. Describe briefly, how a microcomputer is different from a microcomputer.

TUTORIAL PROBLEMS

1. Which of the following involve analog quantities and which involve digital quantities?

   (*a*) Automobile speedometer (*b*) Temperature of a room

   (*c*) Ten- position Switch (*d*) Sand grains on the beach

   (*e*) Current flowing out of an electrical outlet.

2. Determine the frequency of the waveform shown in Fig. 1-20.

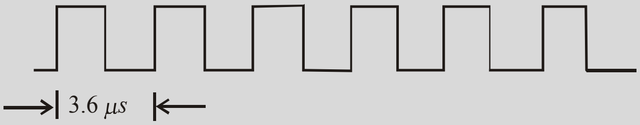


Fig. 1-20.

3. What is the frequency of a periodic waveform having a period of 50 µs? (**Ans**. 20 MHz.)
4. What is the time period from the rising edge of one pulse to the rising edge of the next pulse on a waveform whose frequency is 8 MHz. (**Ans.** 0. 125 µs)
5. Sketch the serial and parallel representations of the 4-bit number 0111. If the clock frequency is 5 MHz find the time to transmit using each method. (**Ans.** t_{serial}= 0.8 µs and $t_{parallel}$ = 0.2 µs)
6. Sketch the serial and parallel representations of hexadecimal number 4A (equivalent to binary 01001010) Assume a 4- bit parallel system and a clock frequency of 4 KHz.
7. Differentiate between positive logic and negative logic.

(*Mahatma Gandhi University, Jan.2007*)

8. Define analog and digital system and list various characteristics of digital system. (*GBTU/MTU, 2007-08*)

9. Describe the merit and shortcomings of a digital signal. (*Jamia Millia lslamia University. 2007*)

MULTIPLE CHOICE QUESTIONS

1. A quantity having continuous values is :

(*a*) an analog quantity (*b*) a digital quantity

(*c*) a binary quantity (*d*) a natural quantity.

2. The term bit means:

(*a*) a small amount of data (*b*) a binary digit

(*c*) a 1 or a 0 (*d*) both answers (*b*) and (*c*).

3. The time interval on the leading edge of the pulse between 10 % and 90% of the amplitude is:

(*a*) rise time (*b*) fall time

(*c*) period (*d*) pulse width

4. A pulse in a certain digital waveform occurs every 10 ms. The frequency is:

(*a*) 1 Hz (*b*) 10 Hz

(*c*) 100 Hz (*d*) 1 kHz

5. In a certain digital waveform, the period is twice the pulse width. The duty cycle is:

(*a*) 200 % (*b*) 100 %

(*c*) 50 % (*d*) none of the above.

ANSWERS

1. (*a*) **2.** (*b*) **3.** (*a*) **4.** (*c*) **5.** (*c*)

2

NUMBER SYSTEMS AND CODES

OUTLINE

Objectives

Upon completion of this chapter, you should be able to:

- Know the decimal, binary, octal and hexadecimal-number systems.
- Convert numbers from decimal-to-binary and from binary-to-decimal.
- Convert numbers from binary-to-octal and from octal-to-binary.
- Convert numbers from binary-to-hexadecimal and from hexadecimal-to-binary.

- Express decimal numbers using the BCD code.
- Understand the difference between the BCD code and the straight binary code.
- Describe the parity method for error detection.
- Determine parity (odd or even) of digital data.

2-1. Introduction

The number systems are used quite frequently in the field of digital electronics and computers. However the type of number system used in computers could be different at different stages of the usage. For example, when a user key-in some data into the computer, he/she will do it using decimal number system *i.e.* the system we all have used for several years for doing arithmetic problems. But when the information goes inside the computer, it needs to be converted to a form suitable for processing data by the digital circuitry. Similarly when the data has to be displayed on the monitor for the user, it has to be again in the decimal number system. Hence the conversion from one number system to another one is an important issue to be understood.

Now we shall start our studies in this chapter with the different types of number systems. We will understand how the counting is done in different number systems. We will also discuss, how to convert one number system to another one.

2-2. Types of Number Systems

There are several number systems but the following are the important ones in the field of digital electronics:

1. Decimal number system
2. Binary number system
3. Octal number system
4. Hexadecimal number system

Now we shall study all these number systems one by one in the following pages.

2-3. Decimal Number System

The decimal number system is composed of 10 numerals or symbols. These symbols are 0,1,2,3,4,5,6,7,8 and 9. Using these 10 symbols as digits of a number, we can express any quantity-how big or small it could be. The decimal system is also called the base-10 system because it has 10 digits. The decimal number system has evolved naturally as a result of the fact that people have 10 fingers on their hands. The word "digit" is derived from the Latin word for "finger".

The decimal number system is a positional value system in which the value of a digit depends on its position. For example, consider the decimal number 462. We know that the digit 4 actually represents 4 hundreds, the 6 represents 6 tens and the 2 represents 2 units. Now since, the digit 4 carries the most weight of the three digits, therefore, it is referred to as the *most-significant digit* (MSD). On the other hand, the digit 2 carries the least weight and is called the *least-significant weight* (LSD).

Let us consider another example, say the decimal number 26.48. This number is actually equal to 2 tens plus 6 units plus 4 tenths plus 8 hundredths. It can be thought of to be equal to (2*10+6*1+4*0.1+8*0.01). The decimal point is used to separate the integer and fractional parts of the number.

As a matter of fact, the various positions relative to the decimal point carry weights that can be expressed as powers of 10. This is illustrated in Fig. 2-1 where the number 3547.216 is represented. The decimal point separates the positive powers of 10 from the negative powers.

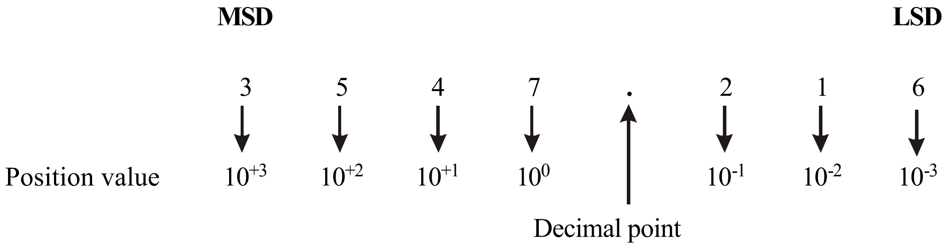


Fig. 2-1. *Illustrating position values as powers of 10 in a decimal number system.*

It is evident from the above diagram that the number 3547.216 is equal to:

$(3 \times 10^{+3}) + (5 \times 10^{+2}) + (4 \times 10^{+1}) + (7 \times 10^{0}) + (2 \times 10^{-1}) + (1 \times 10^{-2}) + (6 \times 10^{-3})$

Thus in general any decimal number is equal to the sum-of- products of each digit value and its position value.

2-4. Counting in Decimal Number System

We have already discussed in the last article that the decimal system is composed of 10 digits, *i.e.*, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Thus when counting in the decimal number system, we start with 0 in the units position and take each digit in progression until we reach 9. After reaching 9 we have finished with the decimal digits. To count beyond 9, we form two digit combinations, with the second digit *i.e.*, "1" in the tens position followed by 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 in the units position. Thus we count beyond 9 as 10, 11, 12, 13 and so on. This process continues until we reach 19.

To count beyond 19, we take the digit 2 at the tens position and digits 0, 1, 2,…..at the units position. Thus we count as 20, 21, 22, and so on until the count of 99 is reached. After reaching at 99, we have again finished with the decimal digits. To count beyond 99, we form three digit combinations with the second digit, *i.e.*, 1 at the hundreds position, first digit *i.e.*, 0 at the tens position and another 0 at the units position. So we count as 100, 101, 102 and so on. This process is shown in Fig. 2-2.

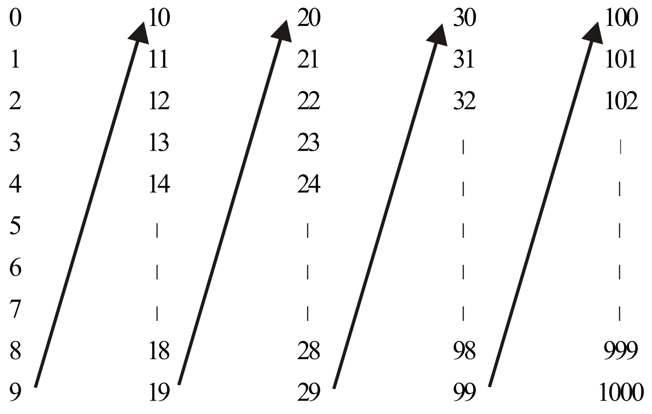


Fig 2-2. *Counting in a decimal number system.*

2-5. The Binary Number System

We have already discussed in the last article about the most commonly used number system (*i.e.*, the decimal number system) in day-to-day life. Unfortunately, the decimal system cannot be conveniently implemented in digital systems. The major reason for this is that it is very difficult to design digital electronic equipment which can work with 10 different voltage levels (each voltage level representing one decimal character, 0 through 9). On the other hand, it is very easy to design simple but accurate digital electronic circuits which operate with only two voltage levels. It is because of this reason, that almost every digital electronic system uses the binary (or base-2) number system as the basic number system of its operations. However, it may be noted that other number systems are also used often with the binary number system.

As discussed in the last chapter (Art. 1-10), the binary number system has two digits, 0 and 1. These digits are also called bits. The binary number system can be used to represent any quantity that can be represented in decimal or other number systems. However, it may be noted carefully, that it will take a greater number of bits to express a given quantity.

Like decimal system, the binary number system is also a position-value system where each bit has its own value or weight expressed as a power of 2. This is illustrated in Fig. 2-3 by considering a binary number 1101.101. As shown in the diagram, places to the left of the binary point are positive powers of 2. Whereas the places to the right of the binary point are negative powers of 2.

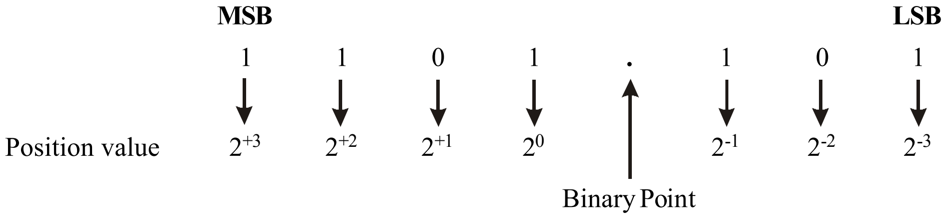


Fig. 2-3. *Illustrating a position values as powers of 2 in a binary number system.*

It may be noted that in the binary number indicated in Fig. 2-3 there are four bits to the left of the binary point. These four bits represent the integer (or whole number) part of the number. There are three bits to the right of the binary point. These three bits represent the fractional part of the binary number. The bit at the left-most position (*i.e.*, the bit with the largest position value) is called the most-significant bit (MSB). Similarly, the bit at the right-most position (*i.e.*, the bit with the smallest position value) is called the least significant bit (LSB).

2-6. Counting in the Binary Number System

In order to count in binary number system, first we look at how we count in decimal system. We start at 0 and count up to 9 before we run out of digits. We then start another digit position (to the left) and continue counting 10 through 99. At this point we have exhausted all two-digit combinations, so a third digit is needed to count from 100 through 999.

A comparable situation occurs when we count in binary, except that we have only two digits called bits. Thus we begin counting as 0,1. At this point we have used both the digits. To count beyond this, we form two digit combinations: second digit (*i.e.*, 1) followed by the first digit (*i.e.*, 0) and next second digit followed by second digit again. So we continue to count as 10,11. Now we have exhausted all two-digit combinations, so we need three digit combinations. The three-digit combinations again start with the second binary digit followed by the first binary digit repeated two times(i.e.,100). Thus we continue to count as 100, 101, 110 and 111. Now again we have exhausted all the three-digit combinations and hence to continue counting, we need to form four-digit combinations and this process continues.

Table 2-1 shows a binary count of zero through fifteen . Notice the pattern with which the 1s and 0s alternate (or toggle) in each column. Thus in the column with position value of 2^0 (*i.e.*, the right most column), the 1s and 0s toggle each time we go to the next count. However, in the column with position value of 2^{+1}, the 1s and 0s toggle after two counts. Similarly in the columns with position value of 2^{+2} and 2^{+3}, the 1s and 0s toggle after four and eight counts respectively.

Table 2-1. Counting in Binary

Decimal Number	*Binary number* 2^{+3}	2^{+2}	2^{+1}	2^0
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

It may be noted from the table 2-1 that, we need four bits or digits to count from zero to 15 (*i.e.*, 2^4–1). In general with n bits, we can count up to a number equal to $2^n - 1$. Thus,

$$\text{Largest decimal number } = 2^n - 1$$

For example, if $n = 5$, then we can count from zero to $(2^5 - 1) = 32 - 1 = 31$. Similarly, if $n = 6$, we can count from zero to $(2^6 - 1) = 64 - 1 = 63$.

Note: In order to avoid confusion among the different number systems, we will add subscript 2 to the number represented in the binary system and subscript 10 to the numbers represented in the decimal system. For example, the number 1101.101_2 is a binary number and $16.538\ _{10}$ is a decimal number. Another way is to write B for binary and D for decimal at the end of the numbers. For example, 1010B and 526D are the binary and decimal numbers respectively.

Example 2-1. *What is the largest decimal value that can be represented by* (*a*) *a* 8-*bit binary,* (*b*) *a* 16 *bit binary number*?

Solution.

(*a*) *Largest decimal number for a* 8-*bit number*

We know that the largest decimal value for a 8-bit binary number,

$$= 2^8 - 1 = 256 - 1 = 255 \textbf{ Ans.}$$

(*b*) *Largest decimal number for a* 16-*bit number*

We know that the largest decimal value for a 16-bit binary number,

$$= 2^{16} - 1 = 65536 - 1 = 65535 \textbf{ Ans.}$$

2-7. Binary-to-Decimal Conversion

We have already discussed the representation of numbers in decimal and binary number system. In order to understand the binary-to-decimal conversion, we will study the conversion of integer (or whole) numbers and the fractional numbers separately in the following pages.

2-8. Integer Binary-to-Decimal Conversion

Following is the procedure for converting an integer (or whole) binary number to its equivalent decimal number.

Step 1. Write the binary number.

Step 2. Directly under the binary number, write the position values or weights of each bit working from right to left.

Step 3. If a zero appears in a digit position, cross-out the weight for that position.

Step 4. Add the remaining weights to obtain the decimal equivalent.

For example, let us consider the conversion of binary 101 to its decimal equivalent.

Step 1.	1	0	1
	↓	↓	↓
Step 2.	2^2	2^1	2^0
Step 3.	2^2	$\cancel{2^1}$	2^0

Step 4. $2^2 + 2^0 = 4 + 1 = 5_{10}$ **Ans.**

Consider another example of converting binary 10101 to its decimal equivalent.

Step 1.	1	0	1	0	1
	↓	↓	↓	↓	↓
Step 2.	2^4	2^3	2^2	2^1	2^0
Step 3.	2^4	$\cancel{2^3}$	2^2	$\cancel{2^1}$	2^0

Step 4. $2^4 + 2^2 + 2^0 = 16 + 4 + 1 = 21_{10}$ **Ans.**

Example 2-2. *Convert the binary number* 01010110_2 *to its equivalent decimal number.*

Solution. Given the binary number = 01010110_2

Step 1.	0	1	0	1	0	1	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
Step 2.	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Step 3.	$\cancel{2^7}$	2^6	$\cancel{2^5}$	2^4	$\cancel{2^3}$	2^2	2^1	$\cancel{2^0}$

Step 4. $2^6 + 2^4 + 2^2 + 2^1 = 64 + 16 + 4 + 2 = 86_{10}$ **Ans.**

2-9. Fractional Binary-to-Decimal Conversion

We have already discussed the integer binary-to-decimal conversion. Let us now see how a binary fraction can be converted into corresponding decimal equivalents. Consider for example the conversion of fractional binary number 0.1010 to decimal.

Step 1.	0 .	1	0	1	0
		↓	↓	↓	↓
Step 2.		2^{-1}	2^{-2}	2^{-3}	2^{-4}
Step 3.		2^{-1}	$\cancel{2^{-2}}$	2^{-3}	$\cancel{2^{-4}}$

Step 4. $2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625_{10}$ **Ans.**

Example 2-3. *Convert the binary number* 0.1100 *to its equivalent decimal number.*

Solution. Given: the binary number = 0.1100

Step 1.	0 .	1	1	0	0
		↓	↓	↓	↓
Step 2.		2^{-1}	2^{-2}	2^{-3}	2^{-4}
Step 3.		2^{-1}	2^{-2}	$\cancel{2^{-3}}$	$\cancel{2^{-4}}$

Step 4. $2^{-1} + 2^{-2} = 0.5 + 0.25 = 0.75$

$\therefore$ $0.1100_2 = 0.75_{10}$ **Ans.**

Example 2-4. *Convert the binary number* 1101.0110 *to its equivalent decimal number.*

(*Anna University Nov./Dec. 2007*)

Solution. Given : the binary number = 110.0110

Step 1.	1	1	0	1	.	0	1	1	0
	↓	↓	↓	↓		↓	↓	↓	↓
Step 2.	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
Step 3.	2^3	2^2	$\cancel{2^1}$	2^0		$\cancel{2^{-1}}$	2^{-2}	2^{-3}	$\cancel{2^{-4}}$

Step 4. $2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = 8 + 4 + 1 + 0.25 + 0.125 = 9.375$

$\therefore$ $1101.0110_2 = 9.375_{10}$ **Ans.**

Example 2-5. *Find the decimal forms for the two binary numbers:* 1101_2 *and* 10111_2

(*Gauhati: University 2007*)

Solution. Given: the binary number = 1101 Binary number 1101_2

Step 1.	1	1	0	1
	↓	↓	↓	↓
Step 2.	2^3	2^2	2^1	2^0
Step 3.	2^3	2^2	$\cancel{2^1}$	2^0

$$\begin{array}{ll} \text{Step 4.} & 2^3 + 2^2 + 2^0 \\ & = 8 + 4 + 1 \\ & = 13_{10} \end{array}$$

$\therefore$ $1101_2 = 13_{10}$ **Ans.**

Binary number 10111_2

Given: the binary number 10111

$$\begin{array}{llccccc} \text{Step 1.} & & 1 & 0 & 1 & 1 & 1 \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \\ \text{Step 3.} & & 2^4 & \cancel{2^3} & 2^2 & 2^1 & 2^0 \end{array}$$

$$\begin{array}{ll} \text{Step 4.} & 2^4 + 2^2 + 2^1 + 2^0 \\ & = 16 + 4 + 2 + 1 \\ & = 23 \end{array}$$

$\therefore$ $10111_2 = 23_{10}$ **Ans.**

Example 2-6. *Convert* $(1101)_2$ *to decimal number.* (*GBTU/MTU. 2007*)

Solution. Given: the binary number $= 1101_2$

$$\begin{array}{llcccc} \text{Step 1.} & & 1 & 1 & 0 & 1 \\ & & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & & 2^3 & 2^2 & 2^1 & 2^0 \\ \\ \text{Step 3.} & & 2^3 & 2^2 & \cancel{2^1} & 2^0 \end{array}$$

Step 4. $2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$ **Ans.**

Example 2-7. *Convert the binary number* 11011110 *into the decimal equivalent.*

(*Anna University, May /June 2007*)

Solution. Given: the binary number $= 11011110_2$

$$\begin{array}{llcccccccc} \text{Step 1.} & & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \\ \text{Step 3.} & & 2^7 & 2^6 & \cancel{2^5} & 2^4 & 2^3 & 2^2 & 2^1 & \cancel{2^0} \end{array}$$

Step 4. $2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1$

$= 128 + 64 + 16 + 8 + 4 + 2$

$= 222_{10}$ **Ans.**

Example 2-8. *Determine the decimal value of the fractional binary number 0.1011.*

(*Anna University, Nov/Dec. 2006*)

Solution. Given: the binary number 0.1011

$$\begin{array}{llllll}
\text{Step 1.} & 0. & 1 & 0 & 1 & 1 \\
 & & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\
 & & & & & \\
\text{Step 3.} & & 2^{-1} & \cancel{2^{-2}} & 2^{-3} & 2^{-4}
\end{array}$$

Step 4. $2^{-1} + 2^{-3} + 2^{-4} = 0.5 + 0.125 + 0.0625$

$= 0.6875$

$\therefore \quad 0.1011_2 = 0.6875_{10}$ **Ans.**

Example 2-9. *Determine the decimal number represented by 101101.10101.*

(*Anna University, April/May 2008*)

Solution. Given: the binary number 101101.10101

$$\begin{array}{llllllllllll}
\text{Step 1.} & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\
 & & & & & & & & & & & \\
\text{Step 3.} & 2^3 & \cancel{2^4} & 2^3 & 2^2 & \cancel{2^1} & 2^0 & 2^{-1} & \cancel{2^{-2}} & 2^{-3} & \cancel{2^{-4}} & 2^{-5}
\end{array}$$

Step 4. $2^5 + 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-3} + 2^{-5}$

$= 32 + 8 + 4 + 1 + 0.5 + 0.125 + 0.03125$

$= 45.65625$

$\therefore \quad 101101 \cdot 10101_2 = 45.65625_{10}$ **Ans.**

2-10. Decimal-to-Binary Conversion

The conversion from decimal-to-binary is usually performed by a digital computer for ease of interpretation by the person reading the number. On the other hand, when a person enters a decimal number into a digital computer, that number must be converted to binary before it can be operated on. There are two methods of decimal-to-binary conversion:

(1) Sum-of-weights method and

(2) Repeated division by-2 method

Both these methods can be used for converting the integer and fractional decimal numbers. The methods are discussed below one by one in the following pages.

2-11. Sum-of-weights method for Integer Decimal-to-Binary Conversion

This method makes use of binary weights. To find the binary number that is equivalent to a given decimal number is to determine the set of binary weights whose sum is equal to the decimal number. An easy way to remember binary weights is that the lowest weight is 1 (*i.e.*, 2^0) . By doubling this weight, we get the next higher weight. Thus a list of eight binary weights would be (read from right to left):

128, 64, 32, 16, 8, 4, 2, 1

Consider, for example, the conversion of decimal number to its binary equivalent using sum-of-weights method. Then we know that, the number 9 can be expressed as the sum of binary weights as follows:

$$9 = 8 + 1 \qquad \text{or} \qquad 9 = 2^3 + 2^1$$

Placing 1s in the appropriate weight positions, 2^3 and 2^0 and 0s in the 2^2 and 2^1 positions, we can write the binary number for decimal 9 as:

$$\begin{array}{cccc} 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 1 \end{array}$$

$\therefore \quad 9_{10} = 1001_2$

Example 2-10. *Convert each of the following decimal numbers to their binary equivalents using sum-of-weights methods*:

(*a*) 17 (*b*) 24 (*c*) 61 (*d*) 93

Solution.

(*a*) Given : The decimal number = 17

We know that the decimal number can be expressed by the sum-of-weights as follows:

$$17 = 16 + 1 = 2^4 + 2^0$$

Placing 1s in the appropriate weight positions, 2^4 and 2^0 and 0s in the 2^3, 2^2 and 2^1 positions, determines the binary number for decimal 17. Thus,

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

$\therefore \quad 17_{10} = 10001_2$ **Ans.**

(*b*) Given : The decimal number = 24

We know that the decimal number,

$$24 = 16 + 8 = 2^4 + 2^3$$

Placing 1s in the appropriate weight positions, 2^4 and 2^3 and 0s in the 2^2, 2^1 and 2^0 positions, determines the binary number for decimal 24. Thus,

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

$\therefore$ $24_{10} = 11000_2$ **Ans.**

(*c*) Given : The decimal number = 61

We know that the decimal number,

$$61 = 32 + 16 + 8 + 4 + 1$$

Placing 1s in the appropriate weight positions, 2^5, 2^4, 2^3, 2^2 and 2^0 and 0s in the 2^1 position determines the binary number for decimal 61. Thus

$$\begin{array}{cccccc} 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1 & 1 & 0 & 1 \end{array}$$

$\therefore$ $61_{10} = 111101_2$ **Ans.**

(*d*) Given : The decimal number = 93

We know that the decimal number,

$$93 = 64 + 16 + 8 + 4 + 1$$

Placing 1s in the appropriate weight positions 2^6, 2^4, 2^3, 2^2 and 2^0 and 0s in the 2^5 and 2^1 position determines the binary number for decimal number 93. Thus;

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

$\therefore$ $93_{10} = 1011101_2$ **Ans.**

2-12. Sum-of-weights method for Conversion of Fractional Decimal-to-Binary

This method makes use of position weights. In order to find a binary fractional number that is equivalent to a given decimal fractional number, we have to determine the set of binary weights whose sum is equal to the decimal number. An easy way to remember fractional binary weights is that the most significant weight is 0.5 (2^{-1}) and that by dividing by 2, we get the next lower weight. Thus a list of four fractional binary weights would be: 0.5, 0.25, 0.125 and 0.0625.

Consider for example, the conversion of the decimal fraction 0.625 to its equivalent binary. We know that,

$$\begin{aligned} 0.625 &= 0.5 + 0.125 \\ &= 2^{-1} + 2^{-2} \\ &= 0.101 \end{aligned}$$

There is a 1 in the 2^{-1} position, a 0 in the 2^{-2} position and a 1 in the 2^{-3} position.

Example 2-11. *Convert the decimal fraction* 0.375 *by using sum-of-weights method to its equivalent binary fraction.*

Solution. Given : The decimal fraction = 0.375

We know that the decimal fraction,

$$0.375 = 0.25 + 0.125$$
$$= 2^{-2} + 2^{-3} = 0.011_2$$

There is a 0 in the 2^{-1} position, a 1 in the 2^{-2}, and 2^{-3} positions.

Thus $0.375_{10} = 0.011_2$ **Ans.**

2-13. Repeated Division-by-2 Method for Integer Decimal-to-Binary Conversion

It is a systematic way of converting integer numbers from decimal-to-binary. The procedure for conversion is as given below:

Step 1. Begin by dividing the given decimal number by 2.

Step 2. Devide each resulting quotient by 2 until there is a 0 whole number quotient.

Step 3. The remainders generated by each division form the binary number. The first remainder to be produced is the least significant bit (LSB) in the binary number, and the last remainder to be produced is the most significant bit (MSB). In other words, reading the remainders from bottom-to-top constitutes the required binary number.

In order to illustrate the procedure, let us consider the conversion of decimal 10 to its equivalent binary number.

Top ↑

$10 \div 2 = 5$	with a remainder	0	(LSB)
$5 \div 2 = 2$	with a remainder	1	
$2 \div 2 = 1$	with a remainder	0	
$1 \div 2 = 0$	with a remainder	1	(MSB)

Bottom

Reading the remainders from bottom to top, we find that the binary number for the given decimal number 10 is 1010.

$\therefore$ $10_{10} = 1010_2$

Example 2-12. *Convert each of the following decimal numbers using repeated-division by-*2 *method.*

(*a*) 19 *and* (*b*) 45

Solution.

(*a*) Given the decimal number = 19.

We know that the conversion from decimal-to-binary can be done by using repeated-division-by 2 method.

Top ↑

$19 \div 2 = 9$	with a remainder	1	(LSB)
$9 \div 2 = 4$	with a remainder	1	
$4 \div 2 = 2$	with a remainder	0	
$2 \div 2 = 1$	with a remainder	0	
$1 \div 2 = 0$	with a remainder	1	(MSB)

Bottom

Reading the remainders from bottom to top, the binary number for the decimal number 19 is 10011.

$\therefore$ $\qquad 19_{10} = 10011_2$ **Ans.**

(*b*) Given the decimal number = 45

We know that the conversion from decimal-to-binary can be done by using repeated division-by-2 method.

			Top
$45 \div 2 = 22$	with a remainder	1	↑ (LSB)
$22 \div 2 = 11$	with a remainder	0	
$11 \div 2 = 5$	with a remainder	1	
$5 \div 2 = 2$	with a remainder	1	
$2 \div 2 = 1$	with a remainder	0	
$1 \div 2 = 0$	with a remainder	1	(MSB)
			Bottom

Reading the remainders from bottom to top, the binary number for the decimal number 45 is 101101_2.

$\therefore$ $\qquad 45_{10} = 101101_2$ **Ans.**

Note that most computers or digital systems deal with groups of 4, 8, 16, or 32 bits, therefore we should keep all our answers in that form. Thus by adding leading zeros to the binary number 101101 will not change its numeric value. Therefore the 8-bit answer is:

$$101101_2 = 00101101_2$$

2-14. Repeated Multiplication-by-2 for Fractional Decimal to Binary Conversion

We have already seen in the last article that the repeated division-by-2 method can be used to convert integer (or whole number) decimal to its equivalent binary. Decimal fractions can be converted to binary by repeated multiplication-by-2. Following is the procedure that is used for conversion:

Step 1. Begin by multiplying the given decimal fraction by 2 and then multiplying each resulting fractional part of the product by 2.

Step 2. Repeat step 1 until the fractional product is zero or until the desired number of decimal places is reached.

Step 3. The carried bits or carries generated by the multiplication produce the binary number. The first carry produced is the most-significant bit (MSB) and the last carry is the least significant bit (LSB). In other words, reading from top to bottom constit-utes the required fractional binary.

In order to illustrate the procedure, consider the conversion of 0.3125 to its equivalent binary fraction.

				Top
$0.3125 \times 2 = 0.625$	with a carry 0			MSB
$0.625 \times 2 = 1.25$	$= 0.25$	with a carry	1	
$0.25 \times 2 = 0.50$	$= 0.50$	with a carry	0	
$0.50 \times 2 = 1.00$	$= 0$	with a carry	1	↓ LSB
				Bottom

Since the fractional part is zero, so we will stop the repeated multiplication. Reading from top to bottom, the required binary number is 0101.

$\therefore \quad 0.3125_{10} = 0.101_2$

Example 2-13. *Convert the decimal fraction* 0.9028 *to its equivalent binary fraction* (*up to* 4 *binary places*) *using repeated multiplication-by-*2 *method.*

Solution. Given : The decimal fraction = 0.9028

Top

9028×2	$= 1.8056$	$= 0.8056$	with a carry of	1	LSB
0.8056×2	$= 1.6112$	$= 0.6112$	with a carry of	1	
0.6112×2	$= 1.2224$	$= 0.2224$	with a carry of	1	
0.2224×2	$= 0.4448$	$= 0.4448$	with a carry of	0	MSB

Bottom

and so on.........

We can continue this process further if we wish because the binary fractional part has not reduced to zero. However we stop here because the desired number of binary places is reached. Reading from top to bottom, we find that the binary fraction is 0.1110.

$\therefore \quad 0.9028_{10} = 0.1110_2$ **Ans.**

Example 2-14. *Obtain the binary equivalent of* 15 *and* 25. (*Gauhati University, 2007*)

Solution. Binary equivalent of 15

Given: The decimal number = 15

We know that the decimal number can be expressed by the sum-of-weight as follows.

$$\begin{aligned} 15 &= 8 + 4 + 2 + 1 \\ &= 2^3 + 2^2 + 2^1 + 2^0 \end{aligned}$$

Placing 1s in the appropriate positions, 2^3, 2^2, 2^1 and 2^0 position, Thus

$$\begin{array}{cccc} 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1 & 1 \end{array}$$

$\therefore \quad 15_{10} = 1111_2$ **Ans.**

Binary equivalent of 25

Given: The binary number = 25.

We know that the decimal number can be expressed by the sum-of-weight as follow.

$$\begin{aligned} 25 &= 16 + 8 + 1 \\ &= 2^4 + 2^3 + 2^0 \end{aligned}$$

Placing 1s in the appropriate weight positions 2^4,2^3 and 2^0 and 0s in the 2^2 and 2^1 positions determines the binary number for decimal 25, Thus

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

$\therefore \quad 25_{10} = 11001_2$ **Ans.**

Example 2-15. *Convert the decimal 0.8125 to its binary equivalent.*

(*Gujarat Technological University, Dec 2009*)

Solution. Given: The decimal number 0.8127

	Top	
$0.8127 \times 2 = 1.6254 = 0.6254$ with a carry of	1	LSB
$0.6254 \times 2 = 1.2508 = 0.2508$ with a carry of	1	
$0.2508 \times 2 = 0.5016 = 0.5016$ with a carry of	0	
$0.5016 \times 2 = 1.0032 = 0.0032$ with a carry of	1	MSB
$0.0032 \times 2 = 0.0064 = 0.0064$ with a carry of	0 ↓	

and so on…

We can continue this process further if we with because the binary fractional part has not reduced to zero. However we stop here because the desired number of binary places reached. Reading from top to bottom, we find that the binary fraction is 0.11010.

$\therefore \quad 0.8125_{10} = 0.11010_{2.}$ **Ans.**

2-15. Octal Number System

The octal number system provides a convenient way to express binary numbers and codes. This system is composed of eight digits 0, 1, 2, 3, 4, 5, 6, and 7. After reaching 7 we have finished with the digits. To count beyond 7, we form two digit combinations in the same way as discussed for decimal and binary number system . Thus beyond 7, we count as 10, 11, 12, 13, 14, 15, 16, 17, 20, 21…and so on.

It may be noted that counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used. To distinguish octal numbers from decimal numbers, we will use the subscript 8 to indicate the octal number. For example, 148 is in octal is equivalent to 12 in decimal.

With N octal digits, we can count from 0 up to $(8^N – 1)$, for a total of 8^N different counts. For example with 2 octal digit positions we can count from 00_8 to 77_8 which is 0_{10} to 63_{10} for a total of $8^2 = 64_{10}$ different octal numbers. Similarly with three octal digit positions, we can count from 000_8 to 777_8 which is 0_{10} to 511_{10} for a total of $8^3 = 512_{10}$ different octal numbers. Fig 2-4 shows the position weight of each digit in an octal number.

$$8^{+5} \quad 8^{+4} \quad 8^{+3} \quad 8^{+2} \quad 8^{+1} \quad . \quad 8^{-1} \quad 8^{-2} \quad 8^{-3} \quad 8^{-4}$$

↑ Octal point

Fig. 2-4 *Position weights in an octal number.*

Table 2-2 shows the binary and octal numbers corresponding to first ten decimal numbers.

Table 2-2. Binary and Octal Numbers corresponding to first ten decimal numbers

Decimal	*Binary*	*Octal*
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	10
9	1001	11
10	1010	12

Please note that to signify an octal number, a subscript 8 or the letter O is used (*i.e.*, 17_8 or 17 O). However the use of subscript 8 is more common in actual practice.

2-16. Octal-to-Decimal Conversion

The octal-to-decimal conversion is a useful tool in the field of digital electronics (or computers). Let us study how the conversion from octal to decimal is done. Since the octal number system has a base of eight, each successive digit has a position weight of 8^0, the next higher digit has a position weight of 8^1, the next of 8^2 and so on. For a fractional octal, the first digit to the right of octal point, has a position weight of 8^{-1}, the next has a position weight of 8^{-2}, the next 8^{-3} and so on.

Following is the procedure to convert an octal number to its decimal equivalent.

Step 1. Write the octal number.

Step 2. Directly under the octal number write the position weight of each digit working from right to left.

Step 3. Multiply each octal digit by its position weight and take sum of the products.

In order to illustrate the method, consider an example, of converting 23748.

The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, consider for example, the conversion of octal number 2374 to its equivalent decimal. The conversion is shown below:

Step 1.	2	3	7	4
	↓	↓	↓	↓
Step 2.	8^{+3}	8^{+2}	8^{+1}	8^0

Step 3.
$$\begin{aligned} &= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0) \\ &= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1) \\ &= 1024 + 192 + 56 + 4 \\ &= 1276_{10} \end{aligned}$$

$\therefore \quad 2374_8 = 1276_{10}$

Consider another example of converting 372.6 to its equivalent decimal. Then

Step 1.	3	7	2	6
	↓	↓	↓	↓
Step 2.	8^{+3}	8^{+2}	8^{+1}	8^0

Step 3 $= (3 \times 8^{+3}) + (7 \times 8^{+2}) + (2 \times 8^{+1}) + (6 \times 8^{0})$

$= (3 \times 64) + (7 \times 8) + (2 \times 1) + (6 \times 0.125)$

$= 192 + 56 + 2 + 0.750 = 250.75$

$\therefore \quad 3726_8 = 250.75_{10}$

Example 2-16. *Convert the number* 326_8 *to its equivalent decimal number.*

(*Anna University, Nov./Dec. 2007*)

Solution. Given the octal number = 326.

We know that, in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{llll} \text{Step 1.} & 3 & 2 & 6 \\ & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 8^{+2} & 8^{+1} & 8^{0} \end{array}$$

Step 3. $= (3 \times 8^{+2}) + (2 \times 8^{+1}) + (6 \times 8^{0})$

$= (3 \times 64) + (2 \times 8) + (6 \times 1)$

$= 192 + 16 + 6$

$= 214_{10}$

$\therefore \quad 326_8 = 214_{10}$ **Ans.**

2-17. Decimal-to-Octal Conversion

We have already discussed in the last article, the method of converting the octal number to its decimal equivalent. Now we shall study the conversion of decimal-to-octal conversion. In order to make the study more effective we will consider the conversion of integer and fractional numbers separately.

2-18. Integer Decimal-to-Octal Conversion

In order to convert an integer decimal number to its equivalent octal number, we use a repeated division by-8 method.

Step 1. Begin by dividing the given decimal number by 8.

Step 2. Divide each resulting quotient by 8 until there is a zero whole number quotient.

Step 3. The remainders are generated by each division from the octal number. The first remainder to be produced is the least significant digit (LSD) in the octal number and the last remainder to be produced is the most significant digit (MSD). In other words, reading the remainders from bottom-to-top constitutes the octal number.

If a calculator is being used to perform the divisions in the conversion process, the results will include a decimal fraction of the remainder. The remainder can be obtained by multiplying the fraction by 8. Let us illustrate the procedure by considering the conversion of 266_8 to its decimal equivalent.

$$\begin{array}{lllll} & & & \text{Top} & \\ 266 \div 8 = 33.25 \text{ with a remainder} & 0.25 & \rightarrow 0.25 \times 8 = 2 & \uparrow & \text{LSD} \\ 33 \div 8 = 4.125 \text{ with a remainder} & 0.125 & \rightarrow 0.125 \times 8 = 1 & | & \\ 4 \div 8 = 0.5 \;\; = 0 \;\; \text{with remainder} & 0.5 & \rightarrow 0.5 \times 4 = 1 & | & \text{MSD} \\ & & & \text{Bottom} & \end{array}$$

Since the quotient is zero, therefore we will stop the division by 8. Reading the remainders from bottom to top, we find that the octal equivalent of the decimal number 266 is 112.

2-19. Fractional Decimal to Octal Conversion

In order to convert a fractional decimal number to an equivalent octal number, we use a repeated multiplication-by-8-method.

Step 1. Begin by multiplying the given decimal fraction by 8 and thus multiplying each resulting fractional part of the product by 2.

Step 2. Repeat step 1 until the fractional product is zero with the desired number of Decimal places is reached.

Step 3. The carried digits or carries generated by the multiplication produce the octal number.

For example consider the conversion of fraction 0.25 to its octal equivalent. The first carry produced is the most-significant digit (MSD). In other words reading from top to bottom constitutes the required fractional octal.

In order to illustrate the procedure, consider the conversion of 0.3125 to its equivalent octal.

Top

$0.3125 \times 8 = 2.5 = 0.5$ with a carry 2 MSD

$0.5 \times 8 = 4.0 = 0$ with a carry 4 ↓ LSD

Bottom

Since the fractional part is zero, so we will stop the repeated multiplication, Reading from top to bottom, the required octal number is 24.

$\therefore \quad 0.3125_{10} = 0.24_8$

2-20. Octal-to Binary Conversion

The primary advantage of the octal number system is the case with which conversion can be made between binary and octal numbers. The conversion from octal-to-binary is performed by converting each octal digit to its 3-bit binary equivalent. Table 2.3 shows the 3-bit binary equivalent for the octal digits 0,1, 2, ...7.

Table 2-3. Octal-to-binary conversion

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

Using Table 2-3, we can convert any octal number to binary by individually converting each digit. Consider, for example, the conversion of 13 to its equivalent binary. From the Table 2-3, we find that octal digit 1 is equivalent to 001 and octal digit 3 is equivalent to 011. Thus

$$\begin{array}{cc} 1 & 3 \\ \downarrow & \downarrow \\ \overbrace{001} & \overbrace{011} \end{array}$$

$\therefore$ $13_8 = 001011_2$ or simply 1001_2.

Similarly, suppose we need to convert the octal 526_8 to its binary equivalent. Again from the table 2-4, we find that the octal digit 5 is equivalent to binary 101, the octal digit 2 is binary 010 and octal digit 6 is 110. Thus,

$$\begin{array}{ccc} 5 & 2 & 6 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{101} & \overbrace{010} & \overbrace{110} \end{array}$$

Hence the binary equivalent for the octal 526_8 is 101010110_2

$\therefore$ $526_8 = 101010110_2$

Example 2-17. *Convert the following octal numbers to their binary equivalent.*

(*a*) 321_8 (*b*) 4653_8 (*c*) 13274_8

Solution.

(*a*) Given: The octal number = 321

We know that to convert the given octal number to its binary equivalent, we have to convert each octal digit to its 3-bit binary equivalent. Thus

$$\begin{array}{ccc} 3 & 2 & 1 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{011} & \overbrace{010} & \overbrace{001} \end{array}$$

Thus the octal 321_8 is equivalent to the binary 011010001_2.

Dropping off the zero at the left most position, the binary number can be written as 11010001.

$\therefore$ $321_8 = 1101000_2$ **Ans.**

(*b*) Given: The octal number = 4653_8

We know that to convert the given octal number to its binary equivalent, we have to convert each octal digit to its 3-bit binary equivalent. Thus

$$\begin{array}{cccc} 4 & 6 & 5 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \overbrace{100} & \overbrace{110} & \overbrace{101} & \overbrace{011} \end{array}$$

Thus the octal 4653_8 is equivalent to the binary 100110101011_2.

$\therefore$ $4653_8 = 100110101011_2$ **Ans.**

(*c*) Given : The octal number = 13274_8

We know that to convert the given octal number to its binary equivalent, we have to convert each octal digit by its 3-bit binary equivalent. Thus,

$$\begin{array}{ccccc}
1 & 3 & 2 & 7 & 4 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\overbrace{001} & \overbrace{011} & \overbrace{010} & \overbrace{111} & \overbrace{100}
\end{array}$$

Thus the octal number 13274 is equivalent to binary 001011010111100.

Dropping off the zero at the left-most position, the resulting binary number can be written as 1011010111100.

$\therefore$ $13274_8 = 1011010111100_2$ **Ans.**

Example 2-18. *Convert each of the following decimal numbers to octal equivalent by repeated diversion-by -8 method.*

(*a*) *142* (*b*) *221* (*c*) *435*

Solution.

(*a*) Given : The decimal number = 142

We know that in order to convert the given number 142_{10} to its octal equivalent, we have to use the repeated diversion-by-8 method, thus

$$\begin{array}{rllllcl}
 & & & & & \text{Top} & \\
142 \div 8 = 17.75 = 17 & \text{with a remainder} & 0.75 & \rightarrow & 0.75 \times 8 = 6 & \uparrow & \text{LSD} \\
17 \div 8 = 2.125 = 2 & \text{with a remainder} & 0.105 & \rightarrow & 0.125 \times 8 = 1 & | & \\
2 \div 8 = 0.25 = 0 & \text{with a remainder} & 0.25 & \rightarrow & 0.25 \times 8 = 2 & | & \text{MSD} \\
 & & & & & \text{Bottom} &
\end{array}$$

Since the quotient is zero, therefore we will stop the diversion-by-8. Reading the remainders from bottom to top, we find that the octal equivalent of the decimal number 142 is 216.

$\therefore$ $142_{10} = 216_8$ **Ans.**

(*b*) Given: The decimal number = 221

We know that in order to correct the given number 221_{10} to its octal equivalent, we have to use the repeated diversion-8 method. Thus

$$\begin{array}{rllllcl}
 & & & & & \text{Top} & \\
221 \div 8 = 27.625 = 27 & \text{with a remainder} & 0.625 & \rightarrow & 0.625 \times 8 = 5 & \uparrow & \text{LSD} \\
27 \div 8 = 3.375 = 3 & \text{with a remainder} & 0.375 & \rightarrow & 0.375 \times 8 = 3 & | & \\
3 \div 8 = 0.375 = 0 & \text{with a remainder} & 0.375 & \rightarrow & 0.375 \times 8 = 3 & | & \text{MSD} \\
 & & & & & \text{Bottom} &
\end{array}$$

Since the quotient is zero, therefore we will stop the diversion by 8. Reading the remainders from bottom to top, we find that the octal equivalent of 221 is 325.

$\therefore$ $221_{10} = 335_8$ **Ans.**

(*c*) Given: The decimal number = 435

$$\begin{array}{rllllcl}
 & & & & & \text{Top} & \\
435 \div 8 = 54.375 = 54 & \text{with a remainder} & 0.375 & \rightarrow & 0.375 \times 8 = 3 & \uparrow & \text{LSD} \\
54 \div 8 = 6.75 = 6 & \text{with a remainder} & 0.75 & \rightarrow & 0.75 \times 8 = 6 & | & \\
6 \div 8 = 0.75 = 0 & \text{with a remainder} & 0.75 & \rightarrow & 0.75 \times 8 = 6 & | & \text{MSD} \\
 & & & & & \text{Bottom} &
\end{array}$$

Since the quotient is zero, therefore we will stop the diversion by 8. Reading the remainders from bottom to top, we find the octal equivalent of 435 is 663.

∴ $435_{10} = 663_8$ **Ans.**

Example 2-19. *Convert each of the following octal numbers to their equivalent binary numbers.*

(*a*) *624* (*b*) *326* (*c*) *476*

Solution.

(*a*) Given : The octal number = 624

We know that to convert the given octal number to its binary equivalent, we have to convert each octal digit to its binary equivalent . Thus,

$$\begin{array}{ccc} 6 & 2 & 4 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{110} & \overbrace{010} & \overbrace{100} \end{array}$$

∴ $624_8 = 110010100_{10}$ **Ans.**

(*b*) Given : The octal number = 326

We know that to convert the given octal number to its binary equivalent we have to convert each octal digit to its binary equivalent. Thus,

$$\begin{array}{ccc} 3 & 2 & 6 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{011} & \overbrace{010} & \overbrace{110} \end{array}$$

∴ $326_8 = 011010110_2$ **Ans.**

(*c*) Given : The octal number = 476

$$\begin{array}{ccc} 4 & 7 & 6 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{100} & \overbrace{111} & \overbrace{110} \end{array}$$

∴ $476_8 = 100111110_2$ **Ans.**

Example 2-20. *Convert the decimal number* 2497.50_{10} *to its equivalent octal.*

(*Anna University, Nov./Dec. 2007*)

Solution. Given : The decimal number 2497.50.

We know that we can convert the integer part of the given number to its equivalent octal by the repeated division by 8 method. Whereas the fractional part is to be converted to its equivalent octal by the repeated multiplication-by-8 method.

Conversion of integer part

								Top	
2497 ÷ 8	= 312.13	= 312	with a remainder	0.13	→	0.13 × 8	= 1	↑	LSD
312 ÷ 8	= 39.00	= 39	with a remainder	0	→	0 × 8	= 0	│	
39 ÷ 8	= 4.88	= 4	with a remainder	0.88	→	0.88 × 8	= 7	│	
4 ÷ 8	= 0.5	= 0	with a remainder	0.5	→	0.5 × 8	= 4	│	MSD
								Bottom	

∴ $2497_{10} = 4701_8$

Conversion of fractional part

$$0.50 \times 8 = 4.00 = 0 \text{ with a carry } 4$$

Since the fractional part is zero, so we stop the repeated multiplication. Thus,

$$0.5_{10} = 0.4_8$$

Combining the integer and the fractional part of the numbers, we get,

$$2497.50_{10} = 4701.4_8 \textbf{ Ans.}$$

Example 2-21. *Convent decimal number* 214 *to its octal equivalent.*

(*Gujarat Technical University, Dec 2009*)

Solution. Given: The decimal number 214

We know that in order to correct the given number 214_{10} to its octal equivalent we have to use the repeated division 8 method. Thus.

Top

$214 \div 8 = 26.75 = 26$ with a remainder $0.75 \rightarrow 8 = 6$ $\uparrow$ LSD

$26 \div 8 = 3.205 = 3$ with a remainder $0.25 \rightarrow 8 = 2$

$3 \div 8 = 0.375 = 0$ with a remainder $0.375 \rightarrow 8 = 3$ LSD

Bottom

Since the quotient is zero, therefore we will stop the division by 8. Reading the remainder from bottom to top, we find that the octal equivalent of 214 is 326

$\therefore$ $214_{10} = 326_8$ **Ans**.

Example 2-22. *Convert* $(0.513)_{10}$ *to octal* (*Anna University, Nov/Dec 2007*)

Solution. Given: The decimal number = 0.513

$0.513 \times 8 = 4.104 = 0.104$ with a carry 4 MSB

$0.104 \times 8 = 0.832 = 0.832$ with a carry 0

$0.832 \times 8 = 6.656 = 0.656$ with a carry 6 $\downarrow$ LSB

$(0.513)_{10} = (0.406)_8$ **Ans.**

Example 2-23. *Covert the decimal number* 39.75 *to octal.* (*PTU., Dec 2009*)

Solution. Given: The decimal number 39.75

We know that we can covert the integer part of the given number of its equivalent octal by the repeated division by 8 method. Whereas the fractional part is to be converted to its equivalent octal by the repeated multiplication by 8 method.

Conversion of integer part Top

$39 \div 8 = 4.875 = 4$ with a remainder $0.875 \rightarrow 0.875 \times 8 = 7$ $\uparrow$

$4 \div 8 = 0.5 = 0$ with a remainder $0.5 \rightarrow 0.5 \times 8 = 4$

$\therefore$ $39_{10} = 47_8$ Bottom

Conversion of fractional part

$0.75 \times 8 = 6.00 = 0$ with a carry 6

Since the fractional part is zero so we stop the repeated multiplication, Thus,

$$0.75_{10} = 0.6_8$$

Combining the integer and the fractional part of the number we get,

$$39.75_{10} = 47.6_8 \textbf{Ans.}$$

2-21. Binary-to-Octal Conversion

Conversion of a binary integer number to an octal integer number is the reverse of the octal-to-binary conversion. Following is the procedure for converting a given binary number to its octal conversion.

In order to illustrate the procedure, consider the conversion of 100111010 to octal.

$$\begin{array}{ccc} \underbrace{100} & \underbrace{111} & \underbrace{010} \\ \downarrow & \downarrow & \downarrow \\ 4 & 7 & 2 \end{array}$$

$\therefore$ $100111010_2 = 472_8$ **Ans.**

Let us consider another example, the conversion of 011010110 of octal.

$$\begin{array}{ccc} \underbrace{011} & \underbrace{010} & \underbrace{110} \\ \downarrow & \downarrow & \downarrow \\ 3 & 2 & 6 \end{array}$$

Note that we formed 3- bit combination while moving right to left by starting from the right most position. For the left-most group, since it does not have the 3-bit combination, so we added one zero from out side to complete the combination.

$\therefore$ $011010110_2 = 326_8$ **Ans.**

Example 2-24. *Convert the number* 10111001_2 *to octal.*

Solution. Given : The binary number = 10111001

We know that in order to convert the given binary number to its equivalent octal, we have to form 3-bit combinations while moving right to left by starting from the right most position.

$$\begin{array}{cccc} & \underbrace{10} & \underbrace{111} & \underbrace{001} \\ \text{adding a leading zero} \longrightarrow & \underbrace{010} & \downarrow & \downarrow \\ & \downarrow & \downarrow & \downarrow \\ & 2 & 7 & 1 \end{array}$$

Note that for the left most group, since it does not have the 3-bit combination, so we have added one zero from out side to complete the combination.

$\therefore$ $10111001_2 = 271_8$ **Ans.**

Example 2-25. *Convert the following binary numbers to their octal equivalents.*

(*a*) *110101111* (*b*) *100110010* (*c*) *10111111001*

Solution.

(*a*) Given : The binary number = 110101111

We know that in order to convert the given binary number to its octal equivalent, we have to form 3-bit combinations while moving right to left by starting from the right most position. Thus,

$$\begin{array}{ccc} \underbrace{110} & \underbrace{101} & \underbrace{111} \\ \downarrow & \downarrow & \downarrow \\ 6 & 5 & 7 \end{array}$$

$\therefore$ $110101111_2 = 657_8$ **Ans.**

(*b*) Given : The binary number = 100110010

We know that in order to convert the given binary number to its octal equivalent, we have to form 3-bit combinations while moving right to left by starting from the right most position. Thus,

$$\begin{array}{ccc} \underbrace{100} & \underbrace{110} & \underbrace{010} \\ \downarrow & \downarrow & \downarrow \\ 4 & 6 & 2 \end{array}$$

$\therefore$ $100110010_2 = 462_8$ **Ans.**

(*c*) Given : The binary number, 10111111001

We know that in order to convert the given binary number to its octal equivalent, we have to form 3-bit combinations while moving right to left by starting from the right most position. Thus,

$$\begin{array}{lcccc} & \underbrace{10} & \underbrace{111} & \underbrace{111} & \underbrace{001} \\ \text{adding a leading zero} \rightarrow & \underbrace{010} & \Bigg\downarrow & \Bigg\downarrow & \Bigg\downarrow \\ & \downarrow & & & \\ & 2 & 7 & 7 & 1 \end{array}$$

Note that for the left most group, since it does not have the 3-bit combination, so we have added one zero from outside to complete the combination.

$\therefore$ $10111111001_2 = 2771_8$ **Ans.**

2-22. Usefulness of Octal System

We have already discussed the octal number system and conversion from the binary and decimal numbers to octal and vice versa. The ease with which conversions can be made between octal and binary makes the octal system attractive as "shorthand" means of expressing large binary numbers. In computer work, binary number with up to 64 bits are not uncommon. These binary numbers, as we shall see, do not always represent a numerical quantity but are often some type of code that conveys non numerical information. In computers, binary numbers might represent:

1. actual numerical data
2. numbers corresponding to a location called (address) in memory
3. an instruction code
4. a code representing alphabetic and other non numerical characters
5. group of bits representing the status of devices internal or external to the computer

When dealing with a large quantity of binary numbers of many bits, it is convenient and more efficient for us to write the numbers in octal rather than binary. However keep in mind that the digital circuits and systems work strictly in binary. We use octal numbers only as a convenience for the operators of the system.

Example 2-26. *Convert the decimal* 486 *to its eight binary equivalent by first converting it to octal.*

Solution. Given : The decimal number = 486

$$\begin{array}{lllcl} & & & \text{Top} & \\ 486 \div 8 = 60 & \text{with a remainder} & 6 & \uparrow & \text{LSB} \\ 60 \div 8 = 7 & \text{with a remainder} & 4 & | & \\ 7 \div 8 = 0 & \text{with a remainder} & 7 & | & \text{MSB} \\ & & & \text{Bottom} & \end{array}$$

Now in order to convert the octal 746_8, to its binary equivalent, we replace each octal digit by its 3-bit equivalent binary number, i.e.

$$\begin{array}{ccc} 7 & 4 & 6 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{111} & \overbrace{100} & \overbrace{110} \end{array}$$

∴ $486_{10} = 111100110_2$ **Ans.**

Example 2-27. *Convert octal 765 to base 2.* (*Nagpur University, 2004*)

Solution. Given: The octal number 765

We know that to convert the given octal number to its equivalent in base2, (i.e binary equivalent), we have to covert each octal digit to its 3-bit equivalent. Thus

$$\begin{array}{ccc} 7 & 6 & 5 \\ \downarrow & \downarrow & \downarrow \\ 111 & 110 & 101 \end{array}$$

Thus the octal 765_8 is equivalent to the binary 111110101_2

∴ $765_8 = 111110101_2$ **Ans.**

2-23. Hexadecimal Number System

The hexadecimal number system like octal number system is used primarily as a "shorthand" way of displaying binary numbers because it is very easy to convert between binary and hexadecimal. As you are probably aware, long binary numbers are difficult to read and write because it is easy to drop or transpose a bit. Since computers and microprocessors understand only 1s and 0s, it is necessary to use these digits when you program in "machine language". Imagine writing a bit instruction for a microprocessor system in 1s and 0s. It is much more efficient to use the hexadecimal or octal. Hexadecimal is frequently used in computer and microprocessor applications.

The hexadecimal system is composed of 16 digits which includes numerals and alphabetic characters. The 16 digits are 0, 1, 2, 3,...7, 8, 9, A, B, C, D, E and F. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient because each hexadecimal digit represent a 4-bit binary number as shown in Table 2-4.

Table 2-4. Binary and hexadecimal numbers for the first 16 decimal numbers

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

Note that to signify a hexadecimal number, a subscript 16 or the letter H is used (i.e., $D7_{16}$ or D7H). The use of subscript is more common while solving tutorial problems whereas the use of letter H is more common while writing computers programs.

In order to count in hexadecimal beyond F, We form two digit combinations by starting with the second digit followed by the first digit, i.e.,

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31......and so on.

With two hexadecimal digits, you can count up to FF_{16} which is equivalent to decimal 255. To count beyond this, we form three digit combinations. For instance 100_{16} is decimal 256, 101_{16} is decimal 257, and so forth. The maximum 4-digit hexadecimal number is $FFFF_{16}$ which is equivalent to decimal 65,535.

2-24. Hexadecimal-to-Decimal Conversion

We have already discussed the representation of numbers in hexadecimal number system. Now we shall study how a hexadecimal integer (or a whole) number can be converted to its equivalent hexadecimal number.

Following is the procedure to convert a hexadecimal number to its equivalent decimal number:

Step 1. Write the hexadecimal number.

Step 2. Directly under the hexadecimal number, write the position weight of each digit working from right to left.

Step 3. Multiply the decimal value of each hexadecimal digit by its position weight and take sum of the products.

In order to illustrate the method, let us take an example, we will convert E5 to its equivalent decimal number,

Step 1. E 5

↓ ↓

Step 2. 16^{+1} 16^{0}

Step 3. $(E \times 16^{+1}) + (5 \times 16^{0})$

$= (E \times 16) + (5 \times 1)$

$= (14 \times 16) + (5 \times 1)$

$= 224 + 5$

$= 229$

$\therefore$ $E5_{16} = 229_{10}$ **Ans.**

Let us take another example of a hexadecimal number 0.12_{16} and convert to its equivalent decimal fraction.

Step 1. 0 . 1 2

↓ ↓

Step 2. 16^{-1} 16^{-2}

Step 3. $1 \times 16^{-1} + 2 \times 16^{-2}$

$= 0.0625 + 0.0078$

$= 0.0703$ **Ans.**

Example 2-28. *Convert the hexadecimal number* $2A6_{16}$ *to its equivalent decimal number.*

Solution. Given : The hexadecimal number = 2 A6

We know that to convert 2A6 to its equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then table the sum of these products.

Step 1. 2 A 6

↓ ↓ ↓

Step 2. 16^{+2} 16^{+1} 16^{+0}

Step 3. $(2 \times 16^{+2}) + (A \times 16^{+1}) + (6 \times 16^{0})$

$= (2 \times 256) + (10 \times 16) + (6 \times 1)$

$= 678$

$\therefore$ $2A6_{16} = 678_{10}$ **Ans.**

Example 2-29. *Convert the hexadecimal number* B2F8 *to its equivalent decimal number.*

Solution. Given : The hexadecimal number = B2F8

We know that to convert B2F8 to its equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then take the sum of these products.

Step 1. B 2 F 8

↓ ↓ ↓ ↓

Step 2. 16^{+3} 16^{+2} 16^{+1} 16^{+0}

Step 3. $(B \times 16^{+3}) + (2 \times 16^{+2)} + (F \times 16^{+1}) + (8 \times 16^{0})$

$= (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1)$

$= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1)$

$= 45056 + 512 + 240 + 8$

$= 45816$

$\therefore$ $B2F8_{16} = 45816_{10}$ **Ans.**

Example 2.30. *Convert the following hexadecimal number to decimal.*

(*a*) $1C_{16}$ (*b*) $A85_{16}$ (*c*) $E5_{16}$ (*d*) $B2F8_{16}$

(*Anna University, May/June 2007*)

Solution.

(*a*) Given : The hexadecimal number 1C

Step 1. 1 C

↓ ↓

Step 2. 16^{+1} 16^{+0}

Step 3. $(1\times 16^{+1}) + (C \times 16^{+0})$

$= (16) + (13 \times 1)$

$= 29$

$\therefore$ $1C_{16} = 29_{10}$ **Ans.**

(*b*) Given : The hexadecimal number $A85_{16}$

We know that to convert A85 to its equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then take the sum of these products.

Step 1. A 8 5

↓ ↓ ↓

Step 2. 16^{+2} 16^{+1} 16^{+0}

Step 3. $(A\times 16^{+2}) + (8 \times 16^{+1}) + (5 \times 16^{+0})$

$= (A \times 256) + (8 \times 16) + (5 \times 1)$

$= (10 \times 256) + (8 \times 16) + (5 \times 1)$

$= 2560 + 128 + 5$

$= 2693$

$\therefore$ $A85_{16} = 2693_{16}$ **Ans.**

(*c*) Given : The given hexadecimal number

Step 1. E 5

$\downarrow$ $\downarrow$

Step 2. 16^{+1} 16^{+0}

Step 3. $(E\times 16^{+1})+(5\times 16^{+0})$

$=(E\times 16)+(5\times 1)$

$=(14\times 16)+(5)$

$=224+5$

$=229$

$\therefore$ $E5_{16}=2693_{16}$ **Ans.**

2-25. Decimal-to-Hexadecimal Conversion

It is a systematic way of converting numbers from decimal to hexadecimal. The procedure for conversion is as given below:

Step 1. Begin by dividing the given decimal number by 16.

Step 2. Divide each resulting quotient by 16 until there is a zero whole number quotient.

Step 3. The remainders generated by each division form the hexadecimal number. The first remainder to be produced is the least-significant digit (LSD) in the hexadecimal number and the last remainder to be produced is the most-significant digit (MSD). In other words, reading the remainders from bottom-to-top constitutes the hexadecimal number.

Note that any remainders that are greater than 9 are represented by the letters A through F.

If a calculator is being used to perform the divisions in the conversion process, the results will include a decimal fraction of a remainder. The remainder can be obtained by multiplying the fraction by 16.

Let us illustrate the procedure for conversion of the decimal number 650 to its equivalent hexadecimal number.

$$650\div 16=40.625$$

So in this case 40 is a quotient and 0.625 is a fraction remainder. The remainder becomes

$$0.625\times 16=10=\text{A (hexadecimal number)}$$

Thus the complete conversion is as shown below:

							Top	
$650\div 16$	$=40.625$	$=40$	with a remainder	0.625	$\rightarrow$	$0.625\times 16=10\,(=A)$	$\uparrow$	LSD
$40\div 16$	$=2.5$	$=2$	with a remainder	0.5	$\rightarrow$	$0.5\times 16=8$		
$2\div 16$	$=0.125$	$=0$	with a remainder	0.125	$\rightarrow$	$0.125\times 16=2$ MSD		
							Bottom	

Since the quotient is zero, therefore we will stop the division by 16. Reading the remainders from bottom to top, we find that the hexadecimal equivalent of the decimal number 650 is 28A.

$\therefore$ $650_{10}=28A_{16}$ **Ans.**

Example 2-31. *Convert the number* 151_{10} *to its equivalent hexadecimal number.*

Solution. Given : The number $=151_{10}$

We know that we can use the repeated division by 16 for converting a decimal number to its haxadecimal equivalent. Using division by hand, we get,

			Top
151 ÷ 16 = 9	with a remainder	7	↑ LSD
9 ÷ 16 = 0	with a remainder	9	MSD
			Bottom

Since the quotient is zero, so we will stop the further division. Reading the remainders from bottom to top, the hexadecimal number is 97.

∴ $151_{10} = 97_{16}$ **Ans.**

Example 2-32. *Convert the number* 498_{10} *to its equivalent hexadecimal number.*

Solution. Given : The number = 498_{10}

We know that we can use the repeated division by 16 for converting a decimal number to its hexadecimal equivalent . Using division by hand, we get,

			Top
498 ÷ 16 = 31	with a remainder	2	↑ LSD
31 ÷ 16 = 1	with a remainder	15 (= F)	
1 ÷ 16 = 0	with a remainder of	1	MSD
			Bottom

Since the quotient is zero, so we will stop the further division. Reading the remainders from bottom to top, the hexadecimal number is 1F2.

∴ $498_{10} = 1F2_{16}$ **Ans.**

Example 2-33. *Convert the following decimal numbers to their hexadecimal equivalents*:

(*a*) 2890 (*b*) 4019

Solution.

(*a*) Given : The decimal number = 2890

				Top
2890 ÷ 16 = 180.625 = 180	with a remainder	0.625 = 0. 625 × 16 = 10 (=A)		LSD
180 ÷ 16 = 11.25 = 11	with a remainder	0.25 = 0.25 × 16 = 4		↑
11 ÷ 16 = 0.6875 = 0	with a remainder	0.6875 = 0.6875 × 16 = 11 (=B)	MSD	
				Bottom

Since the quotient is zero, so we will stop the further division. Reading from bottom to top, the equivalent hexadecimal number of a decimal number 2890 is B4 A.

∴ $2890_{10} = B4A_{16}$ **Ans.**

(*b*) Given : The decimal number = 4019

			Top
4019 ÷ 16 = 251.875 = 251	with a remainder	0.875 = 0.875 × 16 = 14 (= E)	↑ LSD
251 ÷ 16 = 15.688 = 15	with a remainder	0.688 = 0.688 × 16 = 11 (= B)	
15 ÷ 16 = 0.938 = 0	with a remainder	0.738 = 0.738 × 16 = 15 (= F)	MSD
			Bottom

Since the quotient is zero, so we will stop the further division. Reading from bottom to top, the equivalent hexadecimal under of a decimal number 4019 is FBE.

∴ $4019_{10} = FBE_{16}$ **Ans.**

Example 2-34. *Convert the following decimal number to their hexadecimal equivalent.*

(*a*) 14_{10} (*b*) 80_{10} (*c*) 3000_{10}

(*Anna University, May/June 2007*)

Solution.

(*a*) Given*:* The decimal number = 14_{10}

$(14)_{10} = (E)_{16}$ **Ans.**

(*b*) Given: The decimal number = 80_{15}

$80 \div 16 = 5$ with a remainder 0

$80_{10} = 50_{16}$ **Ans.**

(*c*) Given: The decimal number = 3000_{10}

Top

$3000 \div 16 = 187.5 = 187$	with a remainder $0.5 = 0.5 \times 16 = 8$	↑ LSD
$187 \div 16 = 11.6875 = 11$	with a remainder $0.6875 = 0.6875 \times 16 = 11 = B$	│
$11 \div 16 = 0.6875 = 0$	with a remainder $0.6875 = 0.6875 \times 16 = 11 = B$	│ MSD

Bottom

Since the quotient is zero, so we will stop the further division. Reading from bottom to top, the equivalent hexadecimal number of a decimal number 3000 is BB8.

2-26. Hexadecimal-to-Binary Conversion

Like the octal number system, the hexadecimal number system is used primarily as a "shorthand" method for representing binary numbers. For conversion, each hexadecimal digit is covered to its four-bit binary equivalent (refer to Table 2-5 page 43)

The procedure for converting hexadecimal number to its binary equivalent is as follows:

Step 1. Write the hexadecimal number

Step 2. Write the 4-bit binary equivalent for each hexadecimal digit

Step 3. Check if there are any zeros on the left most position of the binary number obtained. Drop off the zeros and write the answer as a binary number.

In order to illustrate the above procedure, Consider the conversion of $2D6_{16}$ to its equivalent binary.

Step 1. 2 D 6

↓ ↓ ↓

Step 2. $\overbrace{0010}\ \overbrace{1101}\ \overbrace{0110}$

Step 3. $\underbrace{00}1011010110$

↓

Drop off the leading zeros

Therefore the required binary number is, 1011010110

$\therefore\ 2D6_{16} = 1011010110_2$ **Ans.**

Example 2-35. *Convert the following hexadecimal numbers of their binary equivalent*:

(*a*) 9 F 2 (*b*) 2 A 6

Solution.

(*a*) Given the hexadecimal number = 9 F 2

Step 1. 9 F 2

↓ ↓ ↓

Step 2. 1001 1111 0010

Step 3. 100111110010

$\therefore$ 9F 2_{16} = 1001 1111 0010_2 **Ans.**

(*b*) Given : The hexadecimal number = 2 A 6.

Step 1. 2 A 6

↓ ↓ ↓

Step 2. 0010 1010 0110

Step 3. 001010100110

↓

Drop off the leading zeros

$\therefore$ 2 A 6_{16} = 1010100110_2 **Ans.**

2-27. Binary-to-Hexadecimal Conversion

Conversion from binary to-hexadecimal is just the reverse of the process discussed in the last article (i.e. reverse of the process for hexadecimal to-binary conversion.

The procedure for converting hexadecimal number to its binary equivalent is as follows:

Step 1. Write the binary number.

Step 2. Starting from the right most position, group the binary number into groups of four-bits. If necessary, we can add zeros at the left-most position.

Step 3. Convert each 4-bit binary group to its equivalent hexadecimal digit.

In order to illustrate the procedure, consider the conversion of binary 10111 to its equivalent hexadecimal.

right-most position

Step 1. 1 0 1 1 1

Step 2. 0001 0111

Three zeros are added to form a Four-bit Group

↓ ↓

Step 3 1 7

$\therefore$ $10111_2 = 17_{16}$ **Ans.**

Example 2.36. *Convert the following binary numbers to their hexadecimal equivalents.*

(*a*) 10100110 (*b*) 1111 110000 (*c*) 100110000010.

Solution.

(*a*) Given : The binary number = 10100110.

Step 1. 1 0 1 0 0 1 1 0

Step 2. $\underbrace{1010}$ $\underbrace{0110}$

↓ ↓

Step 3. A 6

$\therefore$ $10100110_2 = A6_{16}$ **Ans.**

(*b*) Given : The binary number = 1111110000

Step 1. 1 1 1 1 1 1 0 0 0 0

Step 2. $\underbrace{\overbrace{00}11}$ $\underbrace{1111}$ $\underbrace{0000}$

Two zeros added to form 4-bit group

↓ ↓ ↓

Step 3. 3 F 0

$\therefore$ $1111110000_2 = 3F0_{16}$ **Ans.**

Example 2-37. *Convert* $(10011.101)_2$ *into hexadecimal number.*

(*PTU., May 2009*)

Solution. Given: The binary number = 10011.1101

Step 1. 1 0 0 1 1 . 1 1 0 1

Step 2. $\underbrace{0\ 0\ 0\ 1}_{\downarrow}$ $\underbrace{0\ 0\ 1\ 1}_{\downarrow}$ $\underbrace{1\ 1\ 0\ 1}_{\downarrow}$

Step 3. 1 3 D

$\therefore$ $10011.1101_2 = 13.D_{16}$ **Ans.**

Example 2-38. *Convert the following number into hexadecimal and decimal.* $(1110.101)_2$

(*Gauhati University., 2003*)

Solution. Given: The binary number $(1110.101)_2$

Hexadecimal conversion:

Step 1. $1\ 1\ 1\ 0 \bullet\ 1\ 0\ 1$

Step 2. $\underbrace{1\ 1\ 1\ 0}_{\downarrow} \bullet \underbrace{1\ 0\ 1\ 0}_{\downarrow}$ One zero adde to form 4-bit group

Step 3. $\quad\ \mathrm{E} \qquad\qquad \mathrm{A}$

$\therefore$ $1110.101_2 = \mathrm{EA}_{16}$ **Ans.**

Decimal Conversion:

$$\begin{array}{llllllll} \text{Step 1.} & 1 & 1 & 1 & 0\ . & 1 & 0 & 1 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\ \text{Step 3.} & 2^3 & 2^2 & 2^1 & \cancel{2^0} & 2^{-1} & \cancel{2^{-2}} & 2^{-3} \end{array}$$

Step 4. $2^3 + 2^2 + 2^1 + 2^{-1} + 2^{-3}$

$= 8 + 4 + 2 + 0.5 + 0.125$

$= 14.625$ **Ans.**

Example 2-39. *Covert the following into Binary and Hexadecimal*

(*a*) 243

(*b*) 534 (*Mahatma Gandhi; University, May 2003*)

Solution.

(*a*) Given the decimal number = 243

Decimal-to-Binary

We know that the conversion from decimal-to-binary can be done using repeated division by 2 method.

			Top
243 ÷ 2 = 121	with a remainder	1	(LSB)
121 ÷ 2 = 60	with a remainder	1	
60 ÷ 2 = 30	with a remainder	0	
30 ÷ 2 = 15	with a remainder	0	
15 ÷ 2 = 7	with a remainder	1	
7 ÷ 2 = 3	with a remainder	1	
3 ÷ 2 = 1	with a remainder	1	
1 ÷ 2 = 0	with a remainder	1	↓ (MSB)
			Bottom

Reading the remainder from bottom to top the binary number for the decimal number 243 is 11110011

$\therefore$ $243_{10} = 11110011_2$ **Ans.**

Hexadecimal

We know that binary-to-hexadecimal can be done as

Step 1. 1 1 1 1 0 0 1 1

Step 2. $\underset{\downarrow}{\underbrace{1\,1\,1\,1}}\quad \underset{\downarrow}{\underbrace{0\,0\,1\,1}}$

∴ $243_{10} = F3_{16}$ **Ans.**

(b) *Given:* the decimal number 534

Decimal-to-Binary

We know that the conversion from decimal to binary can be done by using repeated-division-by 2 method

				Top
534 ÷ 2	= 267	with a remainder	0	LSB
267 ÷ 2	= 133	with a remainder	1	
133 ÷ 2	= 66	with a remainder	1	
66 ÷ 2	= 33	with a remainder	0	
33 ÷ 2	= 16	with a remainder	1	
16 ÷ 2	= 8	with a remainder	0	
8 ÷ 2	= 4	with a remainder	0	
4 ÷ 2	= 2	with a remainder	0	
2 ÷ 2	= 1	with a remainder	0	
1 ÷ 2	= 0	with a remainder	1	MSB
				Bottom

Reading the remainder from bottom to top, the binary number for the decimal number 534 is 1000010110

∴ $534_{10} = 1000010110$ **Ans.**

Hexadecimal

Step 1.) 0010 110

Two Zeros added →

Step 2. $\underset{\downarrow}{\underbrace{\overbrace{0\,0}\,1\,0}}\quad \underset{\downarrow}{\underbrace{0\,0\,0\,1}}\quad \underset{\downarrow}{\underbrace{0\,1\,1\,0}}$

Step 3.

∴ $534_{10} = 216_{16}$ **Ans.**

2-28. Hexadecimal-to-Octal Conversion

There are two methods for converting hexadecimal number to its octal equivalent. The first method is to convert the given hexadecimal number to binary equivalent and then from binary to octal. The second method is to convert the given hexadecimal number to its decimal equivalent and then from decimal to octal. However the first method is much simpler and convenient.

The procedure for conversion of hexadecimal-to-octal is as given below.

Step 1. Write the hexadecimal number

Step 2. Replace each hexadecimal digit by its 4-bit binary equivalent. This will give us the binary equivalent of the given hexadecimal number.

Step 3. Form 3-bit combinations by starting from the right-most position. Replace each 3-bit combination by its octal equivalent. This will give us the octal equivalent for the given hexadecimal number.

In order to illustrate the procedure, consider the conversion of 5C2 to its octal equivalent.

$$\begin{array}{llll} \text{Step 1.} & 5 & C & 2 \\ & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & \overbrace{0101} & \overbrace{1100} & \overbrace{0010} \end{array}$$

$$\begin{array}{lcccc} \text{Step 3.} & \underbrace{010} & \underbrace{111} & \underbrace{000} & \underbrace{010} \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ & 2 & 7 & 0 & 2 \end{array}$$

Thus the octal equivalent for the given hexadecimal number is 2702_8

$\therefore \quad 5C2_{16} = 2702_8$

Example 2-40. *Convert the hexadecimal number* $8AD9_{16}$ *to its octal equivalent.*

Solution. Given : The hexadecimal number = $8AD9_{16}$

$$\begin{array}{lcccc} \text{Step 1.} & 8 & A & D & 9 \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 1000 & 1010 & 1101 & 1001 \end{array}$$

$$\begin{array}{lccccc} \text{Step 3.} & \underbrace{100} & \underbrace{010} & \underbrace{011} & \underbrace{011} & \underbrace{001} \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & 4 & 2 & 3 & 3 & 1 \end{array}$$

This the octal equivalent for the given hexadecimal is 42331_8.

$\therefore \quad 8AD9_{16} = 42331_8$ **Ans.**

Example 2-41. *Convert* $(A7.3B)_{16}$ *into its octal equivalent.*

(Nagpur University, Oct. / Nov. 1997*)*

Solution. Given : The hexadecimal number = A7.3B

First of all we will convert $A7.3B_{16}$ to its equivalent binary and then to its octal equivalent.

$$\begin{array}{ccc}
\text{A}\quad 7 & 3\quad \text{B} & \text{— Hexadecimal number}\\
\downarrow\quad\downarrow & \downarrow\quad\downarrow & \\
\overbrace{1010}\overbrace{0111} & \overbrace{0011}\overbrace{1011} & \text{— Binary equivalent}\\
\underbrace{10}\,\underbrace{100}\,\underbrace{111} & \underbrace{00}\,\underbrace{111}\,\underbrace{011} & \\
\downarrow\quad\downarrow\quad\downarrow & \downarrow\quad\downarrow\quad\downarrow & \\
2\quad 4\quad 7 & 1\quad 6\quad 6 & \text{— Octal equivalent}
\end{array}$$

Thus we find that,

$\therefore$ $A7.3B_{16} = 247.166_8$ **Ans.**

2-29. Octal-to-Hexadecimal Conversion

Like hexadecimal-to-octal there are two methods for converting octal-to-hexadecimal. The first method is to convert the given octal number to its binary equivalent and then from binary to hexadecimal. The second method is to convert octal-to-hexadecimal first and then from decimal-to-hexadecimal. However the first method is much simpler and convenient.

The procedure for conversion from octal-to-hexadecimal is as given below:

Step 1. Write the octal number.

Step 2. Replace each octal digit by its 3-bit binary equivalent. This will give us the binary equivalent of the given octal number.

Step 3. Form the 4-bit combinations by starting from the right-most position. Replace each 4-bit combination by its hexadecimal equivalent. This will give us the hexadecimal equivalent for the given octal number.

In order to illustrate the procedure, consider the conversion of 321_8 to its hexadecimal equivalent.

$$\begin{array}{llll}
\text{Step 1.} & 3 & 2 & 1\\
 & \downarrow & \downarrow & \downarrow\\
\text{Step 2.} & \overbrace{011} & \overbrace{010} & \overbrace{001}
\end{array}$$

$$\begin{array}{ll}
\text{Step 3.} & 0\underbrace{1101}\;\underbrace{0001}\\
 & \quad\downarrow\qquad\downarrow\\
 & \quad\text{D}\qquad 1
\end{array}$$

Thus the hexadecimal equivalent for the given octal number is $D1_{16}$

$\therefore$ $321_8 = D1_{16}$ **Ans.**

Example 2-42. *Convert the octal number* 1024_8 *to its hexadecimal equivalent.*

Solution. Given : The octal number = 1024.

$$\begin{array}{lllll}
\text{Step 1.} & 1 & 0 & 2 & 4\\
 & \downarrow & \downarrow & \downarrow & \downarrow\\
\text{Step 2.} & \overbrace{001} & \overbrace{000} & \overbrace{010} & \overbrace{100}
\end{array}$$

$$\begin{array}{ll}
\text{Step 3.} & \underbrace{0\;0\;1\;0}\;\underbrace{0\;0\;0\;1}\;\underbrace{0\;1\;0\;0}\\
 & \quad\downarrow\qquad\quad\downarrow\qquad\quad\downarrow\\
 & \quad 2\qquad\quad 1\qquad\quad 4
\end{array}$$

Thus the hexadecimal equivalent for the given octal number is 214_{16}.

$\therefore$ $1024_8 = 214_{16}$ **Ans.**

Example 2-43. *Convert the following number to the given base*:

(*i*) $(62)_{10} = (?)_2 = (?)_8$

(*ii*) $(AFB)_{16} = (?)_2 = (?)_8$

(*Nagpur University, Mar./Apr.* 1998)

Solution.

(*i*) $(62)_{10} = (?)_2$

Top

$62 \div 2 = 31$	with a remainder	0	LSB
$31 \div 2 = 15$	with a remainder	1	↑
$15 \div 2 = 7$	with a remainder	1	↑
$7 \div 2 = 3$	with a remainder	1	↑
$3 \div 2 = 1$	with a remainder	1	↑
$1 \div 2 = 0$	with a remainder	1	MSB

Bottom

$\therefore$ $(62)_{10} = 111110_2$ **Ans.**

In order to convert the binary 111110 to its equivalent octal, start from the right-most position and form a groups of 3-bits Thus,

$$\underbrace{1\,1\,1}_{\substack{\downarrow\\ 7}}\ \underbrace{1\,1\,0}_{\substack{\downarrow\\ 6}}$$

$\therefore$ $(62)_{10} = (111110)_2 = (76)_8$ **Ans.**

(*ii*) $(AFB)_{16} = (?)_2 = (?)_8$

In order to convert the given hexadecimal number AFB to its equivalent binary, replace each hexadecimal digit by its 4-bit binary equivalent. Thus

$$\overbrace{1\,0\,1\,0}^{\substack{A\\ \downarrow}}\ \overbrace{1\,1\,1\,1}^{\substack{F\\ \downarrow}}\ \overbrace{1\,0\,1\,1}^{\substack{B\\ \downarrow}}$$

$\therefore$ $(AFB)_{16} = 101011111011_2$ **Ans.**

In order to convert the binary number 101011111011 to its octal equivalent, start from the right-most position and form groups of 3-bits and replace each 3-bit group by its equivalent octal digit. Thus,

$$\underbrace{1\,0\,1}_{\substack{\downarrow\\ 5}}\ \underbrace{0\,1\,1}_{\substack{\downarrow\\ 3}}\ \underbrace{1\,1\,1}_{\substack{\downarrow\\ 7}}\ \underbrace{0\,1\,1}_{\substack{\downarrow\\ 3}}$$

$\therefore$ $(AFB)_{16} = 101011111011_2 = (5373)_8$ **Ans.**

Example 2-44. Perform the following

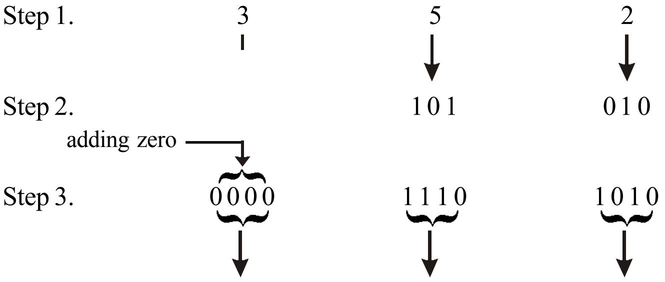
(*i*) $(352)_8 = (?)_{16}$

(*ii*) $(45.367)_{10} = (?)_2$

(Nagpur University., 2008)

Solution

(*i*) Given: $(352)_8 = (\ )_{16}$



$\therefore \quad (352)_8 = (EA)_{16}$

(*ii*) Given: $(45.367)_{10} = (?)_2$

We know that the integral that of decimal number can be expressed by the sum-of-weight as follow.

$$45 = 32 + 8 + 4 + 1$$
$$2^5 + 2^3 + 2^2 + 2^0$$

Placing 1s in the appropriate weight positions 2^5, 2^3, 2^2 and 2^0 0_5 in the 2^4 and 2^1 positions determine the binary number for decimal 45

2^5	2^4	2^3	2^2	2^1	2^0
↓	↓	↓	↓	↓	↓
1	0	1	1	0	1

$\therefore \quad 45_{10} = 101101_2$

The fractional part of decimal number is converted in binary form by multiplication by 2 method.

			Top	
$0.367 \times 2 = 0.734$	with a carry	0	↓	LSB
$0.734 \times 2 = 1.468$	with a carry	1		
$0.468 \times 2 = 0.936$	with a carry	0		
$0.936 \times 2 = 1.872$	with a carry	1		
$0.872 \times 2 = 1.744$	with a carry	1		
$0.744 \times 2 = 1.488$	with a carry	1		
$0.488 \times 2 = 0.976$	with a carry	0	↓	MSB
			Bottom	

And so on ....

$$0.367_{10} = 0.010111_2$$

Thus

$(45.367)_{10} = (101101.010111)_2$ **Ans.**

Example 2-45. *Convert the following*

(*i*) $(129.56)_{10} = (?)_2$

(*ii*) $(ABC)_{16} = (?)_8$

(*iii*) $(11001010.1001) = (?)_{16} = (?)_8$

(*Nagpur University., 2008*)

Solution.

(*i*) Given: $(129.56)_{10} = (?)_2$

We know that integral part can be expressed by the sum-of-weight as follows

$$129 = 128 + 1 = 2^7 + 2^0$$

Placing 1s in the appropriate weight positions 2^7 and 2^0 and 0_5 in the 2^6, 2^5, 2^4 2^3, 2^2, and 2^1 position determines the binary number for 129 thus,

$$\begin{array}{cccccccc} 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$129_{10} = 10000001_2$

The fraction part of decimal number is converted to binary number by repeated multiplication by 2 method.

$$\begin{array}{rclcl} & & & & \text{Top} \\ 0.56\times2 & = & 1.12 & \text{with a carry of} & 1 \quad \uparrow \text{ LSB} \\ 0.12\times2 & = & 0.24 & \text{with a carry of} & 0 \quad \big| \\ 0.24\times2 & = & 0.48 & \text{with a carry of} & 0 \quad \big| \\ 0.48\times2 & = & 0.96 & \text{with a carry of} & 0 \quad \big| \\ 0.96\times2 & = & 1.92 & \text{with a carry of} & 1 \quad \big| \text{ MSB} \\ & & & & \text{Bottom} \end{array}$$

And so on

$$0.56_{10} = 10001_2$$

$(129.96)_{10} = (10000001.1001)_2$ **Ans.**

(*ii*) Given: $(ABC)_{16} = (?)_8$

$$\begin{array}{lcccc} \text{Step 1.} & A & B & C & \\ & \downarrow & \downarrow & \downarrow & \\ \text{Step 2.} & 1010 & 1011 & 1100 & \\ & & & & \\ \text{Step 3.} & \underbrace{101}_{\downarrow} & \underbrace{010}_{\downarrow} & \underbrace{111}_{\downarrow} & \underbrace{100}_{\downarrow} \end{array}$$

This the octal equi for the gi :xadecim 274$_8$

$(ABC)_{16} = (5274)_8$ **Ans.**

(*iii*) Given: $(11001010\,.\,1001)_2 = (?)_{16} = (?)_8$

Binary to hexadecimal:

Step 1. 1 1 0 0 1 0 1 0. 1 0 0 1

Step 2. $\underbrace{1\,1\,0\,0}_{\downarrow}$ $\underbrace{1\,0\,1\,0}_{\downarrow}$ $\underbrace{1\,0\,0\,1}_{\downarrow}$

Step 3.

$\therefore$ (11001010.100 CA.9)$_{16}$ A

Binary to octal

Step 1. 1 1 0 0 1 0 1 0. 1 0 0 1

adding a leading zero $\downarrow$ adding $\underbrace{\text{zero}}$

Step 2. $\underbrace{0\,1\,1}_{\downarrow}$ $\underbrace{0\,0\,1}_{\downarrow}$ $\underbrace{0\,1\,0}_{\downarrow}$. $\underbrace{1\,0\,0}_{\downarrow}$

$\therefore$ (11001010 . 10(312.44)$_8$

Example 2-46. *Convert the following a indicated by their base:*

(*i*) $(650)_{10} \rightarrow (\quad)_{16}$

(*ii*) $(\text{CA57})_{16} \rightarrow (\quad)_2$

(*iii*) $(\text{7BF})_{16} \rightarrow (\quad)_2$

(*iv*) $(110101)_2 \rightarrow (\quad)_8$

(*v*) $(\text{E7F6})_{16} \rightarrow (\quad)_{10}$ *(Jamia University.,2007)*

Solution.

(*i*) Given: $(650)_{10} \rightarrow ()_{16}$

We know that we can use the repeated division 16 for converting a decimal number to its hexadecimal equivalent. Using division by hand. we get

				Top
$650 \div 16 = 40.625$	$= 40$	with a remainder	$0.625 = 0.625 \times 16 = 10 = \text{A}$	↑ LSD
$40 \div 16 = 2.5$	$= 2$	with a remainder	$0.5 = 0.5 \times 16 = 8$	
$2 \div 16 = 0.125$	$= 0$	with a remainder	$0.125 \times 16 = 2$	MSD
				Bottom

Since the quotient i_1 zero, so we will stop the further division. Reading from bottom to top, the equivalent hexadecimal number of a decimal number 650 is 28A

$\therefore$ $650_{10} = 28\text{A}_{16}$ **Ans.**

(*ii*) $(\mathrm{CA57})_{16} \rightarrow (\ )_2$

The given hexadecimal number = CA57

$$\begin{array}{lcccc}
\text{Step 1.} & \mathrm{C} & \mathrm{A} & 5 & 7 \\
 & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & \overbrace{1100} & \overbrace{1010} & \overbrace{0101} & \overbrace{0111} \\
 & & & & \\
\text{Step 3.} & 1100 & 1010 & 0101 & 0111
\end{array}$$

$\therefore$ $\mathrm{CA57}_{16} = 1100101001010111_2$ **Ans.**

(*iii*) $(7\mathrm{BF})_{16} \rightarrow (\ )_2$

The given hexadecimal number $7\mathrm{BF}_{16}$

$$\begin{array}{lccc}
\text{Step 1.} & 7 & \mathrm{B} & \mathrm{F} \\
 & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & \overbrace{0111} & \overbrace{1011} & \overbrace{1111} \\
 & & & \\
\text{Step 3.} & \underset{\downarrow}{\underbrace{0}}111 & 1011 & 1111
\end{array}$$

Drop off the leadin ero

$\therefore$ $7\mathrm{BF}_{16} = 11110111111_2$ **Ans.**

(*iv*) $(110101)_2 \rightarrow (\ )_{58}$

The given binary number 110101_2

We know that in order to convert the given binary number to its equivalent octal, we have to form-3-bit combinations while moving right to left by starting from the right most position.

$$\begin{array}{cc}
\underbrace{110} & \underbrace{101} \\
\downarrow & \downarrow
\end{array}$$

$\therefore$ $110101_2 = 65_8$ A

(*v*) $(\mathrm{E7F6})_{16} \rightarrow (\ )_{10}$

We know that to convert E7F6 to it equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then table the sum of these products.

$$\begin{array}{lcccc}
\text{Step 1.} & \mathrm{E} & 7 & \mathrm{F} & 6 \\
 & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & 16^{+3} & 16^{+2} & 16^{+1} & 16^{+0}
\end{array}$$

Step3. $(\mathrm{E} \times 16^{+3}) + (7 \times 16^{+2}) + (\mathrm{F} \times 16^{+1}) + (6 \times 16^{+0})$

$= (14 \times 4096) + (7 \times 256) + (15 \times 16) + (6 \times 1)$

$= 57344 + 1792 + 240 + 16$

$= 59382$

$\therefore$ $\mathrm{E7F6}_{16} = 59382_{10}$ **Ans.**

Example 2-47. *Convert decimal 225.225 to binary, octal and hexadecimal bases.*

(*PTU, May 2008, 2009, GTU., Dec., 2011, May 2011*)

Solution. Given: The decimal number = 225.225

Decimal to Binary

Conversion of integer part we know that the conversion of integer part is done by using repeated division by 2 method.

				Top ↑
$225 \div 2$	$= 112$	with a remainder	1	LSB
$112 \div 2$	$= 56$	with a remainder	0	
$56 \div 2$	$= 28$	with a remainder	0	
$28 \div 2$	$= 14$	with a remainder	0	
$14 \div 2$	$= 7$	with a remainder	0	
$7 \div 2$	$= 3$	with a remainder	1	
$3 \div 2$	$= 1$	with a remainder	1	
$1 \div 2$	$= 0$	with a remainder	1	MSB
				Bottom

$\therefore \quad 225_{10} = 11100001_2$

Conversion of fractional part

				Top
0.225×2	$= 0.45 = 0.45$	with a carry of	0	LSM
0.45×2	$= 0.9 = 0.9$	with a carry of	0	
0.9×2	$= 1.8 = 0.8$	with a carry of	1	
0.8×2	$= 1.6 = 0.6$	with a carry of	1	
0.6×2	$= 1.2 = 0.2$	with a carry of	1	↓ MSB
				Bottom

and so on

we can continue the process further if we wish because the binary fractional part has not reduced to zero. However we stop here because the desire number of binary places is reached.

$\therefore \quad 0.225_{10} = 0.00111_2$

Combining the integral and fractional part of the number, we get,

$225.225_{10} = 11100001.00111_2$

Decimal-to-Octal

The decimal number 225 . 225.

We know that we can convert the integer part of the given number to its equivalent octal by the repeated division by 8 method. Whereas the fractional part is to be converted to its equivalent octal by the repeated multiplication by 8 method.

Conversion of integer part

		Top ↑
$225 \div 8 = 28.125$	= with a remainder $0.125 \rightarrow 0.125 \times 8 = 1$	LSD
$28 \div 8 = 3.5$	$= 3$ with a remainder $0.5 \rightarrow 0.5 \times 8 = 4$	
$3 \div 8 = 0.375$	$= 0$ with a remainder $0.375 \rightarrow 0.375 \times 8 = 3$	MSD
		Bottom

∴ $225_{10} = 341_8$

Conversion of fractional part

		Top	
$0.225 \times 8 = 1.8 = 1.8$	with a carry of 1	↓	MSD
$0.8 \times 8 = 6.4 = 0.4$	with a carry of 6		
$0.4 \times 8 = 3.2 = 0.2$	with a carry of 3		
$0.2 \times 8 = 1.6 = 0.6$	with a carry of 1		
$0.6 \times 8 = 6.4 = 0.4$	with a carry of 6	↓	LSD
		Bottom	

Combing the integer and the fractional part of the numbers.

We get,

$225.225_{10} = (341.16316\ldots..)_8$ **Ans.**

Decimal-to-Hexadecimal

The decimal number 225.225

Conversion of integer part.	Top	
$225 \div 16 = 14.0625 = 14$ with a remainder $0.0625 \rightarrow 0.625 \times 16 = 1$	LSD ↑	
$4 \div 16 = 0.875 = 0$ with a remainder $0.875 \rightarrow 0.875 \times 16 = 14 = \text{E}$		MSD
∴ $225_{10} = \text{E1}_{16}$	Bottom	

Conversion of fractional part

			Top
$0.225 \times 16 = 3.6 = 0.6$	with a carry of	3	MSD ↓
$0.6 \times 16 = 9.6 = 0.6$	with a carry of	9	LSD ↓
			Bottom

Since the fractional part repeating, so we will stop the further multiplication. Reading the carry from top to bottom, the required hexadecimal number is 39. Combining the integral and fractional part of the number, we get,

$225.225_{10} = \text{E1.39}_{16}$ **Ans.**

Example. 2-48. *Do the following conversions*

(*i*) *$(10110011)_2$ to Hexadecimal conversions*

(*ii*) *$(10110011)_2$ to decimal number conversions*

(*iii*) *$(0.6875)_{10}$ into binary* (*Assam Eniginessing College, 2006*)

Solution.

(*i*) Given: the binary number $(10110011)_2$

Step 1. 1 0 1 1 0 0 1 1

Step 2. $\underbrace{1\,0\,1\,1}$ $\underbrace{0\,0\,1\,1}$

∴ $10110011_2 =$ B s.

(*ii*) Given: the binary number $(10110011)_2$

$$\begin{array}{lcccccccc}
\text{Step 1.} & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 & & & & & & & & \\
\text{Step 3.} & 2^7 & \cancel{2^6} & 2^5 & 2^4 & \cancel{2^3} & \cancel{2^2} & 2^1 & 2^0
\end{array}$$

Step 4. $2^7 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = 179$ **Ans.**

(*iii*) Given: The decimal number $(0.6875)_{10}$

			Top	
$0.6875 \times 2 = 1.375 = 0.375$	with a carry of	1	↓	LSB
$0.375 \times 2 = 0.75 = 0.75$	with a carry of	0		
$0.75 \times 2 = 1.5 = 0.5$	with a carry of	1		
$0.5 \times 2 = 1.0 = .0$	with a carry of	1		LSB
			Bottom	

$\therefore$ $0.6875_{10} = 0.1011_2$ **Ans.**

Example 2-49. *Convert the following numbers :*

(*a*) $(1276)_{10} = (\,)_8$

(*b*) $(36.125)_8 = (\,)_{10}$

(*c*) $(327)_8 = (\,)_2$

(*d*) $(\text{FBA})_{16} = (\,)_2$

(*e*) $(17173)_8 = (\,)_{16}$

(*f*) $(374.37)_{10} = (\,)_{16}$

(*g*) $(3\text{AB})_{16} = (\,)_{10}$

(*RGTU., June 2011*)

Solution.

(*a*) Given: The number $= (1276)_{10} = (\,)_8$

We know that the integer part of the given number to its equivalent binary by sum-of-weights method.

					Top
$1276 \div 8 = 159.5 = 120$	with a carry	0.5	$0.5 \times 8 = 4$	↑	(LSD)
$159 \div 8 = 19.875 = 19$	with a carry	0.875	$0.875 \times 8 = 7$		
$19 \div 8 = 2.375 = 2$	with a carry	0.375	$0.375 \times 8 = 3$		
$2 \div 8 = 0.25 = 0$	with a carry	0.25	$0.25 \times 8 = 2$		(MSD)
					Bottom

Since the quotient is zero, therefore we will stop the division-by-8. Reading the remainders from bottom to top, we find that the octal equivalent of the decimal number 1276 is 2374.

$\therefore$ $1276_{10} = 2374_8$ **Ans.**

(*b*) Given: The number = $(36.125)_8 = (\ )_{10}$

We know that in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{llllll} \text{Step 1.} & 3 & 6\ \ . & 1 & 2 & 5 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 8^{1} & 8^{0} & 8^{-1} & 8^{-2} & 8^{-3} \end{array}$$

Step 3.

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

$$= (3 \times 8) + (6 \times 1) + (1 \times 0.125) + (2 \times 0.015625) + (5 \times 0.00195)$$

$$= 24 + 6 + 0.125 + 0.03125 + 0.00975$$

$$= 30.166_{10}\ \textbf{Ans.}$$

(*c*) Given: The number = $(327)_8 = (\ )_2$

We know that in order to convert an octal number to its binary equivalent, we have to convert each octal digit to its 3-bit binary equivalent. Thus

$$\begin{array}{ccc} 3 & 2 & 7 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{011} & \overbrace{010} & \overbrace{111} \end{array}$$

Thus the octal 327_8 is equivalent to the binary 011010111_2

$\therefore$ $327_8 = 011010111_2$ **Ans.**

(*d*) Given: The number = $(\text{FBA})_{16} = (\ )_2$

$$\begin{array}{llll} \text{Step 1.} & \text{F} & \text{B} & \text{A} \\ & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & \overbrace{1111} & \overbrace{1011} & \overbrace{1010} \end{array}$$

Step 3. 111110111010

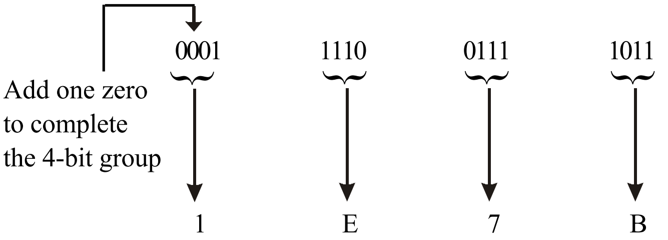
$\therefore$ $FBA_{16} = 111110111010_2$ **Ans.**

(*e*) Given: The number = $(17173)_8 = (\ )_{16}$

First of all we will convert 17173_8 to its equivalent binary number and then to its hexadecimal equivalent.

$$\begin{array}{ccccc} 1 & 7 & 1 & 7 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \overbrace{001} & \overbrace{111} & \overbrace{001} & \overbrace{111} & \overbrace{011} \end{array}$$

then convert to hexadecimal equivalent,



Thus we find that,

$\therefore$ $17173_8 = 1E7B_8$ **Ans.**

(*f*) Given: The number $= (374.37)_{10} = (\,)_{16}$

We know that we can convert the integer part of the given number to its equivalent hexadecimal by the repeated division by 16 method. Whereas the fractional part is to be converted to its equivalent hexadecimal by the repeated multiplication by 16 method.

Conversion of integer part

We know that we can use the repeated division by 16 for converting a integral decimal number to its hexadecimal equivalent and the fractional part is convert by,

Conversion of integer part

			Top ↑	
$374 \div 16 = 23$	with a remainder	$0.375 \rightarrow 0.375 \times 16 = 6$		LSD
$23 \div 16 = 1$	with a remainder	$0.4375 \rightarrow 0.4375 \times 16 = 7$		
$1 \div 16 = 0$	with a remainder	$0.0625 \rightarrow 0.0625 \times 16 = 1$	Bottom	MSD

$\therefore$ $374_{10} = 176_{16}$

Conversion of fractional part

			Top ↓	
0.37×16	$= 5.92$	$= 0.92$ with a carry 5		MSD
0.64×16	$= 14.72$	$= 0.72$ with a carry 14 $(= E)$		
0.72×16	$= 11.52$	$= 0.52$ with a carry 11 $(= B)$		
0.52×16	$= 8.32$	$= 0.44$ with a carry 8		
0.32×16	$= 5.12$	$= 0.12$ with a carry 5		
0.12×16	$= 1.92$	$= 0.92$ with a carry 1		LSD
			Bottom	

and so on.....

$$0.37_{10} = 158BE5_{16}$$

Combining the integer and the fractional part of the numbers, we get,

$\therefore$ $374.37_{10} = 176.158BE5_{16}$ **Ans.**

(*g*) Given: The number $= (3AB)_{16} = (\,)_{10}$

We know that to convert $3AB$ to its equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then table the sum of these products,

$$\begin{array}{ccc} 3 & A & B \\ \downarrow & \downarrow & \downarrow \\ 16^{+2} & 16^{+1} & 16^{+0} \end{array}$$

$$= (3 \times 16^{+2}) + (A \times 16^{+1}) + (B \times 16^{+0})$$
$$= (3 \times 256) + (10 \times 16) + (11 \times 1)$$
$$= 768 + 160 + 11$$
$$= 939_{10} \textbf{ Ans.}$$

Example 2-50. *Convert the following*

(*a*) $(FB17)_{16} = (\,)_2$

(*b*) $(2374)_8 = (\,)_{10}$

(*c*) $(0.865)_{10} = (\,)_2$

(*d*) $(5624.37)_8 = ()_2$

(*RGTU., Dec. 2010*)

Solution.

(*a*) Given: The number $= (FB17)_{16} = (\,)_2$

$$\begin{array}{lcccc} \text{Step 1.} & F & B & 1 & 7 \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & \overbrace{1111} & \overbrace{1011} & \overbrace{0001} & \overbrace{0111} \end{array}$$

Step 3. 1111101100010111

$\therefore$ $FB17_{16} = 1111101100010111_2$ **Ans.**

(*b*) Given: The number $= (2374)_8 = (\,)_{10}$

We know that in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{lcccc} \text{Step 1.} & 2 & 3 & 7 & 4 \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 8^{+3} & 8^{+2} & 8^{+1} & 8^{+0} \end{array}$$

Step 3. $= (2 \times 8^{+3}) + (3 \times 8^{+2}) + (7 \times 8^{+1}) + (4 \times 8^{+0})$

$$= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1)$$
$$= 1024 + 192 + 56 + 4$$
$$= 1276_{10} \textbf{ Ans.}$$

(*c*) Given: The number $= (0.865)_{10} = (\,)_2$

							Top	
0.865×2	=	1.73	=	0.73	with a carry of	1	↓	LSB
0.73×2	=	1.46	=	0.46	with a carry of	1		
0.46×2	=	0.92	=	0.92	with a carry of	0		
0.92×2	=	1.84	=	0.84	with a carry of	1		
0.84×2	=	1.68	=	0.68	with a carry of	1		
0.68×2	=	1.36	=	0.36	with a carry of	1		MSB
							Bottom	

and so on....

We can continue this process further if we wish because the binary fractional part has not reduced to zero. However we stop here because the desired number of binary places is reached. Reading from top to bottom, we find that the binary fraction is 0.110111.

$\therefore$ $0.865_{10} = 110111_2$ **Ans.**

(*d*) Given: The number = $(5624.37)_8 = (\,)_2$

We know that in order to convert an octal number to its binary equivalent, we have to convert each octal digit to its 3-bit binary equivalent. Thus

5	6	2	4	.	3	7
↓	↓	↓	↓		↓	↓
101	110	010	100		011	111

Thus the octal 5624.37_8 is equivalent to the binary 101110010100.011111_2

$\therefore$ $5624.37_8 = 101110010100.011111_2$ **Ans.**

Example 2-51. *Convert the following*

(*a*) $(329.54)_{10} = (\,)_{16}$

(*b*) $(BD0.1A)_{16} = (\,)_{10}$

(*c*) $(1101)_{\text{gray}} = (\,)_2$

(*d*) $(967.345)_{10} = (\,)_8$

Solution.

(*a*) Given: The number = $(329.54)_{10} = (\,)_{16}$

We know that we can convert the integer part of the given number to its equivalent hexadecimal by the repeated division by 16 method. Whereas the fractional part is to be converted to its equivalent hexadecimal by the repeated multiplication by 16 method.

Conversion of integer part

We know that we can use the repeated division by 16 for converting a integral decimal number to its hexadecimal equivalent and the fractional part is convert by,

Conversion of integer part

					Top	
$329 \div 16$ = 20	with a remainder	$0.5625 \rightarrow 0.5625 \times 16$	=	9	↑	LSD
$20 \div 16$ = 1	with a remainder	$0.25 \rightarrow 0.25 \times 16$	=	4		
$1 \div 16$ = 0	with a remainder	$0.0625 \rightarrow 0.0625 \times 16$	=	1		MSD
					Bottom	

∴ $329_{10} = 149_{16}$

Conversion of fractional part

$$\begin{array}{lllll}
 & & & & \text{Top} \\
0.54 \times 16 = 8.64 & = 0.64 & \text{with a carry of } 8 & \downarrow & \text{MSD} \\
0.64 \times 16 = 10.24 & = 0.24 & \text{with a carry } 10\ (= A) & \downarrow & \\
0.24 \times 16 = 3.84 & = 0.84 & \text{with a carry } 3 & \downarrow & \\
0.84 \times 16 = 13.44 & = 0.44 & \text{with a carry } 13\ (= D) & \downarrow & \\
0.44 \times 16 = 7.04 & = 0.04 & \text{with a carry } 7 & \downarrow & \\
0.04 \times 16 = 0.64 & = 0.64 & \text{with a carry } 0 & \downarrow & \\
0.64 \times 16 = 10.24 & = 0.24 & \text{with a carry } 10\ (= A) & \downarrow & \text{LSD} \\
 & & & & \text{Bottom}
\end{array}$$

Since the fractional part is zero, so we stop the repeated multiplication. Thus,

$$0.54_{10} = 8A3D70A_{16}$$

Combining the integer and the fractional part of the numbers, we get,

$$329.54_{10} = 219.8A3D70A_{16}$$ **Ans.**

(*b*) Given: The number = $(BD02.1A)_{16} = (\ )_{10}$

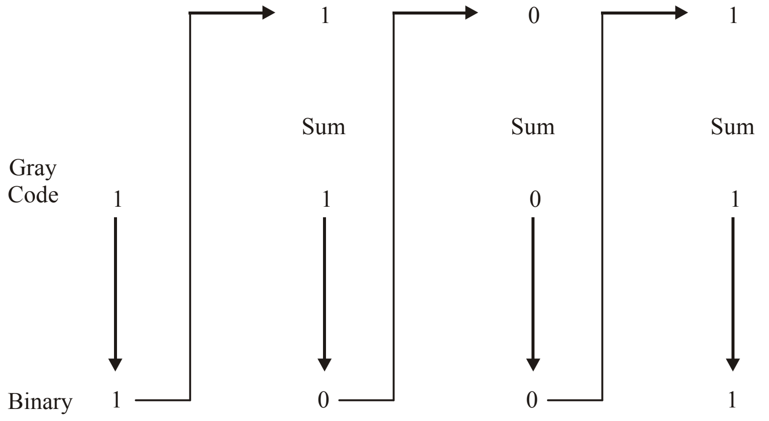
We know that to convert BD02.1A to its equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then table the sum of these products,

$$\begin{array}{ccccccccc}
\text{B} & \text{D} & . & 0 & . & 2 & . & 1 & \text{A} \\
\downarrow & \downarrow & & \downarrow & & \downarrow & & \downarrow & \downarrow \\
16^{+3} & 16^{+2} & & 16^{+1} & & 16^{+0} & & 16^{-1} & 16^{-2}
\end{array}$$

$$\begin{aligned}
&= (\text{B} \times 16^{+3}) + (\text{D} \times 16^{+2}) + (0 \times 16^{+1}) + (2 \times 16^{+0}) + (1 \times 16^{-1}) + (A \times 16^{-2}) \\
&= (11 \times 4096) + (13 \times 256) + (0 \times 16) + (2 \times 1) + (1 \times 16^{-1}) + (10 \times 16^{-2}) \\
&= 405056 + 3328 + 0 + 2 + 0.0625 + 0.0390625 \\
&= 408384.1015625_{10}
\end{aligned}$$ **Ans.**

(*c*) Given: The number = $(1101)_{\text{gray}} = (\ )_2$

We know that the conversion of gray code to binary,



$\therefore \quad 1101_{\text{gray}} = 1001_2$

(*d*) Given: The number $= (967.345)_{10} = (\,)_8$

We know that the integer part of the given number to its equivalent binary by sum-of-weights method. Whereas the fractional part is to be converted to its equivalent octal by repeated multiplication by 8 method.

					Top	
$967 \div 8 = 120.875$	with a carry	0.875	$0.875 \times 8 = 7$		↑	(LSD)
$120 \div 8 = 15 = 15$	with a carry	0	$0 \times 8 = 0$		│	
$15 \div 8 = 1.875 = 1$	with a carry	0.875	$0.875 \times 8 = 7$		│	
$1 \div 8 = 0.125 = 0$	with a carry	0.125	$0.125 \times 8 = 1$		│	(MSD)
					Bottom	

$967_{10} = 1707_8$

Conversion of fraction part

		Top	
$0.345 \times 8 = 2.76$	$= 0.76$ with a carry 2	│	LSD
$0.76 \times 8 = 6.08$	$= 0.08$ with a carry 6	│	
$0.08 \times 8 = 0.64$	$= 0.64$ with a carry 0	│	
$0.64 \times 8 = 5.12$	$= 0.12$ with a carry 5	│	
$0.12 \times 8 = 0.96$	$= 0.96$ with a carry 0	↓	MSD
		Bottom	

and so on...

$0.345_{10} = 0.26050_8$

Thus,

$1707.345_{10} = 231.36050_8$ **Ans.**

Example 2-52. *Convert the following*

(*a*) $(301)_{10} = (\,)_{16}$

(*b*) $(01FA)_{16} = (\,)_{10}$

(*c*) $(1001)_2 = (\,)_{\text{gray}}$

(*d*) $(72.625)_{10} = (\,)_8$

(*RGTU., Dec. 2009*)

Solution.

(*a*) Given: The number $= (301)_{10} = (\,)_{16}$

We know that we can use the repeated division by 16 for converting a decimal number to its hexadecimal equivalent. Using division by hand, we get,

			Top	
$301 \div 16 = 18$	with a remainder	$0.8125 \rightarrow 0.8125 \times 16 = 13 (= D)$	↑	LSD
$18 \div 16 = 1$	with a remainder	$0.125 \rightarrow 0.125 \times 16 = 2$	│	
$1 \div 16 = 0$	with a remainder	$0.0625 \rightarrow 0.0625 \times 16 = 1$	│	MSD
			Bottom	

Since the quotient is zero, so we will stop the further division. Reading the remainders from bottom to top, the hexadecimal number = 12D.

$\therefore$ $301_{10} = 12D_{16}$ **Ans.**

(*b*) Given: The number = $(01FA)_{16} = (\ )_{10}$

We know that to convert the 01FA to its equivalent decimal number, we will multiply the decimal value of each hexadecimal digit by its position weight and then take the sum of these products.

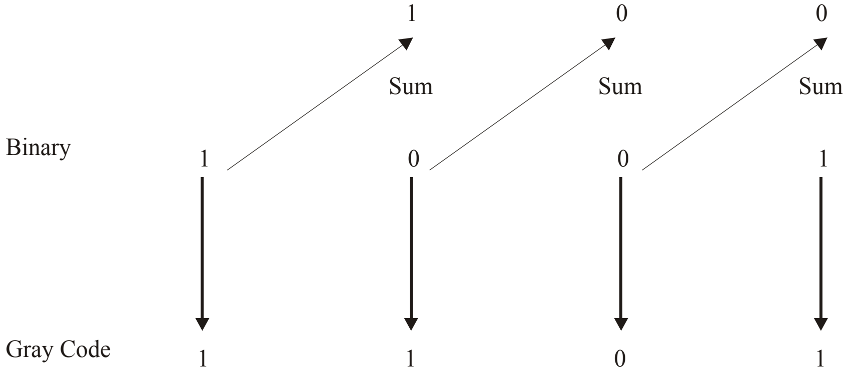
$$\begin{array}{lcccc} \text{Step 1.} & 0 & 1 & \text{F} & \text{A} \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 8^{+3} & 8^{+2} & 8^{+1} & 8^{+0} \end{array}$$

Step 3.
$$\begin{aligned} &= (0 \times 8^{+3}) + (1 \times 8^{+2}) + (F \times 8^{+1}) + (A \times 8^{+0}) \\ &= (0 \times 8^{+3}) + (1 \times 64) + (15 \times 8) + (10 \times 1) \\ &= 0 + 64 + 120 + 10 \\ &= 0 + 64 + 120 + 10 \\ &= 194 \end{aligned}$$

$01FA_{16} = 194_{10}$ **Ans.**

(*c*) Given: The number = $(1001)_2 = (\ )_{gray}$

We know that the conversion of binary to gray,



$\therefore$ $1001_2 = 1101_{\text{gray}}$ **Ans.**

(*d*) Given: The number = $(72.625)_{10} = (\ )_8$

We know that the integer part of the given number to its equivalent binary by sum-of-weights method. Whereas the fractional part is to be converted to its equivalent octal by repeated multiplication by 8 method.

Conversion of integer part

				Top	
$72 \div 8 = 9.0 = 9$	with a carry	0	$0 \times 8 = 0$	$\uparrow$	(LSD)
$9 \div 8 = 1.125 = 1$	with a carry	0.125	$0.125 \times 8 = 1$		
$1 \div 8 = 0.125 = 0$	with a carry	0.125	$0.125 \times 8 = 1$		
					(MSD)
				Bottom	

$72_{10} = 110_8$

Conversion of fraction part

$$0.625 \times 8 \;=\; 5.0 = 0 \text{ with a carry } 5$$

Since the fractional part is zero, so we stop the repeated multiplication. Thus,

$0.625_{10} = 0.5_8$

Thus,

$\therefore$ $72.625_{10} = 231.55_8$ **Ans.**

Example 2-53. *Convert the following*

(*a*) $(225.225)_{10} = ()_8$

(*b*) $(623.77)_8 = ()_{10}$

(*c*) $(11010111.110)_2 = ()_8$

(*d*) $(2\,AC5.\,D)_{16} = ()_8$

(*RGTU., June 2011*)

Solution.

(*a*) Given: The number = $(623.77)_8 = (\,)_{10}$

We know that in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{llllll} \text{Step 1.} & 6 & 2 & 3 \quad . & 7 & 7 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 8^{+2} & 8^{+1} & 8^{+0} & 8^{-1} & 8^{-2} \end{array}$$

Step 3.

$$\begin{aligned} &= (6 \times 8^{+2}) + (2 \times 8^{+1}) + (3 \times 8^{+0}) + (7 \times 8^{-1}) + (7 \times 8^{-2}) \\ &= (6 \times 64) + (2 \times 8) + (3 \times 1) + (7 \times 0.125) + (7 \times 0.015625) \\ &= 384 + 16 + 3 + 0.875 + 0.109375 \\ &= 403.984375_{10} \textbf{ Ans.} \end{aligned}$$

(*b*) Given: The number = $(11010111.110)_2 = ()_8$

We know that to convert the given octal number to its octal equivalent, we have to form 3-bit combination while moving right to left by starting from right most position in integer part and in fraction part we have to move from left to right. Thus,

$$\begin{array}{ccccccc} 011 & 010 & . & 111 & . & 110 \\ \downarrow & \downarrow & & \downarrow & & \downarrow \\ 111 & 110 & & 100 & & 011 \end{array}$$

Thus the octal 7643_8 is equivalent to the binary 111110100011_2

$\therefore$ $7643_8 = 111110100011_2$ **Ans.**

(*c*) Given: The number = $(2\,AC5.\,D)_{16} = (\,)_8$

First of all we will convert $2AC5_{16}$ to its equivalent and then to its octal equivalent.

$$\begin{array}{cccccc} 2 & A & C & 5 & . & D \\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow \\ 0010 & 010 & 1100 & 0101 & & 1101 \end{array}$$

then convert to octal equivalent,

$$\begin{array}{cccccccc} \underbrace{010} & \underbrace{101} & \underbrace{011} & \underbrace{000} & . & \underbrace{101} & \underbrace{110} & \underbrace{100} \\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\ 2 & 5 & 3 & 0 & & 5 & 6 & 4 \end{array}$$

Add two zeros to complete 3-bit group.

Thus we find that,

$\therefore$ $2AC5.D_{16} = 25305.64_8$ **Ans.**

Example 2-54. *Convert the following decimal numbers to their binary equivalents.*

(*a*) $(83)_{10}$

(*b*) $(79.515)_{10}$

(*c*) $(109.125)_{10}$

Solution.

(*a*) Given: The number $(83)_{10}$

We know that the decimal number can be expressed by the sum-of-weights as follows:

$$83 = 64 + 16 + 2 + 1 = 2^6 + 2^4 + 2^1 + 2^0$$

Placing Is in the appropriate weight positions, 2^6, 2^4, 2^1 and 2^0 and 0s in the 2^5, 2^3 and 2^2 positions, determines the binary number for decimal 83. Thus,

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$$

$\therefore$ $83_{10} = 1010011_2$ **Ans.**

(*b*) Given: The decimal number $(79.515)_{10}$

We know that the integer part of the given number to its equivalent binary by sum-of-weights method. Whereas the fractional part is to be converted to its equivalent octal by repeated multiplication by 2 method.

Conversion of integer part

We know that the decimal number can be expressed by the sum-of-weights as follow:

$$79 = 64 + 8 + 4 + 2 + 1 = 2^6 + 2^3 + 2^2 + 2^1 + 2^0$$

Placing 1*s* in the appropriate weight positions, 2^6, 2^3, 2^2, 2^1 and 2^0 and 0s in the 2^5 and 2^4 positions, determines the binary number for decimal 79. Thus,

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

$\therefore$ $83_{10} = 1001111_2$

Conversion of fractional part

the fraction part = 0.515

$$\begin{array}{rclcllc|l} & & & & & & & \text{Top}\\ 0.515\times 2 &=& 1.0 &=& 0.03 & \text{with a carry} & 1 & \text{(MSB)}\\ 0.03\times 2 &=& 0.0 &=& 0.06 & \text{with a carry} & 0 & \\ 0.06\times 2 &=& 0.1 &=& 0.12 & \text{with a carry} & 0 & \\ 0.12\times 2 &=& 0.2 &=& 0.24 & \text{with a carry} & 0 & \downarrow\\ & & & & & & & \text{(LSB)}\\ & & & & & & & \text{Bottom} \end{array}$$

and so on...

We can continue this process further if we wish because the binary fractional part has reduced to zero. However we stop here because the desire number of binary places is reached. Reading from top to bottom, we find that the binary fraction is 0.1000

∴ $83.515_{10} = 1001111.1000_2$ **Ans.**

(*c*) Given: The decimal number = $(109.125)_{10}$

We know that the integer part of the given number to its equivalent binary by sum-of-weights method. Whereas the fractional part is to be converted to its equivalent octal by repeated multiplication by 2 method.

Conversion of integer part

We know that the decimal number can be expressed by the sum-of-weights as follow:

$$109 = 64 + 32 + 8 + 4 + 1 = 2^6 + 2^5 + 2^3 + 2^2 + 2^0$$

Placing 1*s* in the appropriate weight positions, 2^6, 2^5, 2^3, 2^2 and 2^0 and 0s in the 2^4 and 2^1 position, determines the binary number for decimal 79. Thus,

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0\\ \Bigg\downarrow & \Bigg\downarrow & \Bigg\downarrow & \Bigg\downarrow & \Bigg\downarrow & \Bigg\downarrow & \Bigg\downarrow\\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

∴ $109_{10} = 1101101_2$

Conversion of fractional part

the fraction part = 0.515

$$\begin{array}{rclcllc|l} & & & & & & & \text{Top}\\ 0.125\times 2 &=& 0.2 &=& 0.25 & \text{with a carry} & 0 & \text{(MSB)}\\ 0.25\times 2 &=& 0.5 &=& 0.5 & \text{with a carry} & 0 & \\ 0.5\times 2 &=& 1.0 &=& 0 & \text{with a carry} & 1 & \downarrow\\ & & & & & & & \text{(LSB)}\\ & & & & & & & \text{Bottom} \end{array}$$

Since the fractional part is zero, so we will stop the repeated multiplication. Reading from top to bottom, we find that the binary fraction is 0.001.

∴ $109.125_{10} = 1101101.001_2$ **Ans.**

Example 2-55. *Do the following conversions*

(*a*) $(99)_{10} = (\,)_2$

(*b*) $(1527)_8 = (\;)_{10}$

(*c*) $(25)_8 = (\;)_2$

(*d*) $(10110011)_2 = (\;)_{16}$

Solution.

(*a*) Given: The decimal number = 99

We know that the decimal number can be expressed by the sum-of-weights as follow:

$$99 = 64 + 32 + 2 + 1 = 2^6 + 2^5 + 2^1 + 2^0$$

Placing 1*s* in the appropriate weight positions, 2^6, 2^5, 2^1 and 2^0 and 0s in the 2^4, 2^3 and 2^2 positions, determines the binary number for decimal 83. Thus,

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{array}$$

∴ $99_{10} = 1100011_2$ **Ans.**

(*b*) Given: The number $(1527)_8 = (\;)_{10}$

We know that in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{lcccc} \text{Step 1.} & 1 & 5 & 2 & 7 \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 8^{+3} & 8^{+2} & 8^{+1} & 8^{+0} \end{array}$$

Step 3.
$$\begin{aligned} &= (1 \times 8^{+3}) + (5 \times 8^{+2}) + (2 \times 8^{+1}) + (7 \times 8^{+0}) \\ &= (1 \times 512) + (5 \times 64) + (2 \times 8) + (7 \times 1) \\ &= 512 + 320 + 16 + 7 \\ 1527_8 &= 855_{10} \end{aligned}$$ **Ans.**

(*c*) Given: The number = $(25)_8 = (\;)_2$

We know that to convert the given octal number to its binary equivalent, we have to convert each octal digital to its 3-bit equivalent. Thus,

$$\begin{array}{cc} 2 & 5 \\ \downarrow & \downarrow \\ 010 & 101 \end{array}$$

Thus the octal 25_8 is equivalent to the binary 010101_2

Dropping off the zero at the left most position, the binary number can be written as 10101

∴ $25_8 = 10101_2$ **Ans.**

(*d*) Given: The number = $(10110011)_2 = (\;)_{16}$

$$\begin{array}{lcc} \text{Step 1.} & 1011 & 0011 \\ & \downarrow & \downarrow \\ \text{Step 2.} & B & 7 \end{array}$$

$\therefore$ $10110011_2 = B7_{16}$ **Ans.**

Example 2-56. *Convert the following number into the respective index given*

(*a*) $(89)_{10} = ()_2 = ()_{16}$

(*b*) $(1011.101101)_2 = ()_8 = ()_{10}$

(*RGTU., March-April 2010*)

Solution.

(*a*) Given: $(89)_{10} = ()_2 = ()_{16}$

We know that the decimal number can be expressed by the sum-of-weights as follow:

$$89 = 64 + 16 + 8 + 1 = 2^6 + 2^4 + 2^3 + 2^0$$

Placing 1*s* in the appropriate weight positions, 2^6, 2^4, 2^3 and 0s in the 2^5, 2^2 and 2^1 positions, determines the binary number for decimal 83. Thus,

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}$$

$\therefore$ $89_{10} = 1011011_2$ **Ans.**

$$\begin{array}{lcc} \text{Step 1.} & 0101 & 1011 \\ & \downarrow & \downarrow \\ \text{Step 2.} & 5 & B \end{array}$$

$\therefore$ $1011011_2 = 5B_{16}$ **Ans.**

(*b*) $(1011.101101)_2 = ()_8 = ()_{10}$

The binary number = 1011.101101

We know that in order to convert the given binary number to its octal equivalent, we have to form 3-bit combinations. Thus,

$$\begin{array}{lccccc} \text{Step 1.} & \underbrace{001} & \underbrace{011} & . & \underbrace{101} & \underbrace{101} \\ & \downarrow & \downarrow & & \downarrow & \downarrow \\ \text{Step 2.} & 1 & 3 & & 5 & 5 \end{array}$$

$\therefore$ $1011.101101_2 = 13.55_8$ **Ans.**

We know that in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{lccccc} \text{Step 1.} & 1 & 3 & . & 5 & 5 \\ & \downarrow & \downarrow & & \downarrow & \downarrow \\ \text{Step 2.} & 8^{+1} & 8^{+0} & & 8^{-1} & 8^{-2} \end{array}$$

Step 3. $= (1 \times 8^{+1}) + (3 \times 8^{+0}) + (5 \times 8^{-1}) + (5 \times 8^{-2})$

$$= (1 \times 8) + (3 \times 1) + (5 \times 0.125) + (5 \times 0.0156)$$
$$= 8 + 3 + 0.625 + 0.078125$$
$$= 11.703125_{10}\ \textbf{Ans.}$$

Example 2-57. *Convert the following*

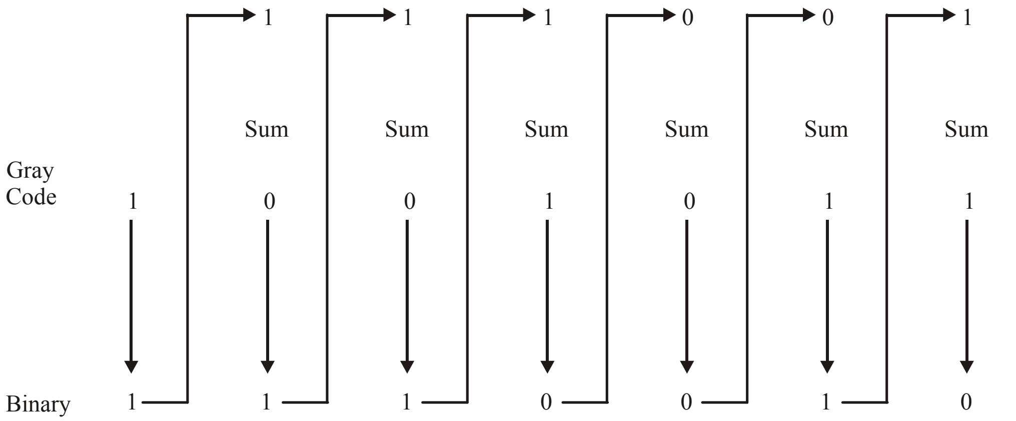
(*a*) $(1001011)_{gray} = (\,)_2$
(*b*) $(10110110.0011)_2 = (\,)_{BCD}$
(*c*) $(10110101_{2421} = (\,)_{10}$
(*d*) $(5762)_8 = (\,)_{16}$
(*e*) $(76)_{10} = (\,)_{gray}$

(*RGTU., June 2009*)

Solution.

(*a*) Given: The number $= (1001011)_{gray} = (\,)_2$

We know that the gray to binary conversion,



Thus,

$\therefore$ $1001011_{gray} = 1110010_2$ **Ans.**

(*b*) Given: The number $= (10110110.0011)_2 = (\,)_{BCD}$

Step 1. 1 0 1 1 0 1 1 0 . 0 0 1 1

Step 2. $\underbrace{1\ 0\ 1\ 1}_{\downarrow}$ $\underbrace{0\ 1\ 1\ 0}_{\downarrow}$. $\underbrace{0\ 0\ 1\ 1}_{\downarrow}$

Step 2. eleven indicates error 6 3

Therefore we find that 10110110.0011 is not a valid code.

(*c*) Given: The number $= (10110101)_{2421} = (\,)_{10}$

We know that in the weight code,

$$\underbrace{1011}_{\downarrow} \qquad \underbrace{0101}_{\downarrow}$$

$$1 \times 2 + 0 \times 4 + 1 \times 2 + 1 \times 1 \qquad 0 \times 2 + 1 \times 4 + 0 \times 2 + 1 \times 1$$

$\therefore$ $10110101_{2421} = 55_{10}$ **Ans.**

(*d*) Given: The number $= (5762)_8 = (\,)_{16}$

Step 1.	5	7	6	2
	↓	↓	↓	↓
Step 2.	101	111	110	010
Step 3.	1011	1111	0010	
	↓	↓	↓	
	B	F	3	

$\therefore \quad 5762_8 = \text{BF3}_{16}$

(*e*) Given: The number $= (76)_{10} = (\,)_{\text{gray}}$

We know that the decimal number can be expressed by the sum-of-weights as follow:

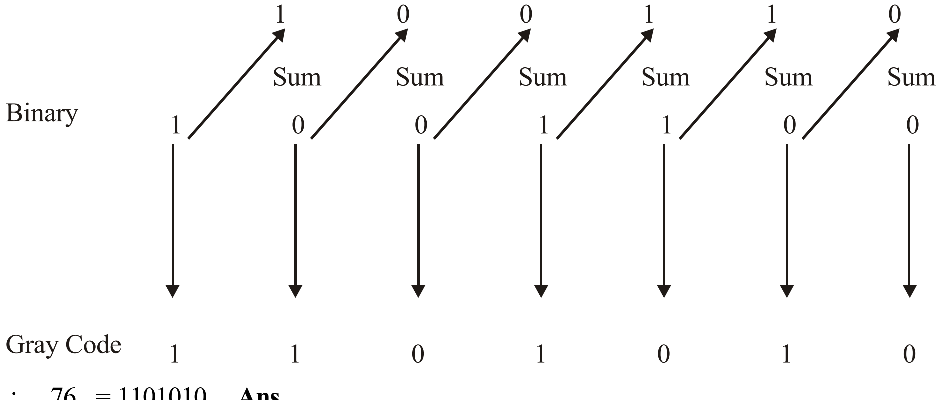
$$76 \;=\; 64 + 8 + 4 = 2^6 + 2^3 + 2^2$$

Placing 1*s* in the appropriate weight positions, 2^6, 2^3 and 2^2 and 0s in the 2^5, 2^4, 2^1 and 2^0 positions, determines the binary number for decimal 76. Thus,

2^6	2^5	2^4	2^3	2^2	2^1	2^0
↓	↓	↓	↓	↓	↓	↓
1	0	0	1	1	0	0

$\therefore \quad 76_{10} = 1001100_2$

We know that the conversion of binary to gray,



$\therefore \quad 76_{10} = 1101010_{\text{gray}}$ **Ans.**

Example 2-58. *Convert the following items:*

(*a*) $(153.6875)_{10} = (\,)_8$

(*b*) $(26153.7406)_8 = (\,)_{16}$

(*c*) $(630.4)_8 = (\,)_{10}$

(*d*) $(7643)_8 = (\,)_2$

(*e*) $(001111001010101)_2 = (\,)_{16}$

(*RGTU., Dec. 2010*)

Solution.

(*a*) Given: The decimal number $= (153.6875)_{10} = (\,)_8$

We know that the integer part of the given number to its equivalent binary by sum-of-weights method. Whereas the fractional part is to be converted to its equivalent octal by repeated multiplication by 8 method.

$$\begin{array}{rcllllcl}
 & & & & & & & \text{Top} \\
153 \div 8 &=& 19.125 = 19 & \text{with a carry} & 0.125 & 0.125 \times 8 = 1 & \Big\uparrow & \text{(LSD)} \\
19 \div 8 &=& 2.37 = 0.375 & \text{with a carry} & 0.375 & 0.375 \times 8 = 3 & \Big| & \\
2 \div 8 &=& 0.25 = 0 & \text{with a carry} & 0.25 & 0.25 \times 8 = 2 & \Big| & \\
 & & & & & & & \text{(MSD)} \\
 & & & & & & & \text{Bottom}
\end{array}$$

$153_{10} = 231_8$

Conversion of fraction part

$$\begin{array}{rcrcllc}
 & & & & & \text{Top} & \\
0.6875 \times 8 &=& 5.5 &=& 0.5 \text{ with a carry } 5 & \Big\uparrow & \text{LSD} \\
0.5 \times 8 &=& 4.0 &=& 0.0 \text{ with a carry } 4 & \Big| & \text{MSD} \\
 & & & & & \text{Bottom} &
\end{array}$$

Since the fractional part is zero, so we stop the repeated multiplication. Thus,

$0.6875_{10} = 0.54_8$

Thus,

$\therefore \quad 153.6875_{10} = 231.55_8$ **Ans.**

(*b*) Given: number $= (26153.7406)_8 = (\,)_{16}$

$$\begin{array}{lcccccccccc}
\text{Step 1.} & 2 & 6 & 1 & 5 & 3 & . & 7 & 4 & 0 & 6 \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & \overbrace{010} & \overbrace{110} & \overbrace{001} & \overbrace{101} & \overbrace{011} & & \overbrace{111} & \overbrace{100} & \overbrace{000} & \overbrace{110}
\end{array}$$

$$\begin{array}{lcccccccc}
\text{Step 3.} & \underbrace{0010} & \underbrace{1100} & \underbrace{0110} & \underbrace{1011} & . & \underbrace{1111} & \underbrace{0000} & \underbrace{0110} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\
 & 2 & C & 6 & B & & F & 0 & 6
\end{array}$$

$\therefore \quad 26153.7406_8 = F06_{16}$ **Ans.**

(*c*) Given: The number $= (630.4)_8 = (\,)_{10}$

We know that in order to convert an octal number to its decimal equivalent, we multiply each digit by its position weight and then sum the products, thus,

$$\begin{array}{lccccc}
\text{Step 1.} & 6 & 3 & 0 & . & 4 \\
 & \downarrow & \downarrow & \downarrow & & \downarrow \\
\text{Step 2.} & 8^{+2} & 8^{+1} & 8^{+0} & & 8^{-1}
\end{array}$$

Step 3.
$$\begin{aligned}
&= (6 \times 8^{+2}) + (3 \times 8^{+1}) + (0 \times 8^{+0}) + (4 \times 8^{-1}) \\
&= (6 \times 64) + (3 \times 8) + (4 \times 0.125) \\
&= 384 + 24 + 0 + 0.5 \\
&= 408.5_{10} \textbf{ Ans.}
\end{aligned}$$

(*d*) Given: The number = $(7643)_8 = (\ )_2$

We know that to convert the given octal number to its binary equivalent, we have to convert each octal digit to its 3-bit binary equivalent. Thus,

$$\begin{array}{ccccccc} 7 & & 6 & . & 4 & . & 3 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \underbrace{}_{111} & & \underbrace{}_{110} & & \underbrace{}_{100} & & \underbrace{}_{011} \end{array}$$

Thus the octal 7643_8 is equivalent to the binary 111110100011_2

$\therefore$ $7643_8 = 111110100011_2$ **Ans.**

(*e*) Given: The number = $(001111001010101)_2 = (\ )_{16}$

Step 1. 0 0 1 1 1 1 0 0 1 0 1 0 1 0 1

$$\begin{array}{lcccc} \text{Step 2.} & 0001 & 1110 & 0101 & 0101 \\ \text{One zero added to} & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{form 4-bit group} & & & & \\ \text{Step 3.} & 1 & E & 5 & 5 \end{array}$$

$\therefore$ $001111001010101_2 = 1E55_{16}$ **Ans.**

2-30. Binary-Coded-Decimal Number System

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary ***code.*** Since there are only ten code groups in the BCD system, it is very easy to convert between the decimal and BCD. Because we like to read and write in decimal, the BCD system provides an excellent interface to binary systems. Examples of such interfaces are keypad inputs and digital readouts.

The BCD system is also called the 8421 code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of four bits. The designation 8421 indicates the binary weights of four bits (2^3, 2^2, 2^1, 2^0). The ease of conversion between BCD (or 8421) code numbers and familiar decimal numbers is the main advantage of this code. There are ten binary combinations that represent the ten decimal digits as shown in Table 2-5.

Table 2-5. BCD code for ten decimal digits

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

It should be noted that with four bits, sixteen numbers (0000 through 1111) can be represented but in the BCD (or 8421) code, only ten of these are used.

There are six code combinations that are not used. These are 1010, 1011,1100, 1101, 1110 and 1111. These six code combinations are refered to as invalid in the BCD (or 8421)code.

The procedure for converting a given decimal number to its BCD equivalent is as follows:

Step 1. Write the decimal number.

Step 2. Convert each decimal digit to its 4-bit binary equivalent.

Step 3. Write the binary number as an answer.

In order to illustrate the procedure, consider the conversion of decimal number 35.

$$\begin{array}{lll}
\text{Step 1.} & 3 & 5 \\
 & \downarrow & \downarrow \\
\text{Step 2.} & \overbrace{0011} & \overbrace{0101} \\
 & & \\
\text{Step 3.} & \underbrace{00}_{\uparrow}110101 &
\end{array}$$

Drop off these zeros

$\therefore$ 35_{10} = 110101 BCD

Example 2-59. *Convert the following decimal numbers to binary-coded-decimal* (*BCD*) *code*:

(*a*) 843 *and* (*b*) 2469

Solution.

(*a*) Given : The decimal number = 843

We know that to convert the decimal number, we need to change each digit to its equivalent 4-bit binary, i.e.

$$\begin{array}{llll}
\text{Step 1.} & 8 & 4 & 3 \\
 & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & 1000 & 0100 & 0011 \\
 & & & \\
\text{Step 3.} & 100001000011 \text{ BCD} & &
\end{array}$$

$\therefore$ 843_{10} = 10000100 0011 BCD **Ans.**

(*b*) Given : The decimal number = 2469

$$\begin{array}{lllll}
\text{Step 1.} & 2 & 4 & 6 & 9 \\
 & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{Step 2.} & \overbrace{0010} & \overbrace{0100} & \overbrace{0110} & \overbrace{1001} \\
 & & & & \\
\text{Step 3.} & 0010 & 0100 & 0110 & 1001
\end{array}$$

$\therefore$ 2469_{10} = 0010 0100 0110 1001 BCD **Ans.**

Example 2-60. *Encode* $(39.584)_{10}$ *into* 8421 *BCD number.*

(Nagpur University, Oct/Nov 1997)

Solution. Given : A decimal number = 39, 584

We know that in order to convert a given decimal number to its equivalent BCD number, we have to replace each decimal digit by its 4-bit binary equivalent. Thus

$$\begin{array}{ccccc} 3 & 9 & 5 & 8 & 4 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \overbrace{0011} & \overbrace{1001} & \overbrace{0101} & \overbrace{1000} & \overbrace{0100} \end{array}$$

$\therefore$ $39584_{10} = 111001010110000100_{BCD}$ **Ans.**

Example 2-61. *Repeated the decimal number 8620 in BCD*

(*PTU, Dec. 2008, GTU., Dec., 2011, May 2011*)

Solution. Given: The decimal number = 8620

We know that to convert the decimal number we need to change each digit to it equivalent 4-bit binary i.e.

$$\begin{array}{lcccc} \text{Step 1.} & 8 & 6 & 2 & 0 \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{Step 2.} & 1000 & 0110 & 0010 & 0000 \end{array}$$

Step 3.

$\therefore$ $8620_{10} = 1000011000100000$ BCD **Ans.**

Example 2-62. *Find the BCD equivalent of* 25_8. (*Gauhati, University. 2007*)

Solution. Given: The octal number = 25_8

We know that to convert octal number into BCD. First we convert this number into decimal number.

$$\begin{array}{lcl} & & \text{Top} \\ 25 \div 8 = 3.125 = 3 \text{ with the remainder } 0.125 \rightarrow 0.125 \times 8 = 1 & \Big\downarrow & \text{LSD} \\ 3 \div 8 = 0.375 = 0 \text{ with the remainder } 0.375 \rightarrow 0.375 \times 8 = 3 & & \text{MSD} \\ & & \text{Bottom} \end{array}$$

$25_8 = 31_{10}$

We know that to convert the decimal number, we need change each digit to its equivalent 4-bil binary, i.e.

$$\begin{array}{lcc} \text{Step 1.} & 3 & 1 \\ & \downarrow & \downarrow \\ \text{Step 2.} & 0011 & 0001 \end{array}$$

Step 3 . 00110001 BCD and

$\therefore$ $31_{10} = 00110001$ BCD **Ans.**

Example 2-63. *Convert* $(175)_{10}$ *into BCD code.* (*GBTU, MTU, 2008*)

Solution. Given: The decimal number 175

We know that to convert the decimal number we need to change each digit to it equivalent 4-bit binary i.e.

Step 1. 1 7 5

↓ ↓ ↓

Step 2. 0001 0111 0101

Step 3. 000101110101.

∴ 175_{10} = 101110101 BCD **Ans.**

2-31. Determination of a Decimal Number from a BCD number

Following is the procedure for the determination of a decimal number from a given BCD numbers.

Step 1. Write the BCD number.

Step 2. Start at the right-most bit and break the code into groups of four bit each. We can add zeros at the left-most position to complete the four-bit group.

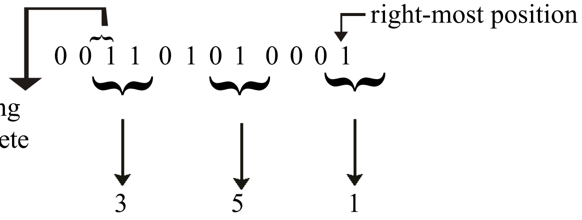
Step 3. Convert each four-bit group to its equivalent decimal digit.

Note that if the four-bit group is in the range of 1010 to 1111, it is a forbidden code group which indicates an error in the BCD number.

Let us illustrate the procedure by considering the BCD number 1101010001

Step 1. 1 1 0 1 0 1 0 0 0 1

Step 2.



Add two leading zeros to complete 4-bit group.

∴ 1101010001 BCD = 351_{10} **Ans.**

Example 2-64. *Convert each of the following BCD numbers to their equivalent BCD numbers*: (*a*) 100001100011 1001 (*b*) 1001 0100 1111 1000

Solution.

(*a*) Given : The BCD number 1000 0110 0011 1001

Step 1. 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1

Step 2. $\underbrace{1\,0\,0\,0}\quad\underbrace{0\,1\,1\,0}\quad\underbrace{0\,0\,1\,1}\quad\underbrace{1\,0\,0\,1}$

↓ ↓ ↓ ↓

Step 3. 8 6 3 9

∴ 1000 0110 0011 1001 BCD = 8639_{10} **Ans.**

(*b*) Given : The BCD number 1001 0100 1111 1000

Step 1. 1 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0

Step 2. Step 3.

$$\underbrace{1\;0\;0\;1}_{9}\quad\underbrace{0\;1\;0\;0}_{4}\quad\underbrace{1\;1\;1\;1}_{\text{fifteen indicates error}}\quad\underbrace{1\;0\;0\;0}_{8}$$

Therefore we find that 1001010011111000 BCD is not a valid code.

2-32. Comparison of BCD and Binary

It is important to realize that BCD is not another number system like binary, octal, decimal and hexadecimal. It is, in fact, the decimal system with each digit encoded in its binary equivalent. It is also important to understand that a BCD number is not the same as a straight binary number. A straight binary code takes the complete decimal number and represents it in binary; the BCD code converts each decimal digit to binary individually. In order to illustrate this point consider the number 173 and compare its straight binary and BCD codes.

$$173 = 0001\ 0111\ 0011$$

The BCD code requires 12 bits while the straight binary code requires only eight bits to represent 173. BCD rquires more bits than straight binary to represent decimal numbers of more than one digit. This is because BCD does not use all possible four-bit groups, as pointed out earlier, and is therefore somewhat efficient.

The main advantage of the BCD code is the relative ease of converting to and from decimal. Only the four-bit code groups for the decimal digits 0 through 9 need to be remembered. This case of conversion is especially important from a hardware standpoint because in a digital system it is the logic circuits that perform the conversions to and from decimal.

2-33. Comparison of Number systems

Table 2-6 gives the representation of the decimal number 1 through 15 in the binary, octal, hex number systems and in BCD code for comparison purpose. Study it carefully and make sure you understand how it was obtained. Especially note how the BCD representation always uses four-bits for each decimal digit.

Table 2-6. Comaprison of dfferent number systems

Decimal	*Binary*	*Octal*	*Hexadecimal*	*BCD*
0	0	0	0	0000
1	1	1	1	0001
2	01	2	2	0010
3	10	3	3	0011
4	100	4	4	0100
5	101	5	5	0101
6	110	6	6	0110
7	111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

Example 2-65. *How many bits are required to represent the decimal numbers in the range from* 0 *to* 999 *using* (*a*) *Straight binary code and* (*b*) *BCD code*?

Solution. Given : The decimal numbers in the range from 0 to 999.

(*a*) *Number of bits required using straight binary code*

We know that the maximum number of decimal numbers which we can represent with 9 bits is 2^9 (i.e 512) and with 10 bits is 2^{10} (i.e. 1024). Since the given range is from 0 to 999, therefore we require at least 10 bits to represent the decimal numbers in the range from 0 to 999. **Ans.**

(*b*) *Number of bits required using BCD code*

We know that the number of bits using BCD code depends upon the maximum value of the given decimal number (i.e. 999). First each digit is to be converted to its equivalent 4-bit BCD code, *i.e.*,

$$\begin{array}{ccc} 9 & 9 & 9 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{1001} & \overbrace{1001} & \overbrace{1001} \end{array}$$

Note that we need 12-bits to represent the decimal number 999 in BCD code.

$\therefore$ Number of bits required to represent the decimal numbers in the range from 0 to 999 = 12 **Ans.**

2-34. The Byte

The byte is a group (or string) of 8-bits. The byte is very commonly used in the field of microprocessors and microcomputers. The reason for this is that the microprocessors and microcomputers handle and store data and information in groups of eight bits. Thus instead of specifying the data as 8-bit, 16-bit, 32-bit etc., we say it as 1-byte, 2-byte, 3-byte respectively.

It will be interesting to know that a group of four-bits is called a ***nibble.*** Hence a byte contains two nibbles. Another point worth noting is that we need two hexadecimal digits to represent a byte.

Example 2-66. *What is the largest decimal value that can be represented in binary using (a) one byte and* (*b*) *two bytes*?

Solution.

(*a*) Given, a largest binary number of one byte long.

We know that one byte is 8 bits. So the largest binary value will be equivalent to decimal.

$$2^8 - 1 = 256 - 1 = 255 \textbf{ Ans.}$$

(*b*) Given a largest number of two byte long.

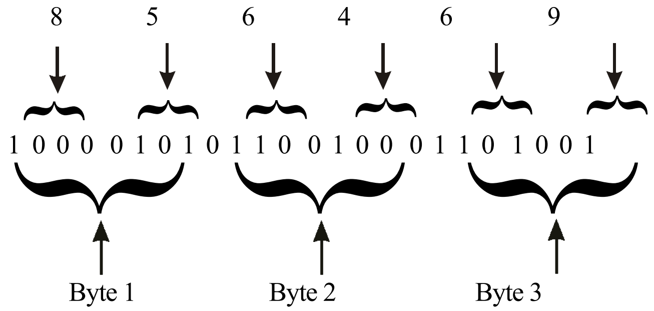
We know that two bytes is 16 bits. So the largest binary value will be equivalent to decimal,

$$2^{16} - 1 = 65{,}536 - 1 = 65{,}535 \textbf{ Ans.}$$

Example 2-67. *How many bytes are needed to represent the decimal value* 856469 *in BCD*?

Solution.

We know that each decimal digit can be converted to its 4-bit BCD code. Since the given number 856469 is a 6-digit number, therefore this number requires 6 × 4 = 24 bits. Further since a byte is equivalent to 8-bits, therefore 24-bits are equal to 3-bytes. This is shown below.



2-35. Alphanumeric codes

In addition to numerical data, a computer must be able to handle non-numerical information used in day-to-day processing. In other words, a computer should recognize codes that represent numbers, letters of the alphabet, punctuation marks, and other special characters. These codes are called alphanumeric codes. A complete alphanumeric code would include the 26 lowercase letters. 26 uppercase letters, 10 numeric digits, 7 punctuation marks, and anywhere from 20 to 40 other characters, such as +, /, #, %, * and so on. We can say that an alphanumeric code represents all of the various characters and functions that are found on a computer keyboard. The most widely used alphanumeric code is the *American standard code for Information Interchange (ASCII).* We will now discuss the ASCII code in the following pages.

2-36. Digital Codes

There are several different types of *digital* codes are used in digital system. All these digital codes may be classified into different categories as follows.

1. **Weighted Codes.** In weighted codes each binary bit is assigned by a weight and values depend on the position of the binary bit. The BCD code has weighted of 8, 4, 2 and 1. The bit assignment 0110, for example, is interpreted by the weights to represent decimal 6 because 8 × 0 + 4 × 1 + 2 × 1 + 1 × 0 = 6. Weights codes are of two types:

   (*a*) Binary – 8421

   (*b*) BCD

2. **Non-Weighted Codes.** In this category of digital code, no specific weights are assigned to the bit position. Each digit position within the number is not assigned fixed value. Non-weighted codes are of two types.

   (*a*) Excess-3 (Ex-3)

   (*b*) Gray Code

3. **Reflective Codes.** A code is said to be a reflective code when the code for 9 is the complement of 0, the code for 8 is complement for 1, 7 for 2, 6 for 3 and 5 for 4. Reflective codes are of the following these types:

   (*a*) 2421

   (*b*) 5211

   (*c*) Ex-3

   For example reflective code

Position weights	2	4	2	1
Code for 9	1	1	1	1
Code for 0	0	0	0	0

   Number 9 is the complement of 0.

4. **Sequential Codes.** In sequential codes, each succeeding code is one binary number greater than its preceding code. In Ex-3 code each decimal digit is obtained by adding decimal 3 to the natural code of the digit. The sequential codes are:

   (*a*) 8421

   (*b*) Excess-3 (Ex-3)

   For example the Ex-3 code of binary number 0010 (decimal code 2) is obtained by adding decimal 3 to the binary code. We add 3 to the decimal code 2 we get 5. The binary value of 5 is 0101. Thus the Ex-3 code of the binary number 0010 is 0101. The Ex-3 code for decimal number 0 through 7 along with these binary equivalent are shown in Table 2-7.

Table 2-7

Decimal	*Binary*	*Excess-3*
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010

5. **Alphanumeric Codes.** The various types of alphanumeric codes are:

   (*a*) ASCII (American Standard Codes for Information Interchange)

   (*b*) EBCDIC (Extended Binary Coded Decimal Interchanged Code)

   (*c*) Hollerith

6. **Error Detecting and Correcting Codes.** The various types of error detecting and correcting codes are:

   (*a*) Parity Code

   (*b*) Hamming Code

2-37. Gray Code

The gray code was designed by Frank Gray at Bell Labs in 1953. It is an un-weighed binary code in which two successive value differ only by one bit. The output of many system are continuous and these data must be converted into digital form before they are applied to a digital system. Continuous or analog information is converted into digital form by analog-to-digital converter. It is sometimes convenient to use Gray code shown in Table 2-8. The main advantage of using capray code is that only one bit in the code group changes, from one number to the next number. For example, from 7 to 8, the Gray code changes from 0100 to 1100 only the first bit is changes, from 0 to 1. The other three bits remain the same. But in binary numbers the change from 7 to 8 will be from 0111 to 1000, all four bits changes. The Gray code is also known as **Reflected Binary Code**.

Gray code is the special case of unit-distance code. In unit distance code bit patterns for two consecutive numbers differ in only bit position. These codes are also called ***Cyclic Code.***

Table 2-9. Gray Code

Decimal	*Binary*	*Gray*
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

2-38. Binary-to-Gray Conversion

There are two methods of converting Gray code to Binary code. The rules for converting from any binary number to its equivalent Gray code number are as follows:

1. The MSB is the same in the Gray code as in the binary number.
2. The second MSB of the Gray code is obtained by adding the first MSB of binary number to the second MSB of the binary number. Ignore the carry if any.
3. The third MSB of the Gray code is obtained by adding the second MSB of binary number to the third MSB of the binary number. Ignore the carry if any.
4. Continue adding the bits as above until the LSB of Gray code is reached.
5. The Gray code number will always have the same number of bits as the binary number.

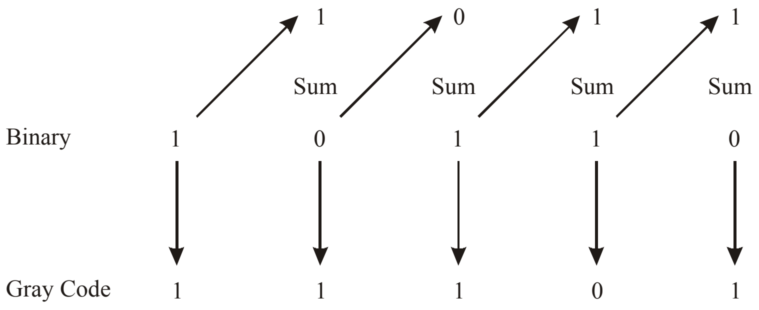


Fig. 2-5.

Converting binary 10110 to its cap Gray code equivalent is shown in Fig. 2-5. Start at the MSB of the binary number. The MSB of the Gray code is same as the MSB of binary number, it shown by the download arrow. Now add the first MSB bit to the next MSB bit of the binary number. This is written as the second MSB bit of the gray code. The procedure is continuing till the LSB bit. All the procedure is shown in Fig. 2-4 with arrow.

2-39. Gray-to Binary Conversion

The rules for converting from any binary number to its equivalent Gray code number are as follows:

1. The MSB of the binary number is the same as the MSB of the Gray code number.
2. The second MSB bit is obtained by adding the first MSB bit of binary number to the second MSB bit of the Gray code number. Ignore the carry if any.
3. The third MSB bit is obtained by adding the second MSB bit of binary number to the third MSB bit of the Gray code number. ignore the carry if any.
4. The process continues till we obtained the LSB of the binary number.

Conversion of the 6-bit Gray code number 011011 to its 6-bit binary equivalent is shown in Fig.2-6. Start at the MSB bit of the Gray code and follows the arrows. Follows the step given procedure mentioned above. Remember that 1 + 1 = 10, the carry of 1 is neglected the second MSB bit is obtained by adding the first MSB bit of the Gray code number. Ignore the carry if any.

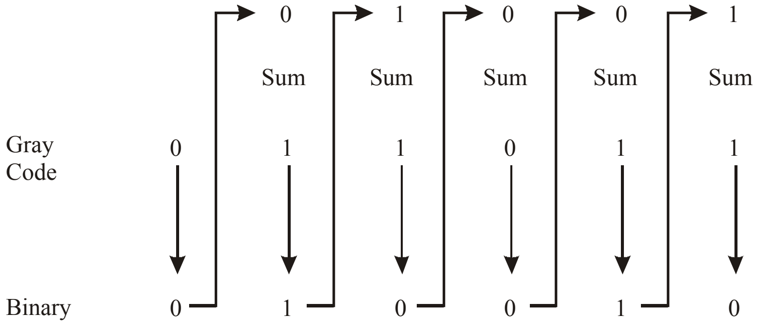
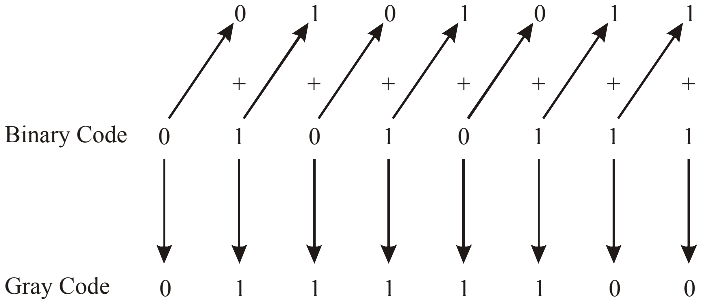


Fig. 2-6.

Example 2-68. *Convert the Gray coded number below into binary (01010111).*

(*Gauhati University, 2003*)

Solution. Given: The binary number 01010111



∴ 01010111 = 01111100 **Ans.**

Example 2-69. *Encode the decimal number 46 to Gray code.*

(Gujarat Technological University., Dec 2009)

Solution. Given: The decimal number 46

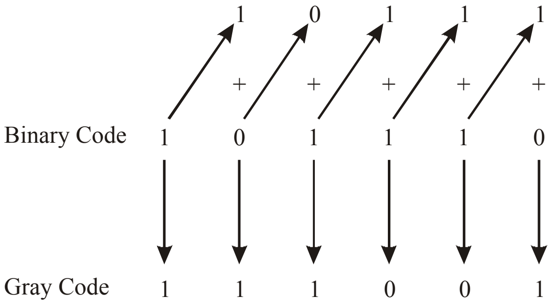
To convert the decimal number into gray code. First we have to convert it in binary form. We know that the decimal number can be expressed by the sum-of-weight as follows:

$$\begin{aligned} 46 &= 32+8+4+2 \\ &= 2^5+2^3+2^2+2^1 \end{aligned}$$

Placing 1_s in the appropriate weight positions 2^5, 2^3, 2^2, and 2^1 and 0_5 in the 2^4 and 2^0 position determines the binary number for decimal 46, thus

$$\begin{array}{cccccc} 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 1 & 1 & 0 \end{array}$$

$46_{10} = 101110_2$.



$\therefore \quad 46_{10} = 111001$ Gray Code **Ans.**

2-40. ASCII Code

We have already discussed in the last article that ASCII stands for American standard code for Information Interchange.

The ASCII code is used for the transfer of alphanumeric information between a computer and input/output devices such as monitors or printers. A computer also uses it internally to store the information that an operator types in at the computer's keyboard.

The ASCII code pronounced “askee is a seven-bit code, and so it has 2^7 (=128) possible code groups. This is more than enough to represent all of the standard keyboard characters as well as control functions such as the <RETURN> and <LINEFEED> functions. Table 2-9 shows a complete list of the ASCII code. Table 2-10 shows a partial list of ASCII code. You may find it more convenient to use.

Table 2-9. American Standard Code for Information Interchange

	MSB							
LSB	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P		p
0001	SOH	DC$_1$	!	1	A	Q	a	q
0010	STX	DC$_2$	”	2	B	R	b	r
0011	ETX	DC$_3$	#	3	C	S	c	s
0100	EOT	DC$_4$	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	↑	n	~
1111	SI	US	/	?	O	—	o	DEL

Definitions of control abbreviations:

ACK	Acknowledge	ESC	Escape	NUL	Null
BEL	Bell	ETB	End of transmission block	RS	Record separator
BS	Backspace	ETX	End text	SI	Shift in
CAN	Cancel	FF	Form feed	SO	Shift out
CR	Carriage return	FS	Form separator	SOH	Start of heading
DC$_1$-DC$_4$	Direct control	GS	Group separator	SP	Space
DEL	Delete idle	HT	Horizontal tab	STX	Start text
DLE	Data link escape	LF	Line feed	SUB	Substitute
EM	End of medium	NAK	Negative acknowledge	SYN	Synchronous idle
ENQ	Enquiry			US	Unit separator
EOT	End of transmission block			VT	Vertical tab

This table is arranged in such a way that the least significant bits (LSB) appear along a column and most-significant bits (MSB) along the horizontal row right at the top.

The least-significant bits (LSB) are a 4-bit group while most-significant bits (MSB) are a 3-bit group.

Let us see how to use this table to final the ASCII code for letter “**A**”. First locate the position of “**A**” in the ASCII table, identify the corresponding MSB-group and LSB-group bits. For the letter “**A**” we find that MSB are 100 and LSB are 0001. Therefore,

ASCII code for letter “**A**” = 100 0001

Similarly, the ASCII code for ‘**P**’ is 1110000 and for ‘**ESC**’ is 0011011.

Table. 2-10. Partial listing of ASCII code

Character	*Seven-Bit ASCII*	*Hex*	*Character*	*Seven-Bit ASCII*	*Hex*
A	100 0001	41	Y	101 1001	59
B	100 0010	42	Z	101 1010	5A
C	100 0011	43	0	011 0000	30
D	100 0100	44	1	011 0001	31
E	100 0101	45	2	011 0010	32
F	100 0110	46	3	011 0011	33
G	100 0111	47	4	011 0100	34
H	100 1000	48	5	011 0101	35
I	100 1001	49	6	011 0110	36
J	100 1010	4A	7	011 0111	37
K	100 1011	4B	8	011 1000	38
L	100 1100	4C	9	011 1001	39
M	100 1101	4D	blank	010 0000	20
N	100 1110	4E	.	010 1110	2E
O	100 1111	4F	(	010 1000	28
P	101 0000	50	+	010 1011	2B
Q	101 0001	51	$	010 0100	24
R	101 0010	52	*	010 1010	2A
S	101 0011	53	)	010 1001	29
T	101 0100	54	—	010 1101	2D
U	101 0101	55	/	010 1111	2F
V	101 0110	56	,	010 1100	2C
W	101 0111	57	=	011 1101	3D
X	101 1000	58	<RETURN>	000 1101	0D
			<LINEFEED>	000 1010	0A

Example 2-70. *The following is a message encoded in ASCII code. What is the message*?

1001000 1000101 1001100 1001100 1001111

Solution. Given : The message encoded in ASCII code.

1001000 1000101 1001100 1001100 1001111

First of all, let us convert seven-bit code to its equivalent hexadecimal number. The resulting number can be obtained as follows.

$$\underbrace{1\ 0\ 0}_{\substack{\downarrow\\ 4}}\ \underbrace{1\ 0\ 0\ 0}_{\substack{\downarrow\\ 8}}\ \underbrace{1\ 0\ 0}_{\substack{\downarrow\\ 4}}\ \underbrace{0\ 1\ 0\ 1}_{\substack{\downarrow\\ 5}}\ \underbrace{1\ 0\ 0}_{\substack{\downarrow\\ 4}}\ \underbrace{1\ 1\ 0\ 0}_{\substack{\downarrow\\ C}}\ \underbrace{1\ 0\ 0}_{\substack{\downarrow\\ 4}}\ \underbrace{1\ 1\ 0\ 0}_{\substack{\downarrow\\ C}}\ \underbrace{1\ 0\ 0}_{\substack{\downarrow\\ 4}}\ \underbrace{1\ 1\ 1\ 1}_{\substack{\downarrow\\ F}}$$

Thus the resulting values are 4 8, 4 5, 4 C, 4 C and 4 F.

Now locate these hexadecimal values in Table 2-9 and determine the character represented by each. The results are

"H E L L O"

The ASCII code is used extensively to exchange information between the computer and its peripheral devices.

Example 2-71. *Represent the decimal number (a) 27 (b) 396 (c) 4096 in (i) binary form (ii) In ASCII Code (iii) Gray code (iv) Excess 3 Codes.*

(*PTU., May 2008 Dec 2008*)

Solution

(*a*) *Given: The decimal number 27*

(*i*) *Binary form*

We know that the conversion from decimal-to-binary can be done by using repeated-division by 2 method.

		Top	
$27 \div 2 = 13$	with a remander	1 ↑	LSB
$13 \div 2 = 6$	with a remander	1	
$6 \div 2 = 3$	with a remander	0	
$3 \div 2 = 1$	with a remander	1	
$1 \div 2 = 0$	with a remander	1	LSB
		Bottom	

Reading the remander from bottom to top, the binary number for the decimal number 27 is 11011_2

$\therefore \quad 27_{10} = 11011_2$ **Ans.**

(*ii*) *ASCII Code*

Using the 2-11, we know that the conversion of decimal to ASCII can be done, as follows:

Step 1 2 7

↓ ↓

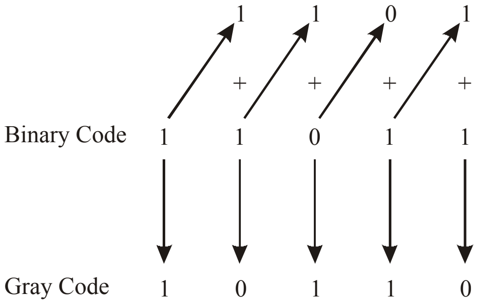
Step 2 0 1 1 0 0 1 0 0 1 1 0 1 1 1

Step 3 0 1 1 0 0 1 0 0 1 1 0 1 1 1

$\therefore \quad 27_{10} = 01100100110111.$

(*iii*) *Gray Code.*

We know that the conversion from decimal to gray code is done as, first change the decimal number into binary code. Then the conversion process start,



$\therefore \quad 27_{10} = 10110$ **Ans.**

(*iv*) *Excess: 3code*

Excess 3 code is obtained by adding 11 to its binary code

$$\begin{array}{ccccc} 1 & 1 & 0 & 1 & 1 \\ + & & & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array} \quad \textbf{Ans.}$$

(*b*) *Given the decimal number 396*

(*i*) *Binary form*

We know that the conversion from decimal-to-binary can be done by using repeated – division by 2 method.

				Top
396 ÷ 2	= 198	with a remainder	0	(LSB)
198 ÷ 2	= 99	with a remainder	0	
99 ÷ 2	= 49	with a remainder	1	
49 ÷ 2	= 294	with a remainder	1	
24 ÷ 2	= 12	with a remainder	0	
12 ÷ 26	= 6	with a remainder	0	
6 ÷ 2	= 3	with a remainder	0	
3 ÷ 2	= 1	with a remainder	1	
1 ÷ 2	= 0	with a remainder	1	↓ (MSB)
				Bottom

Reading the remained from bottom to top, the binary number for the decimal number 396 is 11000100

∴ 396_{10} = 110001100 **Ans**.

(*ii*) *ASCII code*

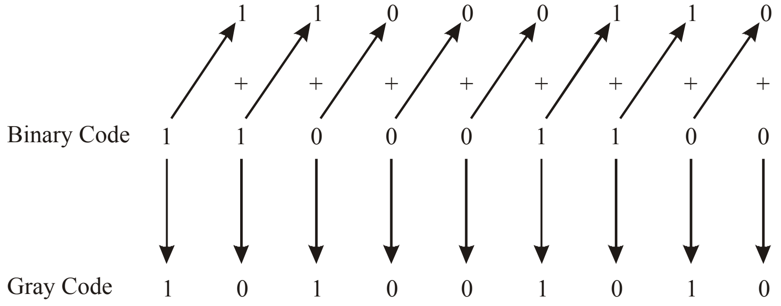
Using the table 2-11 we know that the conversion of decimal-to-ASCII can be done as follows:

Step 1.	3	9	6
	↓	↓	↓
Step 2.	0 1 1 0 0 1 1	0 1 1 1 0 0 1	0 1 1 0 1 1 0
Step 3.	0 1 1 0 0 1 1	0 1 1 1 0 0 1	0 1 1 0 1 1 0

∴ 396_{10} = 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 0 1 1 0 **Ans.**

(*iii*) *Gray Code*

We know that the conversion from decimal-to-gray code is done as.



$\therefore$ $396_{10} = 101001010$. **Ans.**

(*iv*) Excess 3 code

Excess 3 code is obtained by adding 11 to its binary code.

$$\begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ + & & & & & & & 1 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array} \quad \textbf{Ans.}$$

(*c*) Given: the decimal number 4096

(*i*) *Binary form*

We know that conversion from decimal-to-binary can be done by using repeated-division by 2 method.

				Top
4096 ÷ 2	= 2048	with a remainder	0	(LSB)
2048 ÷ 2	= 1024	with a remainder	0	
1024 ÷ 2	= 512	with a remainder	0	
512 ÷ 2	= 256	with a remainder	0	
123 ÷ 2	= 128	with a remainder	0	
128 ÷ 2	= 64	with a remainder	0	
64 ÷ 2	= 32	with a remainder	0	
32 ÷ 2	= 16	with a remainder	0	
16 ÷ 2	= 8	with a remainder	0	
8 ÷ 2	= 4	with a remainder	0	
4 ÷ 2	= 2	with a remainder	0	
2 ÷ 2	= 1	with a remainder	0	
1 ÷ 2	= 0	with a remainder	1	(MSB)
				Top

Reading the remainder from bottom to top, the binary numbers for decimal number 4096 is 1000000000000

$\therefore$ $4096_{10} = 1000000000000$ **Ans.**

(*ii*) ASII Code

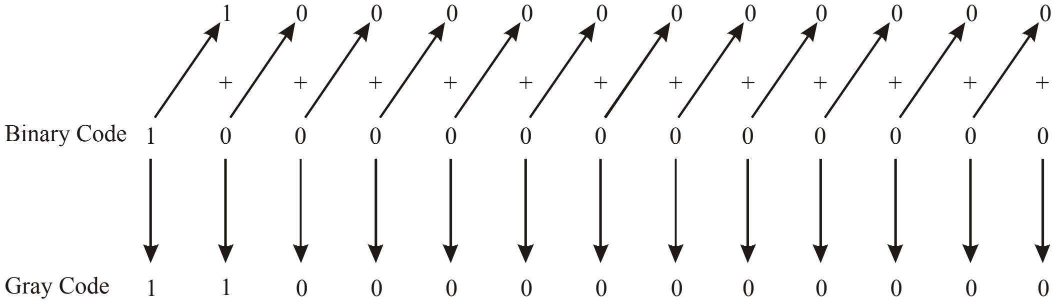
Using the table 2-11 we know that the conversion of decimal-to-ASCII can be done as follows:

Step 1.	4	0	9	6
	↓	↓	↓	↓
Step 2.	0110100	0110000	0111001	0110110
Step 3.	0110100	011000	00111001	0110110

∴ 4096_{10} = 01101000110000 0111001 011 0110 **Ans.**

(*iii*) *Gray Code*

We know that the conversion from decimal to gray code is done as, first change the decimal number into binary code, then the conversion process start,



		1	0	0	0	0	0	0	0	0	0	0	0
		+	+	+	+	+	+	+	+	+	+	+	+
Binary Code	1	0	0	0	0	0	0	0	0	0	0	0	0
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Gray Code	1	1	0	0	0	0	0	0	0	0	0	0	0

∴ 4096_{10} = 1100000000000 **Ans.**

(*iv*) Excess-3 code

Excess-3 code is obtained by adding 11 to its binary code

1	0	0	0	0	0	0	0	0	0	0
+									1	1
1	0	0	0	0	0	0	0	0	1	1

2-41. Error Detection and Correction

As a matter of fact, the movement of binary data and codes from one location to another is the most frequent operation performed in digital electronic systems. For example:

1. The transmission of digitized voice over a mobile phone
2. The storage of data in and retrieval of data from external memory devices such as magnetic tape and disk
3. The transmission of information from a computer to a remote user terminal or another computer over telephone lines

Whenever information is transmitted from one electronic system (say transmitter) to another system (say receiver), there is a possibility that errors can occur such that the receiver does not receive the identical information that was sent by the transmitter. The major reason for any transmission errors is *electrical noise*. The noise consists of spurious fluctuations in voltage or current that are present in all electronic systems to varying degrees. Fig. 2-7 is a simple illustration of a type of transmission error.

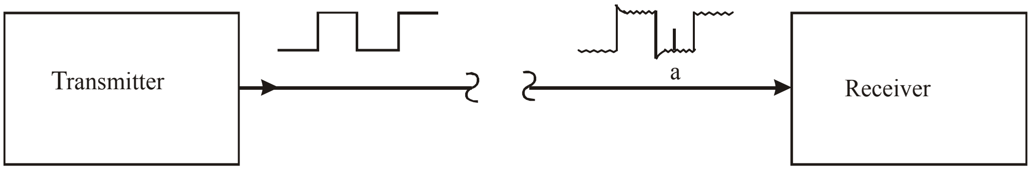


Fig. 2-7. *Illustration of a producing noise an error in the transmission of digital data.*

The transmitter sends a relatively noise-free serial digital signal over a signal line to a receiver. However, by the time the signal reaches the receiver, it contains a certain degree of noise superimposed on the original signal. Occasionally, the noise is large enough in amplitude that it will alter the logic level of the signal as it does at point 'a'. When this occurs, the receiver may incorrectly interpret that bit as a logic 1, which is not what the transmitter had sent.

Most modern digital electronic equipment is designed to be relatively error-free, and the probability of errors such as shown in Fig. 2-5 is very low. However, we must realize that digital electronic systems often transmit millions, of bits per second, so that even a very low rate of occurrence of errors can produce an occasional error that might prove to be bothersome, if not disastrous. Due to this reason, many digital electronic systems employ some method for detection (and correction) of errors. One of the simplest and most widely used methods for error detection is the *Parity Method* or the Parity code. A code used for error detection and correction is called 'Hamming Code'.

Before we understand the parity method, it is essential to study about the parity bit.

2-42. Parity Bit

Strictly speaking, a *Parity bit* is an extra bit that is attached to a code group that is being transmitted from one location to another. The parity bit is made either 0 or 1, depending on the number of 1s that are contained in the code group. Two different methods are used for attaching an extra bit. These are discussed below.

The Even Parity Method

In this method, the value of the parity bit is chosen so that the total number of 1s in the code group (including the parity bit) is an *even* number. For example, suppose that the code group is 1000101. This is the ASCII character "E". The code group has *three* 1s. Therefore, we will add a parity bit of 1 to make the total number of 1s an even number. The *new* code group, including the parity bit, thus becomes 11000101.

The Odd Parity Method

In this method, the value of the parity bit is chosen so that the total number of 1s in the code group (including the parity bit) is an *odd number*. For example, suppose the code group is 1000101. The code group has three 1s. Therefore we will add a parity bit of o to keep the total number of 1s an odd number. The new code group including the party bit becomes 01000101.

Example 2-72. *What parity do the following words have*

(*i*) *11001101*

(*ii*) *10110100* (*Assam Engineering College,2006*)

Solution.

(*i*) Given: the word 11001101

From the given word group we find that it has odd number of 1_s. So this word is odd parily.

(*ii*) Given: the word 10110100

From the given word group we find that it has even number of 1_s. So this word is even parity.

Example 2-73. *Assign the proper even parity bit to the following code groups*:

(*a*) *1010* (*b*) *111000* (*c*) *101101*

Solution.

(*a*) Given : The code group 1010.

From the given code group we find that it has even number of 1s. Therefore we attach a 0 as the parity bit at most - significant bit position.

$$\underset{\text{Parity bit}\ \uparrow}{0}\qquad\underbrace{1010}_{\text{Code group}}$$

(*b*) Given : The code group 111000

For a given code group we find that it has odd number of 1s. Therefore we attach a 1 as the parity bit at the most-significant bit position, i.e.,

$$\underset{\text{Parity bit}\ \uparrow}{1}\qquad\underbrace{111000}_{\text{Code group}}$$

(*c*) Given the code group 101101

For a given code group we find that it has even number of 1s. Therefore we attach a 0 as the parity bit at the most-significant bit position.

$$\underset{\text{Parity bit}\ \uparrow}{0}\qquad\underbrace{101101}_{\text{Code group}}$$

Example 2-74. *Computers often communicate with other remote computers over the telephone lines-this is how the communication takes place over the internet. When one computer is transmitting a message to another, the information is usually encoded in ASCII. What actual bit strings would a computer transmit to send the message TECHNOLOGY using ASCII with even parity*.

Solution. Given, the message TECHNOLOGY.

In order to find the bit strings, we will first look up the ASCII codes for each character in the message by using the Table 2-11. Then for each code, we count the number of 1s. If it is an even number, attach a 0 as a parity bit at the most-significant bit position. If it is an odd number, attach a 1. Thus the resulting 8-bit codes will all having an even number of 1s (including parity) are:

Table. 2-11.

Character	*Parity bit*	*Code group*
T–	1	1010100
E–	1	1000101
C–	1	1000011
H–	0	1001000
N–	0	1001110
O–	1	1001111
L–	1	1001100
O–	0	1001110
G–	0	1000111
Y–	0	1011001

Example 2-75. *The following is a message encoded in ASCII code.*

1001000 1000101 1001100 1001100 1001111

What is the message?

Solution: Given the message encoded in the ASCII code:

1001000 1000101 1001100 1001100 1001111

We know that to read the message we have to convert the given data string into the corresponding characters by using the ASCII code Table 2-9. From this table we find that,

1001000	Corresponds to	'H'
1000101	Corresponds to	'E'
1001100	Corresponds to	'L'
1001100	Corresponds to	'L'
1001111	Corresponds to	'O'

Therefore the given data string represents the message "HELLO"

2-43. Hamming Code

Hamming code is a linear error-correcting code named after its inventor, Richard Hamming. Hamming codes can detect up to two simultaneous bit errors, and correct single-bit errors.

When data is transmitted from one location to another there is always the possibility that an error may occur. There are a number of reliable codes that can be use to encode data so that the error can be detected and corrected. With this project you will explore a simple error detection-correction technique called a Hamming Code. A Hamming Code can be used to detect and correct one-bit change in an encoded code word. This approach can be useful as a change in a single bit is more probable than a change in two bits or more bits.

Hamming code makes use of the concept of parity and parity bits, which are bits that are added to data so that the validity of the data can be checked when it is read or after it has been received in a data transmission. Using more than one parity bit, an error-correction code can, not only identify a single bit error in the data unit, but also its location in the data unit.

In the Hamming code, k parity bits are assed to an n-bit data word forming a new word of n + k bits. The bit positions are numbered in sequence form 1 to n + k. Those positions numbered as a power of 2 are reserved for the parity bits. The remaining bits are the data bits. The code can be used with word of any length.

For example of hamming code consider the table below which has 15 positions. Data is represented (stored) in every position except 1,2.4 and 8.These positions (which are powers of 2) are used to store parity (error correction) bits.

Table 2-12.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		1		2	3	4		5	6	7	8	9	10	11

Using the four parity (error correction bits) positions we can represent 15 values (1-15). These values and their corresponding binary representation are shown in the Table. 2-13 below:

Table 2-13

Pos	$2^0=1$	$2^1=2$	$2^2=4$	$2^3=8$
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

Consider, for example the 10101101011 data. We include 4 parity bits with the data word and arrange the 15 bits as follows.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P_1	P_2	1	P_4	0	1	0	P_8	1	1	0	1	0	1	1

The 4 parity P_1, P_2, P_4 and P_8 are in positions 1,2,4 and 8 respectively. The 11bits of the data word are in the remaining positions.

After placing the data in the table we find that in positions 3,6,9,10,12,14 and 15 we have a '1'. Using our previous conversion table we obtain the binary representation for each of these values. We then exclusive OR the resulting values. Remember that the exclusive-OR operation performs the odd function. It is equal to 1 for an odd number of 1's in the variable and to 0 for an even number of1's.

	1	1	0	0	3
	0	1	1	0	6
	1	0	0	1	9
	0	1	0	1	10
	0	0	1	1	12
	0	1	1	1	14
	1	1	1	1	15
XOR	1	1	0	1	(11)

The parity bits are then put in the proper locations in the Table

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	1	0	1	1	1	0	1	0	1	1

This is the encoded code word that would be sent. The receiving side would re-compute the parity bits and compare them to the ones received. If they were the same no error occurred – if they were different the location of the flipped bit is determined.

For example let's say that the bit position 14 was flipped during transmission. The receiving end would see the following encoded sequence:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	0	1	0	1	1	1	0	1	0	0	1

Below is the re-calculation of the receiving end data is done. Notice the information for position 14 has been left out as it was flipped from 1 to 0.

	1	1	0	0	3
	0	1	1	0	6
	1	0	0	1	9
	0	1	0	1	10
	0	0	1	1	12
	1	1	1	1	15
XOR	1	0	1	0	

The calculated parity information is then compared to the parity information sent and received. If they are both the same, the result (again using an XOR – even parity) will be all 0's. If a single bit was flipped the resulting number will be the position of the error bit. For example

Sent Bit	1	1	0	1
Received bit when bit 14 is left out	1	0	1	0
XOR	0	1	1	1

The XOR result of the sent bit and the received bit when the error is generated is 0111. This bit was position number 14.

2-44. Applications of the Number Systems

We have already discussed a variety of number systems in this chapter. Which number system is used depends on how the data were developed and how they are to be used. Now we will go through several applications that depend on the translation and interpretation of these digital representations.

Example 2-76. *Typically digital thermometers use BCD to drive their digital displays. How many BCD bits are required to drive a* 3-*digit thermometer display*? (b) *What* 12-*bits are sent to display for a temperature of* 157 ºC?

Solution.

(*a*) *Number of BCD bits required to drive a 3-digit thermometer display*

Given, a digital thermometer using BCD for display

We know that each BCD bit requires a 4-bit code, therefore 3-digit disply will have 3 × 4 = 12-bits. **Ans.**

(*b*) *12-bit BCD code sent for display*

Given: The temperature =157 $^{\circ}$C

We know that to find the 12-bit BCD code representing a temperature of 157 $^{\circ}$C, we need to convert each digit to its 4-bit BCD code. Thus,

$$\begin{array}{ccc} 1 & 5 & 7 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{0001} & \overbrace{0101} & \overbrace{0111}. \end{array}$$

$\therefore$ 157 °C = 0001 0101 0111 $_{\text{BCD}}$ **Ans.**

Example 2-77. *A particular brand of compact disk* (*CD*) *player has the capability of converting* 12-*bit signals from a CD into their equivalent analog values.* (*a*) *What are the largest and smallest hex values that can be used in the CD system.* (*b*) *How many different analog values can be represented by the system.*

Solution.

(*a*) *Number of analog values*

We know that the CD player has a capability of converting 12-bit signals from a CD into their equivalent analog values, and

Number of analog values in decimal = 2^{12} = 4096 **Ans.**

(*b*) *Largest and smallest hex values*

We know that the largest decimal value is, $2^{12}-1$ = 4095 and the smallest decimal value is 0.

The largest value in hexadecimal number system can be obtained by using the repeated division by 16 as given below:

$$\begin{array}{llllllll} & & & & & & & \text{Top} \\ 4095 \div 16 = 255.9375 & = 255 & \text{with a remainder} & 0.9375 & \rightarrow 0.9375 \times 16 & = 15 \rightarrow \text{F} & \uparrow & \text{LSD} \\ 255 \div 16 = 15.9375 & = 15 & \text{with a remainder} & 0.9375 & \rightarrow 0.9375 \times 16 & =15 \rightarrow \text{F} & | & \\ 15 \div 16 = 0.9375 & = 0 & \text{with a remainder} & 0.9375 & \rightarrow 0.9375 \times 16 & =15 \rightarrow \text{F} & | & \text{MSD} \\ & & & & & & & \text{Bottom} \end{array}$$

Since the quotient is zero, therefore we stop the repeated-division by16. Reading the remainders from bottom-to-top, we find that the required hexadecimal number is FFF.

$$\therefore \qquad \left.\begin{array}{l} \text{Largest value in hex} = \text{FFF}_{16} \\ \text{Smallest value in hex} = 0_{16} \end{array}\right\} \quad \textbf{Ans.}$$

Example 2-78. *Most PC–compatible computer systems use a* 20–*bit address code to identify each of over* 1 *million memory locations.* (*a*) *How many hex characters are required to identify the address of each memory location?* (*b*) *What is the* 5–*digit hex address of the* 500th *memory location?* (*c*) *If* 60 *memory locations are used for data storage starting at location 000FFH, what is the location of the last data item*?

Solution.

(*a*) *Number of hexadecimal characters*

We know that each hexadecimal digit represents 4 bits. Therefore 20-bit requires five hexadecimal digits. **Ans.**

(*b*) *5-digit hexadecimal address of 500th memory location*

We know that the decimal number 500 is to be converted to its equivalent hexadecimal number by using repeated division-by -16 method, i.e.,

Top

$500 \div 16 = 31$ with a remainder 4 ↑ LSD

$31 \div 16 = 1$ with a remainder 15 (=F)

$15 \div 16 = 0$ with a remainder 15 (=F) MSD

Bottom

$\therefore \quad 500_{10} = FF4_{16}$.

Now we know that the first memory laocation is 00000 and the 500th location is 00FF3 (i.e 00FF4 minus 1).

∴ The 5- digit hexadecimal address of the 500th memory location is 00FF3 **Ans.**

(*c*) *Location of the last data item*

We know that the data is stored from the memory location 00FF4 onwords. If 60 locations are to be filled up, then, 00FF4 + 60 = 01054H. But we have to subtract 1 from the value 01054H because the memory location 00FF4H received the first data item, so we needed 59 more memory spaces

∴ The location of the last data item = (01054 –1)H = 01053H **Ans.**

Example 2-79. *A chemical processing plant uses a computer to monitor the temperature and pressure of four chemical tanks as shown in Fig* 2-8. *Whenever a temperature or a pressure exceeds the danger limit, an internal tank sensor applied a* "1" *to its corresponding output to the computer. If all conditions are OK, then all outputs are zero.* (*a*) *If the computer reads the binary string* 0010 1000, *what problems exist.* (*b*) *What problems exist if the computer is reading 55 H*? (*c*) W*hat hexadecimal number is read by the computer if the temperature and pressure in both tanks B and D are high*? (*d*) *Tanks A and B are taken out of their use and their sensor outputs are connected to* 1's. *A computer programmer must write a program to ignore these new circuit conditions. The computer program must check that the value read is always less than what decimal equivalent when no problem exists.*

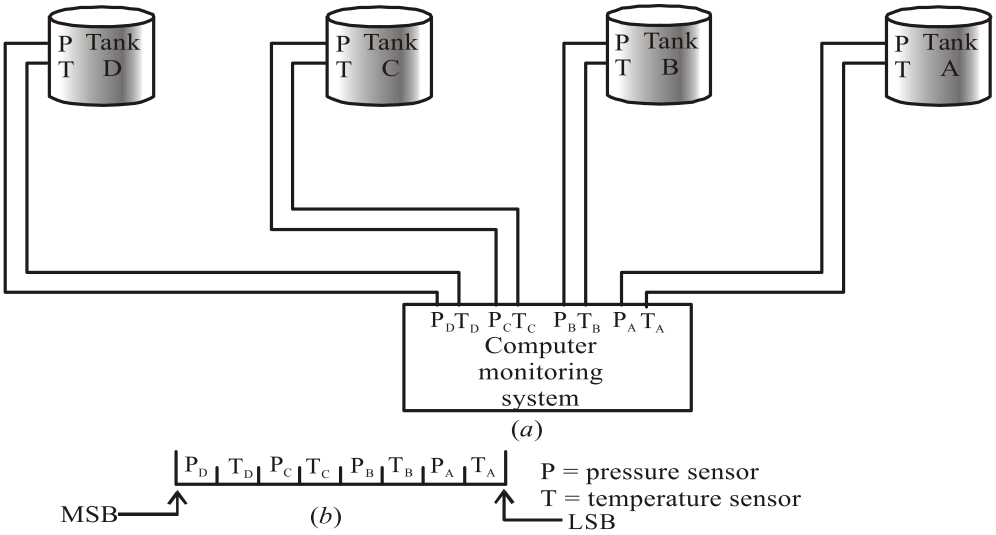


Fig. 2-8.

Solution. Given: A chemical processing plant using a computer to monitor the temperature and pressure of four chemical tanks.

(*a*) *Problems that exist if computer reads the binary string 00101000.*

We know that the computer reads the data in the format shown below

P_D	T_D	P_C	T_C	P_B	T_B	P_A	T_A

MSB LSB

By comparing the corresponding values of pressure and temperature with data input, we find P_B and P_C are 1. This implies that the pressure in tanks B and C are dangerously high. **Ans.**

0	0	1	0	1	0	0	0
P_D	T_D	P_C	T_C	P_B	T_B	P_A	T_A

(*b*) *Problems that exist if the computer is reading 55H.*

We know that the computer reads the data 55 H which is read in the format shown below :

0	1	0	1	0	1	0	1
P_D	T_D	P_C	T_C	P_B	T_B	P_A	T_A

By comparing the corresponding values of pressure and temperature with the data input, we find that the temperature of all the four tanks A,B,C and D are dangerously high. **Ans.**

(*c*) *Hexadecimal number read by the computer if the temperature and pressure in both tanks B and D are high.*

We know that the computer must read 1 for the temperature and pressure bits for both tanks B and D.

The data read by the computer should be as shown below :

1	1	0	0	1	1	0	0
P_D	T_D	P_C	T_C	P_B	T_B	P_A	T_A

From the above diagram we find that the binary value is 11001100. The hexadecimal equivalent of this binary number is CC_{16} or CC H **Ans.**

(*d*) *Decimal number read by the computer with A and B out of use.*

We know that the tanks A and B are taken out of their use and their outputs are connected to 1's. This means P_A, T_A, P_B and T_B all are '1'. Now if T_C goes high, then the number read by the computer is as indicated below :

0	0	0	1	1	1	1	1
P_D	T_D	P_C	T_C	P_B	T_B	P_A	T_A

From the above diagram, we find that the binary number read by the computer is 00011111. This is equivalent to decimal number 31_{10}. Thus so long the computer reads below 31_{10}, there is no problem with the pressure and temperature of the tanks C and D. **Ans.**

SUMMARY

In this chapter, we have learnt that:

1. Using an n-bit number, we can represent decimal values ranging from 0 to $2^n – 1$.
2. The octal number system consists of eight digits, 0 through 7.
3. A decimal number can be converted to octal by using the repeated division-by-8 method.

4. Octal-to-binary conversion is accomplished by simply replacing each octal digit with its 3-bit binary equivalent. The process is reversed for binary-to-octal conversion.
5. The hexadecimal number system consists of 16 digits and characters, 0 through 9 and A through F.
6. One hexadecimal digit represents a four-bit binary number and its primary usefulness is in simplifying bit patterns and making them easier to read.
7. A decimal number can be converted to hexadecimal by the repeated division-by-16 method.
8. A decimal number is converted to Binary-Coded-Decimal (BCD) by replacing each decimal digit with the appropriate 4-bit binary code.
9. The ASCII code is a 7-bit alphanumeric code that is widely used in computer systems for input and output information.
10. The parity method for error detection attaches a special parity bit to each transmitted group of bits.

GLOSSARY

Octal number system. A number system that has a base of 8, and uses digits from 0 to 7 to express an octal number.

Hexadecimal number system. A number system that has a base of 16, and uses digits from 0 through 9 plus letters A through F to express a hexadecimal number.

Straight binary coding. Representation of a decimal number by its equivalent binary number.

Binary-coded-decimal (BCD) code. Four bit code used to represent each digit of a decimal number by its four-bit binary equivalent.

Byte. Group of eight bits.

Alphanumeric Codes. Codes that represent numbers, letters, punctuation marks and special characters.

ASCII (American Standard Code for Information Interchange). Seven-bit alphanumeric code used by most computer manufacturers.

Parity method. Scheme used for error detection during the transmission of data.

Parity bit. Additional bit that is attached to each code group so that the total number of 1s being transmitted is always odd (or always even).

DESCRIPTIVE QUESTIONS

1. Why is the binary number system commonly used in digital electronics?
2. How are the position values (or weighting factors) determined for each binary position in a base-2 (i.e. binary) number system?
3. Why is hexadecimal number system used instead of the octal number system when working with 8- and 16- bit digital computers?
4. How does BCD differ from the straight binary number system?
5. Derive the truth table of a binary-to-Gray code converter for number between 0 and 7. Draw the converter.

   (*Gauhati University, 2007*)
6. What is the difference between weighted and non-weighted codes? Give one example for each.

   (*Nagpur University, 2008*)

7. What is a self complementing code. (*Nagpur University, 2008*)

8. Explain with examples self complementing codes. (*Mahatma Gandhi University, Jan.2007*)

9. What are alphanumeric codes? Explain giving examples. (*Mahatma Gandhi University, Dec.2007*)

10. What are alphanumeric codes? Where they are used? (*Mahatma Gandhi University, Dec. 2007*)

11. What are weighted and unweighted codes? Explain with examples. (*Mahatma Gandhi University, Nov.2005*)

12. List the advantages of octal over hexadecimal number system. (*PTU, Dec 2008*)

13. What do you mean by weighted code? Give example. (*PTU, May 2009*)

14. What is the largest decimal number that can be represented as a 16 bit binary word?

15. What is the number of bits in ASCII code? What is the need for ASCII code? (*Anna University, Nov./Dec. 2005*)

16. Write a note on ASCII code. (*Anna University, Nov./Dec. 2005*)

17. Write a note on ASCII code. (*Anna University, Nov./Dec. 2005*)

18. Give an example each of one-byte, two-type and three-byte instruction. (*Gauhati University, 2007*)

19. Define bit, byte and nibble. (*PTU, May 2009*)

20. How error correction is does using parity codes? (*PTU, May 2009*)

21. What is cyclic code? Write the principle of detection and correction of one bit error with the help of it. (*GBTU/MTU, 2005-06*)

22. What do you mean by cyclic code? Explain giving a example, the application of cyclic code. (*GBTU/MTU, 2006-07*)

23. Write notes on Hamming Code. (*GBTU/MTU,2004-05, 2006-07*)

24. Write a note on error detecting and correcting codes. (*GBTU/MTU, 2006-07*)

25. Why ASCII code required by digital computer systems?

26. Why can't the parity method detect a double error in transmitted data?

TUTORIAL PROBLEMS

1. What is the largest decimal value that can be represented by (*a*) a 10-bit (*b*) a 20-bit number.

(**Ans.** (*a*) 1048, (*b*) 1,098,304)

2. Convert the following binary numbers to their decimal equivalents.

(*a*) 11011 (*b*) 10110101

(*c*) 10111111 (*d*) 10001101 (**Ans.** 27, 181,191,141)

3. Convert the following decimal numbers to their binary equivalent.

(*a*) 45 (*b*) 76

(*c*) 177 (*d*) 975

(**Ans.** (*a*) 101101, (*b*) 1001100, (*c*) 10110001, (*d*) 1111001111)

4. Convert the following octal numbers to their decimal equivalent.

   (*a*) 24.6 (*b*) 372

   (*c*) 547 (*d*) 2374

   (**Ans.** (*a*) 20.75, (*b*) 250, (*c*) 359, (*d*) 1276)

5. Convert each of the following decimal numbers to their octal equivalent.

   (*a*) 266 (*b*) 255 (*c*) 372 (*d*) 919

   (**Ans.** (*a*) 412, (*b*) 377, (*c*) 564, (*d*) 1627)

6. Convert each of the following octal values to binary.

   (*a*) 13 (*b*) 25 (*c*) 140 (*d*) 7526

   (**Ans.** (*a*) 001011, (*b*) 010101, (*c*) 001100000, (*d*) 111101010110)

7. Convert each of the following binary numbers to their octal equivalent.

   (*a*) 110101 (*b*) 101111001 (*c*) 100110011010 (*d*) 11010000100

   (**Ans.** (a) 65, (b) 571, (c) 4632, (d) 3204)

8. When a large decimal number is to be converted to a binary, it is sometimes easier to convert it first to octal and then from octal to binary. Try this method for 2048_{10}.

   (**Ans.** 4000_8 100 000 000 000)

9. Convert each of the following hexadecimal values to their decimal equivalent.

   (*a*) E5 (*b*) 2AF (*c*) 1BC2 (*d*) B2F8

   (**Ans.** (*a*) 229, (*b*) 687, (*c*) 7106, (*d*) 45,816)

10. Convert each of the following decimal values to their hexadecimal equivalent values.

    (*a*) 214 (*b*) 423 (*c*) 650 (*d*) 2591

    (**Ans.** (*a*) D6, (*b*) 1A7, (*c*) 28A, (*d*) A1F)

11. List the hexadecimal numbers in sequence from 380 to 3A0.

12. How many hexadecimal digits are required to represent decimal numbers upto 1,098,304.

    (**Ans.** 4 hex digits)

13. Encode each of the following decimal numbers to their BCD equivalent.

    (*a*) 479 (*b*) 1204

    (**Ans.** (*a*) 0100 0111 1001, (*b*) 0001 0010 0000 1000)

14. Given the 8-bit data word 01011011, generate the 13 bit composite word for the Hamming code that corrects single errors and detects double errors. (*GBTU/MTU, 2009-10*)

MULTIPLE CHOICE QUESTIONS

1. What range of decimal values can be represented by a eight-bit binary number?

   (*a*) 0 to 64 (*b*) 0 to 128

   (*c*) 0 to 256 (*d*) 0 to 512

2. What is the next number in the octal counting sequence: 824, 825, 826, 827,

   (*a*) 828 (*b*) 82A (*c*) 820 (*d*) 830

3. What range of decimal values can be represented by a four-digit hexadecimal number?

(*a*) 0 to 1024 (*b*) 0 to 4096 (*c*) 0 to 8192 (*d*) 0 to 65535

4. What is next four numbers in the hexadecimal counting sequence D9A, D9B, D9C, D9D,......

(*a*) D9E, D9F, D 90, D 91 (*b*) D 9E, D9F, DA0, DAI

(*c*) D9E, E10, E11, E12 (*d*) D 9E, D9F, D10, D 11.

5. The number of bits required to represent an eight-digit decimal number in BCD is

(*a*) 8 (*b*) 16 (*c*) 24 (*d*) 32

ANSWERS

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

3

LOGIC GATES

OUTLINE

Objectives

Upon completion of this unit, you should be able to,

- Describe the operation and construct truth tables of an INVERTER, the AND gate and the OR gate.
- Describe the operation and construct truth tables of the NAND gate, the NOR gate, the exclusive –OR gate and the exclusive –NOR gate.
- Express the logical operation of INVERTER, AND, OR, NAND, NOR, XOR and XNOR gates in terms of a Boolean equation
- Construct timing diagrams showing the proper time relationships of inputs and outputs for the various logic gates.
- Use the logic gates to implement a circuit represented by a Boolean equation.
- List specific devices that contain the various logic gates.
- Use each logic gate in simple applications.

3-1. Introduction

Logic gates are the basic building blocks for forming digital electronic circuitry. A logic gate has one output terminal and one or more input terminals. Its output would be HIGH (1) or Low (0) depending on the digital level (s) at the input terminal (s). The logic gates can be used to design digital systems that will evaluate digital input levels and produce a specific output response based on that particular logic circuit design.

We will study the following logic gates in this chapter.

(1) INVERTER (or NOT gate) (2) AND gate (3) OR gate (4) NAND gate
(5) NOR gate (6) Exclusive-OR gate (7) Exclusive- NOR gate.

The INVERTER, AND and OR gates are considered to be the **basic** logic gates in the field of digital electronics. The NAND, NOR, Exclusive-OR and Exclusive- NOR gates are constructed using the basic logic gates.

Now before we start our studies on logic gates, let us know about Boolean constants, variables, truth tables and Boolean expressions. These terms are very important to understand the operation of Logic gates.

3-2. Boolean Constants and Variables

Boolean algebra differs in a major way from an ordinary algebra in that Boolean constants and variables are allowed to have only two possible values, 0 or 1. A Boolean variable is a quantity that may, at different times, be equal to either 0 or 1. Boolean variables are often used to represent the voltage level present on a wire or at the input/output terminals of a circuit. For example, in a certain digital system the Boolean value of 0 might be assigned to any voltage in the range from 0 to 0.8 V while the Boolean value of 1 might be assigned to any voltage in the range 2 to 5 V.

Thus Boolean 0 and 1 do not present actual numbers but instead represent the state of voltage, or what is called its LOGIC LEVEL. A voltage in a digital circuit is said to be at the logic level 0 or the logic level 1. Several other items are used synonymously with 0 and 1. Some of the more common ones are shown in Table- 3-1. We will use the 0/1 and LOW/HIGH designations most of the time.

As discussed earlier in Chapter 1, Boolean algebra is a means for expressing the relationship between the logic circuit's inputs and outputs. The inputs are considered logic variables whose logic levels at any time determine the output levels. In all our discussion to follow, we shall use letter symbols to represent logic variables. For example “A” might represent a certain digital circuit input, and at any time we must have either “A” = 0 or “A” = 1

Table 3-1. Boolean constants and Variables

Logic 0	*Logic 1*
False	True
Off	On
Low	High
No	Yes
Open-switch	Closed-switch

Boolean algebra is relatively easy to work with as compared to an ordinary algebra because of only two possible values. It may be noted that in Boolean algebra there are no fractions, decimals, negative numbers, square roots, cube roots, logarithms, imaginary numbers and so on. As a matter of fact, in Boolean algebra there are only three basic operations, NOT, AND and OR.

These basic operations are called logic operations. Digital circuits called logic gates can be constructed from diodes, transistors, and resistors connected in such a way that the circuit output is the result of a basic logic operation (OR, AND and NOT) performed on the inputs. We will use Boolean algebra in this chapter to describe the logical operation of basic logic gates. In Chapter 4 we will use Boolean algebra to analyse and design logic circuits using combinations of basic logic gates.

3-3. Truth Tables

A truth table is a means for describing how a logic circuit’s output depends on the logic levels present at the circuit’s inputs. Table 3-2 illustrates a truth table for one type of two-input logic circuit. The table lists all possible combinations of logic levels present at inputs (A and B) along with the corresponding output level (X). The first row in the table shows that when A and B are both at the 0 level, the output X is at the level 1 or, equivalently, in the 1 state . The second row shows that when input B is changed to the 1 state, so that A = 0 and B = 1, the output X becomes a 0. Similarly, the third row shows that when input A is changed to the 1 state and B, to 0 state, the output X becomes a 1. And the last row shows when both the inputs A and B are changed to “1” state, the output X becomes a 0.

Table 3-2. Two-inputs truth table

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

Tables 3-3 and 3-4 show the truth tables for a typical three-and four-input logic circuits. Again, each table lists all possible combinations of input logic levels on the left, with the resultant logic level for output X on the right. It may be noted carefully that the actual values for X will depend on the type of logic circuit.

Table 3.3

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

Table 3.4

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

As seen from the table 3.2, 3.3 and 3.4, there are 4 rows for the two-input truth table, 8 rows for a three-input truth table, and 16 rows for the four-input truth table. In general, the list of all possible input combinations is equal to 2^N for N-input truth table. Moreover the list of all possible input combinations follows the binary counting sequence, and so it is an easy matter to write down all of the combinations without missing any. For example, in a two-input truth table, the sequence of combinations is 00, 01, 10 and 11.

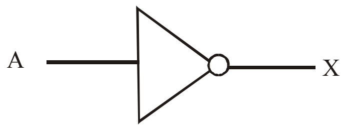
3-4. Boolean Expression

We have already discussed in Chapter 1 that digital or logic circuits operate in the binary mode where each input and output voltage is either a 0 or 1. The 0 and 1 designations represent predefined voltage ranges. The characteristic of logic circuits allows us to use Boolean algebra as a tool for the analysis and design of digital circuits. Boolean algebra is a relatively simple mathematical tool that allows us to describe the relationship between logic circuit output (s) and its inputs as an algebraic equation. This algebraic equation is known as Boolean expression or simply Boolean equation.

3-5. The INVERTER

The INVERTER is also known as a NOT gate. It is a logic gate that performs inversion (also called complementation) operation. It changes one logic level to the opposite logic level, i.e., from HIGH to LOW level or from LOW to HIGH level. In terms of bits, it changes a 1 to 0 and a 0 to 1.

Fig 3-1 (a) shows a logic symbol of an INVERTER. Note that input is shown on the left (labeled as A) and the output is on the right (labeled as X).



(*a*) logic symbol

Input	*Output*
A	*X*
0	1
1	0

(*b*) truth table

Fig. 3-1. *An INVERTER.*

Logical Operation: When the INVERTER input is LOW (0), the output, X is HIGH (1). On the other hand, if the input is HIGH(1), the output is LOW (0). This operation is summarized in the form of a truth table as shown in Figure 3-1 (b).

3-6. Pulsed Operation of an INVERTER

Consider a pulse input to an INVERTER as shown in Figure 3-2. In order to study the pulse operation, we will consider the time instances t_1 and t_2 along the time axis. Thus at t_1, the input goes from LOW(0) to HIGH(1), therefore, the output goes from HIGH(1) to LOW(0). At t_2, the input goes from HIGH(1) to LOW(0), therefore, the output goes from LOW(0) to HIGH(1).

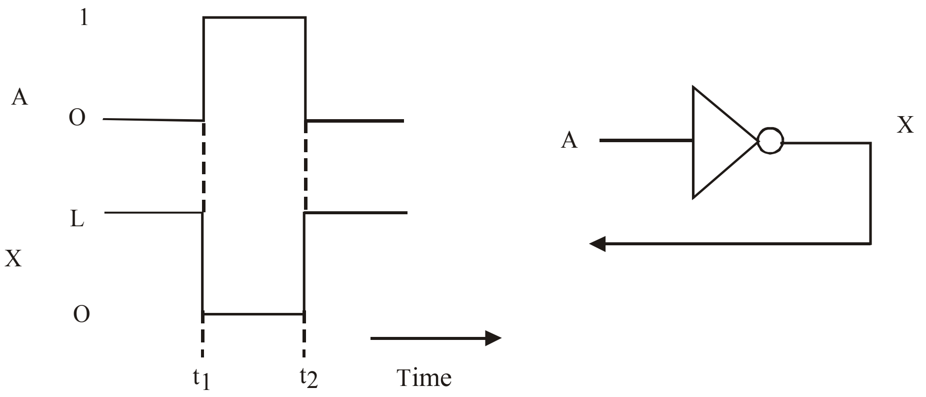


Fig. 3-2. *Illustrating the pulsed operation of an INVERTER.*

It is evident from the figure shown above that an INVERTER produces an output that is opposite or complement of its input.

Example 3-1. *Fig.* 3-3 *shows a waveform applied to the input of an INVERTER. Sketch the output waveform corresponding to the input.*

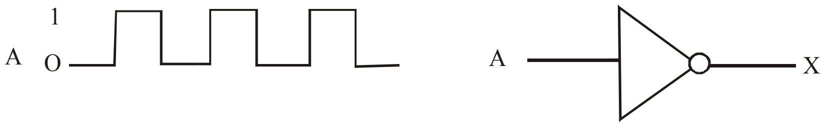


Fig. 3-3

Solution. Given : An INVERTER with a waveform A at its input.

We know that for a given waveform, an INVERTER produces an inverted output waveform. Thus the output waveform it exactly opposite to the input as shown in Figure 3-4.

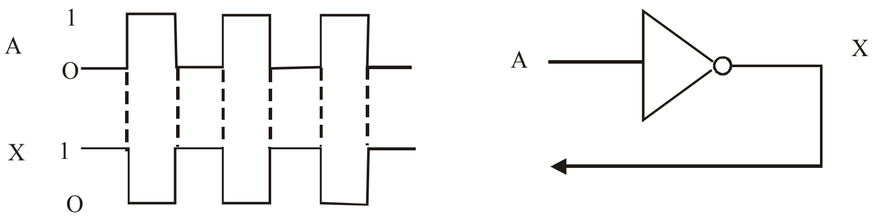


Fig. 3-4

3-7. Boolean Expression for an INVERTER

We have already discussed in Article 3-5 that the INVERTER output is complement of its input, i.e., output is 0 if the input is 1. The output is 1 if the input is 0. This operation of the INVERTER (or NOT gate) can be expressed in the form of Boolean expression as follows:

Let A = the input variable and

X = the output variable,

Then $X = \overline{A}$ ...(*i*)

Equation (*i*) can be read as X *equals A bar.* It states that the output X is complement or opposite of the input A, i.e., $X = 0$ if $A = 1$, and $X = 1$ if $A = 0$.

Example 3-2. *Write the Boolean equation at X for the circuit shown in Fig* 3-5. E*valuate the expression if A* = 1.

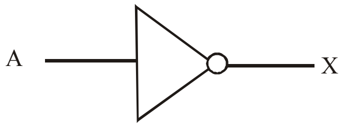


Fig. 3-5

Solution. Given: An INVERTER with input A and output X.

We know that the Boolean equation for an INVERTER is given by the relation:

$$X = \overline{A} \quad ...(i)$$

Value of X when A = 1

Substituting $A = 1$ in equation (*i*), we get,

$$X = \overline{1} = 0 \textbf{ Ans.}$$

Example 3-3. (*a*) *Write the Boolean equation at X and Z for an INVERTER circuit shown in Fig* 3-6. *Evaluate the value of X and Z when A* = 0.

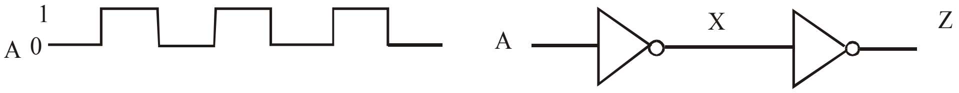


Fig. 3-6

(*b*) *Sketch the waveforms at X an Z if the timing wave form shown in Figure* 3-6 *is input at A*.

Solution. Given : An INVERTER circuit with input labeled as A.

We know that the Boolean expression for the INVERTER '1' is given by the relation,

$$X = \overline{A} \textbf{ Ans.}$$

and the Boolean expression for the INVERTER '2' is,

$$Z = \overline{X} \textbf{ Ans.}$$

Value of X and Z if A = 0

We know that, $X = \overline{A} = \overline{0} = 1$ **Ans.**

and $Z = \overline{X} = \overline{1} = 0$ **Ans.**

(*b*) Given, the waveform shown in Fig. 3-6 is applied at the input A.

We know that for a given input pulse, the INVERTER produces an inverted pulse at its output. Therefore the waveforms obtained at X and Z are as shown in Fig. 3-7.

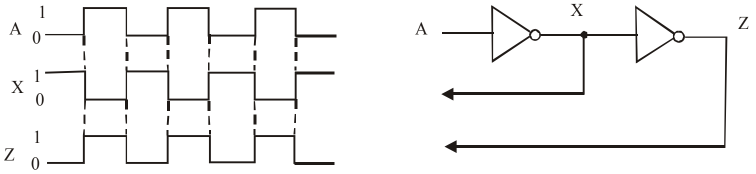


Fig. 3-7

3-8. Application of an INVERTER

Fig 3-8 shows a very simple application of an INVERTER. Here the INVERTER is used for producing 1's complement of an 8-bit number. The bits of the given binary number are applied to the INVERTER inputs and the 1's complement of the number appears on the outputs.

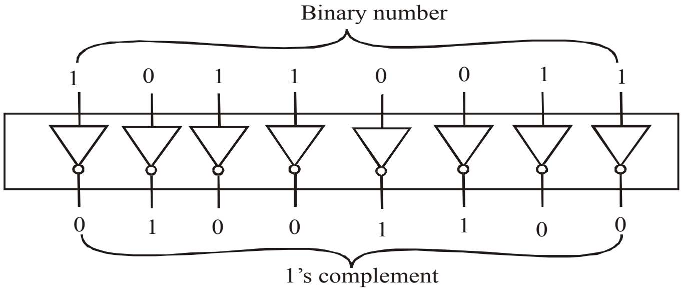
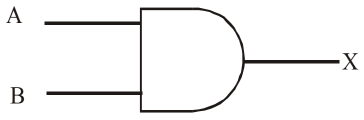


Fig. 3-8. *INVERTER for producing 1's complement*

3-9. The AND Gate

The AND gate is one of the basic gates from which all logic functions are constructed. It is composed of 2 or more inputs and a single output. Fig 3-9(a) shows a logic symbol for a 2-input AND gate. Note that inputs are shown on the left (labeled as A and B) and the output is on the right (labeled as X).



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

(*a*) *logic symbol*

(*b*) *truth table*

Fig. 3-9. *A 2-input AND gate.*

Logical operation. An AND gate produces a HIGH(1) output *only* when *all* of the inputs are HIGH(1). When any of the inputs is LOW(0), the output is LOW(0).

In a 2-input AND gate operation, output X is HIGH(1) if input A and B both are HIGH(1). The output X is LOW(0) if input A is LOW(0), or input B is LOW(0), or if A and B both are LOW(0). This operation is summarized in the truth table shown in Fig 3-9 (b).

Fig. 3-10 shows the resulting output for all the possible input combinations (i.e., 00, 01, 10 and 11).

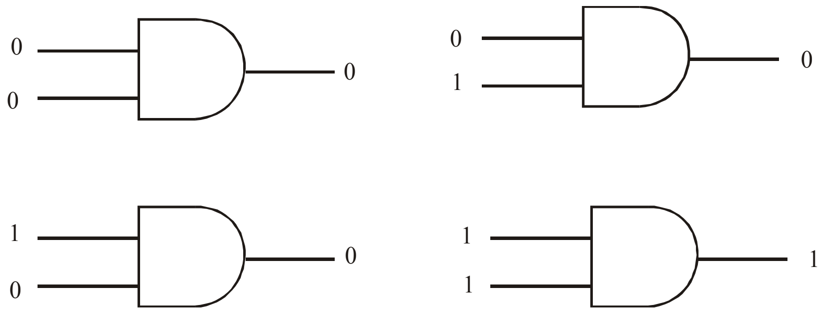


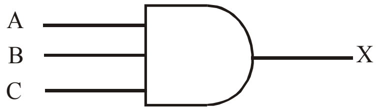
Fig. 3-10. *Illustrating logic operation of a 2-input AND gate for all possible input combinations.*

It is evident from the figure shown above that the output of an AND gate is 1 only when both the inputs are 1. In all other possible combinations (i.e., 00, 01, and 10) the output is 0.

Example 3-4. *Determine the total number of possible input combinations of a* 3-*input AND gate. Sketch a logic symbol and develop a truth table for this logic gate.*

Solution. Given : A 3-input AND gate.

We know that for a 3-input AND gate, there are eight possible combinations ($2^3 = 8$). Figure 3-11 (a) shows the logic symbol and (b) a truth table for a 3-input AND gate.



(*a*) *logic symbol*

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

(*b*) *truth table*

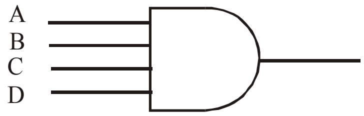
Fig. 3-11.

It is evident from the truth table of a 3-input AND gate that output is 1 only when all the three inputs are 1. The output is 0 if any of the three inputs is 0 or all the three inputs are 0.

Example 3-5. *Determine the total number of possible input combinations of a* 4-*input AND gate. Sketch a logic symbol and develop a truth table for this logic gate.*

Solution. Given: A 4-input AND gate.

We know that for a 4-input AND gate, there are sixteen possible combination ($2^4 = 16$). Fig. 3-12 (a) shows the logic symbol and (b) a truth table for a 4-input AND gate.



(*a*) logic symbol

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

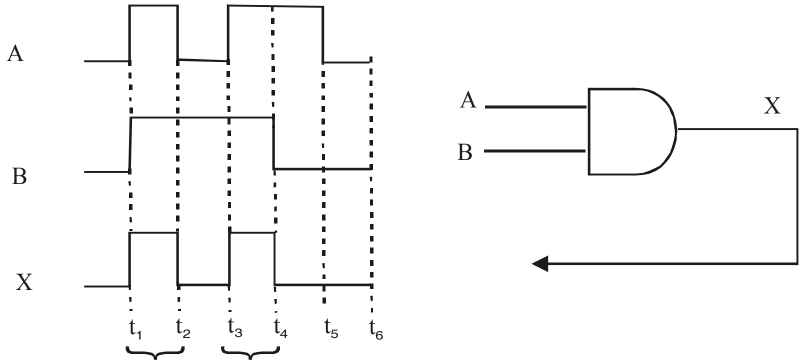
(*b*) *truth table*

Fig. 3-12.

It is evident from the truth table of a 4-input AND gate that output is 1 only when all the four inputs are 1. The output is 0 if any of the four inputs is 0 or all the four inputs are 0.

3-10. Pulsed Operation of an AND Gate

It will be interesting to know that in a majority of applications, the inputs to a logic gate (of any type) are not stationery voltage levels, but voltage waveforms that change frequently between 1 and 0 logic levels. Now let us study the operation of an AND gate with pulse waveform inputs A and B as shown in Fig. 3-13. Keep it in mind that an AND gate obeys the truth table operation (refer to Fig. 3-9 (b), page 72) regardless of whether its inputs are constant levels or levels that change back and forth.



A and B both are 1 during these two intervals, therefore, output X is 1.

Fig. 3-13. *Illustrating pulsed operation of a 2-input AND gate.*

At t = t_1, both inputs A and B goes 1, therefore output, X changes from 0 to 1. At t = t_2, input A goes 0, B = 1, therefore X goes 0. At t = t_3, A goes 1, B = 1, therefore X goes 1. At t = t_4, A = 1, B goes 0, therefore X goes 0. At t = t_5, A goes 0, B = 0, therefore X = 0, At t = t_6, A = 0, B = 0, therefore X = 0. The resulting output waveform at x is shown in Fig. 3.13.

Example 3.6. *Fig* 3-14 *shows the two input waveforms*, *A and B applied to an AND gate.*

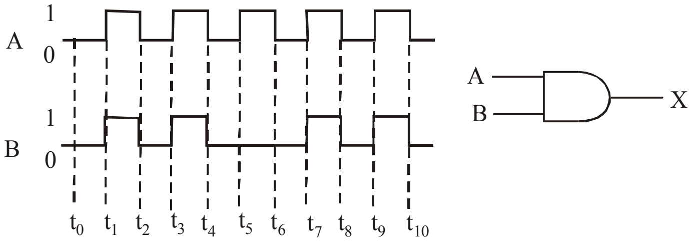

A
1
0
B
1
0
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
A
B
X

Fig. 3-14

Sketch the resulting output waveform at X.

Solution. Given: A two-input AND gate with the waveforms at A and B.

We know that an AND gate output is 1 only when both the inputs A and B are 1. When any one of the inputs is 0, the output is 0. The output is also 0 when both the inputs are 0. Thus at t = $t_{0,}$ A = 0, B = 0, therefore X = 0. At t = t_1, A goes 1, B goes 1, therefore X also goes 1. At t = t_2, A goes 0, B goes 0, therefore X also goes 0, At t = t_3, A and B again goes 1, therefore X goes 1. At t = t_4, both A and B goes 0, Therefore X goes 0. At t = t_5, A goes 1, B = 0, but X remains 0. At t = t_6, A goes 0, B = 0, therefore X still remains at 0. At t = t_7 A goes 1, B goes 1, Therefore X goes 1. At t = t_8, both A and B goes 0, therefore X also goes 0. At t = t_9, both A and B goes 1, therefore X goes 1. At t = t_{10}, both A and B goes 0, therefore X goes 0. The resulting output waveform at X is shown in Fig. 3-15.

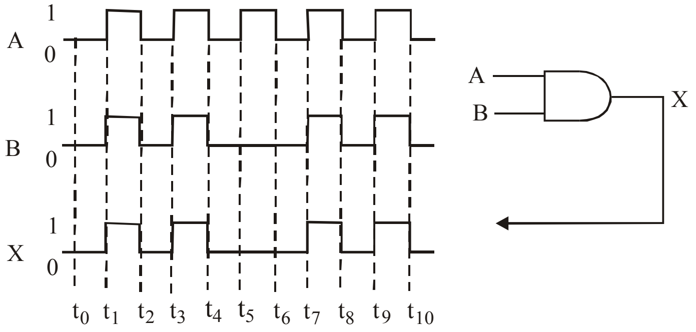

A
1
0
B
1
0
X
1
0
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
A
B
X

Fig. 3-15.

Example 3-7 *Fig* 3-16 *shows the two input waveforms A and B applied to an AND gate.*

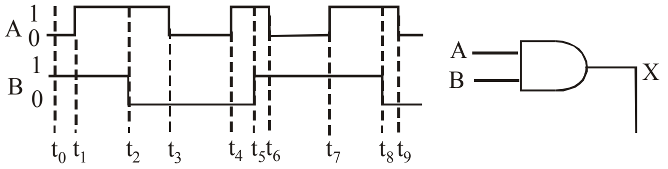


Fig. 3-16.

Sketch the resulting output waveform showing its proper relation to the inputs.

Solution. Given: A two-input AND gate with waveforms at A and B.

We know that an AND gate output is 1 only when both the inputs A and B are 1. When anyone of the inputs is 0, the output is 0. Thus at t = t_0, A = 0, B = 1, therefore X = 0. At t = t_1, A goes 1, B = 1, therefore X goes 1. At t = t_2, A = 1, B goes 0, there X goes 0. At t = t_3, A goes 0, B = 0, therefore X = 0. At t = t_4, A goes 1, B = 0, therefore X = 0. At t = t_5, A = 1, B goes 1,Therefore X goes 1. At t = t_6, A goes 0, B = 1, therefore X goes 0. At t = t_7, A goes 1, B = 1, therefore X goes 1. At t = t_8, A = 1, B goes 0, therefore X goes 0. At t = t_9, A goes 0, B = 0, therefore X = 0 . The resulting output waveform at X is shown in Fig. 3-17.

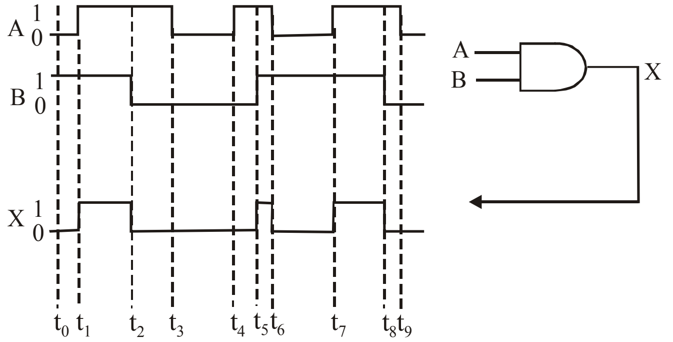


Fig. 3-17.

Example 3-8 *Fig* 3-18 *shows the input waveforms A, B and C applied to an AND gate.*

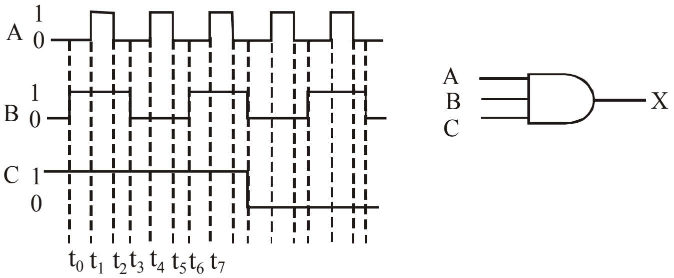


Fig. 3-18.

Sketch the resulting output waveform at X.

Solution. Given: A three-input AND gate with the waveforms at A, B and C.

We know that an AND gate output is 1 only when all the 3-inputs, A, B and C are 1. When any one of three inputs are 0, the output is 0. The output is also 0, when all the three inputs are 0. Thus at t = t_1, A = 0, B goes 1, C = 1, therefore X = 0. At t = t_1, A goes 1, B = 1 and C = 1. Therefore X goes 1. At t = t_2, A goes 0, B = 1 and C = 1, therefore X goes 0.

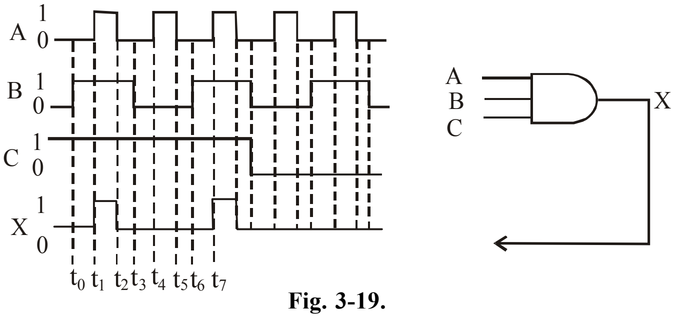


Fig. 3-19.

At $t = t_3$, A = 0, B goes 0, C = 1, therefore X = 0. At $t = t_4$, A goes 1, B = 0, C = 1, therefore X = 0. At $t = t_5$, A goes 0, B = 0, C = 1, therefore X = 0. At $t = t_6$, A = 0, B goes 1, C = 1, therefore X = 0. At $t = t_7$, A goes 1, B = 1, C = 1, therefore X goes 1 and so on... The resulting output waveform at X is shown in Fig 3-19.

3-11. Boolean Expression for an AND gate

We have already discussed in the Art 3-9 that for a 2-input AND gate, output is 1 only if both the inputs A and B are 1. If either of the inputs A and B is 0, the output is 0. This operation of an AND gate can be expressed in the form of a Boolean expression as follows:

$$X = A.B \qquad ...(i)$$

The "." sign in the above expression stands for the Boolean AND operation and not the multiplication operation. However the AND operation on Boolean variables operates the same way as in ordinary multiplication. This fact is shown below:

$$0.0 = 0$$

$$0.1 = 0$$

$$1.0 = 0$$

$$1.1 = 0$$

In actual practice, it is more convenient to express the AND operation without a "." sign. Thus the euation (*i*) can also be written as,

$$X = AB \qquad ...(ii)$$

To extend the AND operation to more than two input variables. Simply use a new letter for each input variable. For example, for a three input AND gate, the Boolean expression is expressed as.

$$X = ABC \qquad ...(iii)$$

Where A, B, and C are the input variables. Similarly for a 4-input AND gate, the Boolean expression is:

$$X = ABCD \qquad ...(iv)$$

Where A, B, C and D are the input variables. Thus this can be extended to an AND gate with any number of inputs.

3-12. Application of an AND Gate

An AND gate can be used in a simple seat belt alarm system in a car to detect when the ignition switch is on *and* the seat belt is unbuckled. If the ignition switch is ON, a 1 is produced on input A of the AND gate shown in Fig. 3-20.

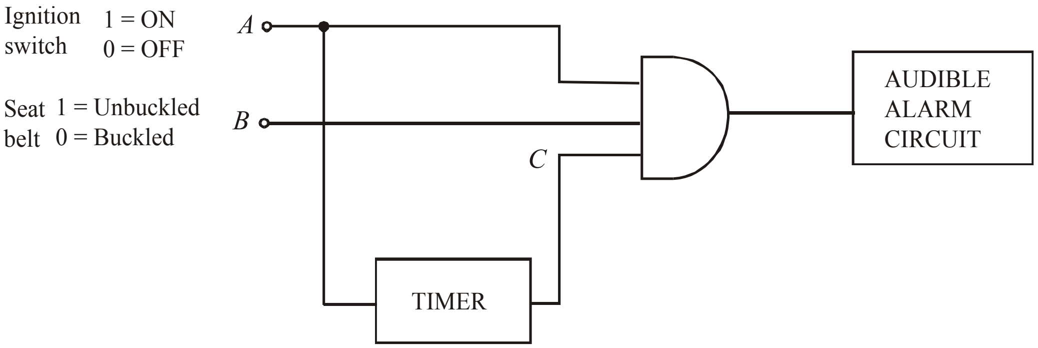


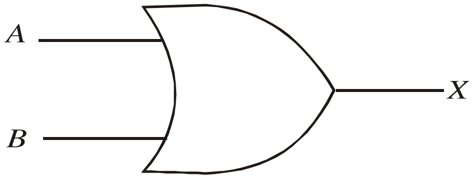
Fig. 3-20. *A simple seat belt alarm circuit using an AND gate.*

If the seat belt is not properly buckled, 1 is produced on input B of the AND gate. Also when the ignition switch is turned ON, a *timer* is started that produces a 1 on input C for 30 seconds.

If all the three conditions exist, i.e. if the ignition switch is ON, *and* the seat belt is unbuckled *and* the time is running, the output of the AND gate is 1. This will activate the audible alarm to remind the driver (say to buckle the seat belt).

3-13. The OR Gate

Like AND gate, the OR gate, is another basic logic gate from which all logic functions are constructed. It is composed of two or more inputs and a single output. Fig-3-21(a) shows a logic symbol for a 2-input OR gate. Notice that the inputs are shown on the left labelled as A and B and the output is on the right (labelled as X).



(*a*) *Logic symbol*

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

(*b*) *Truth table*

Fig. 3-21. *A 2-input OR gate*

Logical operation of the OR gate. The logical operation of the OR gate may be described as below: The OR gate produces a HIGH(1) on the output when *any* of the inputs is HIGH(1). The output is LOW(0) when all the inputs are LOW(0).

In a 2-input OR gate operation, output X is HIGH(1) if either input A or input B is HIGH(1), or if both A and B are HIGH(1). The output, X is LOW(0) if both A and B are LOW (0), This operation is summarized in the truth table shown in Fig 3-21 (b)

Notice that the table shows the resulting output for all the possible input combinations i.e., 00, 01, 10 and 11.

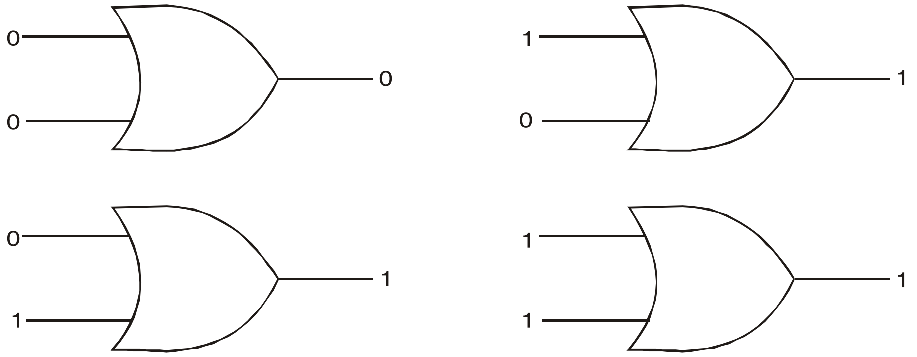


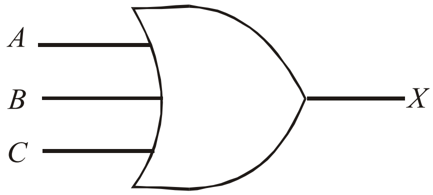
Fig. 3-22. *Illustrating the four possible input combinations and the resulting output for each input combination in a two-input OR gate operation.*

It is evident from the figure shown above, that the output of an OR gate is 0 only when both the inputs are 0. In all other possible input combinations i.e. (01, 10 and 11), the output is 1.

Example 3-9. *Determine the total number of possible input combinations of a* 3*-input OR gate. Also draw a logic symbol and develop the truth table for this logic gate.*

Solution. Given: A 3-input OR gate.

We know that for a 3-input OR gate, there are eight possible input combinations ($2^3 = 8$). Fig 3-23 (a) shows the logic symbol and 3-23 (b) a truth table for a 3-input OR gate.



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

(*a*) Logic symbol (*b*) Truth table

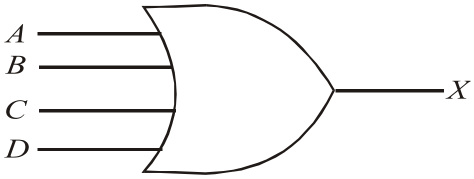
Fig. 3-23. *A 3-input OR gate.*

Note that a 3-input OR gate output is 1 when any of the three inputs, A, B and C is 1. The output is 0 only when all the three inputs are 0.

Example 3-10. *Determine the total number of possible input combinations of a* 4*-input OR gate. Also draw a logic symbol and develop the truth table for this logic gate.*

Solution. Given : A 4-input OR gate.

We know that for a 4-input OR gate, there are sixteen possible input combinations ($2^4 = 16$). Fig. 3-24. (*a*) shows the logic symbol and 3-24 (*b*) a truth table for a 4-input OR gate.



(*a*) Logic symbol

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

(*b*) Truth table

Fig. 3-24. *A 4-input OR gate.*

Note that a 4-input OR gate output is 1 when any of the four inputs A, B, C and D is 1. The output is 0 only when all the four inputs are 0.

3-14. Pulsed Operation of an OR Gate

We know that in a majority of applications, the inputs to a logic gate (of any type) are not stationery voltage levels but are voltage waveforms, that change frequently between 1 and 0 logic levels. Now let us study the operation of an OR gate with pulse waveform inputs. Keep it in mind that an OR gate obeys the truth table operation regardless of whether its inputs are constants levels or levels that change back and forth.

Consider an OR gate with two-inputs A and B as shown in Fig. 3-25. In order to determine the output level, we will look at the inputs with respect to each other with reference to the truth table shown in Fig. 3-21(page 78).

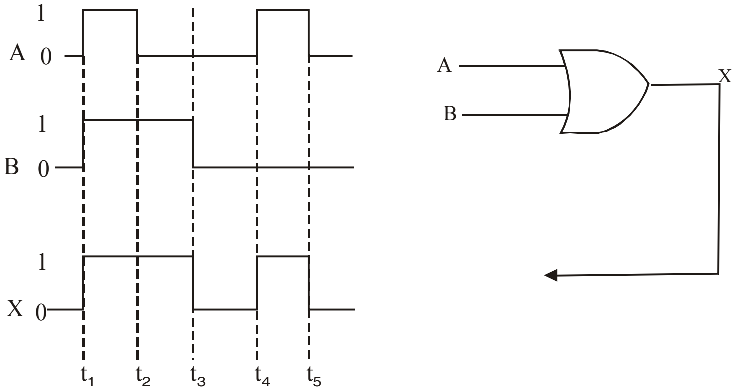


Fig 3-25. *Illustrating pulsed operation of a two-input OR gate.*

At t = t_1, both the inputs A and B goes 1, therefore the output, X changes from 0 to 1. At t = t_2, input A goes 0, B is 1, therefore X is 1, At t = t_3, A is 0, B goes 0, therefore X goes 0. At t = t_4, A goes 1, B is 0, therefore X is 1. The resulting output waveform is as shown in Fig. 3-25.

Example 3-11. *Fig* 3-26 *shows the two input waveforms A and B applied to an OR gate.*

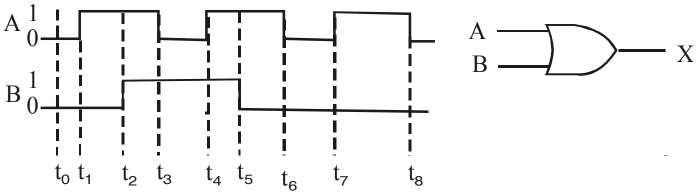


Fig. 3-26

Sketch the resulting output waveform at X.

Solution. Given: The OR gate with the waveforms A and B at its two inputs.

We know that an OR gate output is 1 whenever A is 1 or B is 1 or both A and B are 1. The OR gate output is 0 when both A and B are 0.Thus at t = t_0, A = 0 and B = 0, therefore X = 0. At t = t_1, A goes 1, B = 0, therefore X goes 1. At t = t_2, A = 1, B goes 1, therefore X remains 1. At t = t_3, A goes 0, B = 1, therefore X still remains at 1. At t = t_4, A goes 1, B = 1, therefore X again remains 1. At t = t_5, A = 1, B goes 0. However the output X still remains at 1. At t = t_6, A goes 0, B = 0, therefore

X goes 0. At t = t_7, A goes 1, B = 0, therefore X goes 1, At t = t_8, A goes 0, B = 0, therefore X goes 0. The resulting output waveform is as shown in Fig 3-27.

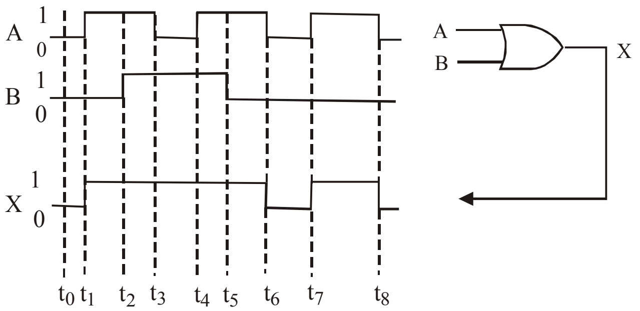


Fig. 3-27.

Example 3-12. *Fig* 3-28 *shows the two input waveforms, A and B applied to an OR gat*e.

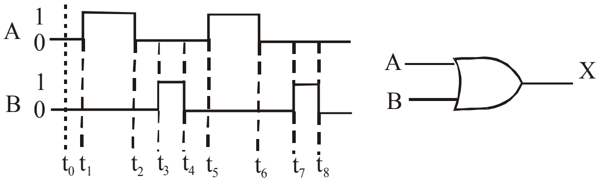


Fig. 3-28.

Sketch the resulting output waveform at X.

Solution. Given: two-input OR gate with waveforms applied at inputs A and B. We know that the OR gate output is 1 whenever A is 1 or B is 1 or both A and B are 1. The OR gate output is 0 when both A and B are 0. Thus at t = t_0, A = 0 and B = 0, therefore X = 0, At t = t_1, A goes 1, B = 0, therefore X goes 1. At t = t_2, A goes 0, B = 0, therefore X goes 0. At t = t_3, A = 0 and B goes 1, therefore X goes 1. At t = t_4, A = 0, B goes 0, therefore X goes 0. At t = t_5, A goes 1, B = 0, therefore X goes 1. At t = t_6, A goes 0, B = 0, therefore X goes 0. At t = t_7, A = 0, B goes 1, therefore X goes 1. At t = t_8, A = 0, B goes 0, therefore X goes 0. The resulting output waveform is as shown in Fig. 3-29.

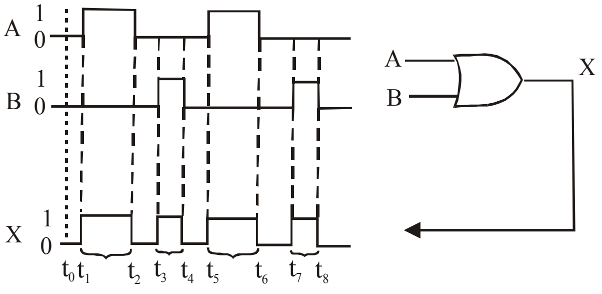


Fig. 3-29.

Example 3-13. *Fig* 3-30 *shows the three input waveforms*, *A*, *B and C applied to an OR gate.*

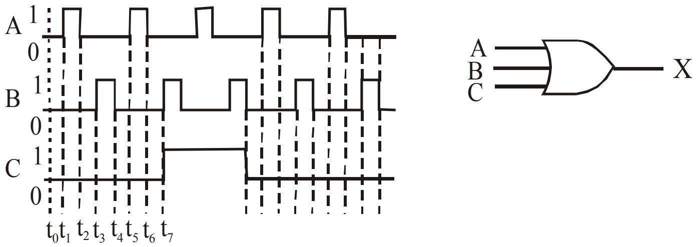


Fig. 3-30.

Sketch the resulting output waveform at X.

Solution. Given: a 3- input OR gate with waveforms at A, B and C inputs.

We know that an OR gate output is 1, whenever A is 1 or B is 1 or C is 1 or all A, B and C are 1. The OR gate output is 0 when all A, B and C are 0. Thus at t = t_0, A = 0, B = 0 and C = 0, therefore x = 0. At t = t_1, A goes 1, B = 0 and C = 0, therefore X goes 1. At t = t_2, A goes 0, B = 0 and C = 0, therefore X goes 0. At t = t_3, A = 0, B goes 1, C = 0, therefore X goes 1. At t = t_4, A = 0, B goes 0, C = 0, therefore X goes 0. At t = t_5, A goes 1, B = 0, C = 0, therefore X goes 1. At t = t_6, A goes 0, B = 0, C = 0, therefore X goes 0. At t = t_7, A = 0, B goes 1, C goes 1, therefore X goes 1 and so on... The resulting output waveform is as shown in Fig. 3-31.

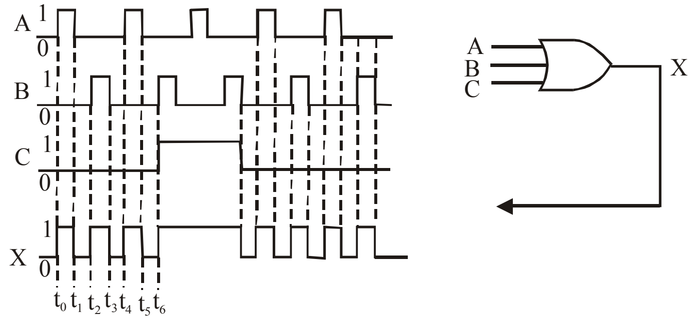


Fig. 3-31.

3-15. Boolean Expression for an OR Gate

We have already discussed in the last article that for a two-input OR gate, the output is 1, whenever either of the inputs A and B is 1 or both the inputs A and B are 1. If both the inputs are 0, the output is 0. This operation of an OR gate can be expressed in the form of a Boolean expression as follows,

$$X = A + B$$

The "+" sign in the above expression stands for the Boolean OR operation or the Boolean addition operation. The OR operation on Boolean variables operates the way as shown below.

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

Notice that the OR operation produces 1+1 = 1, not 1+1 = 2. In Boolean algebra, we can never have a result greater than 1.

The expression, X = A+B is read as X = A OR B. The key theory to remember is that the "+" sign stands for the OR operation and not for ordinary addition.

To extend the OR operation for more than two-input variables, simply use a new letter for each input variable. For example for a three-input OR gate, the Boolean expression is expressed as:

$$X = A + B + C \quad ...(5)$$

where A, B, C are the input variables. Similarly for a four-input OR gate, the Boolean expression is:

$$X = A+B+C+D \quad ...(6)$$

where A, B, C and D are the input variables. Thus this approach can be extended to an OR gate with any number of inputs.

3-16. Application of an OR Gate

An OR gate can be used in a simplified portion of an intrusion detection and alarm system as shown in Fig 3-32. The circuit shown in Fig 3-32 consists of three sensors which can be placed on the doors and windows within a room (say with one door and two windows). The sensors are magnetic switches that produces a 1 output when open and a 0 output when closed. As long as the windows and the door are secured, the switches are closed. Therefore all the three inputs of the OR gate are 0 (Due to this the output is 0 and the alarm is de-activated). When one of the windows or the door is opened, a 1 is produced on the input to the OR gate. As a result of this, the OR gate output becomes 1. This activates an alarm circuit to warn of the intrusion.

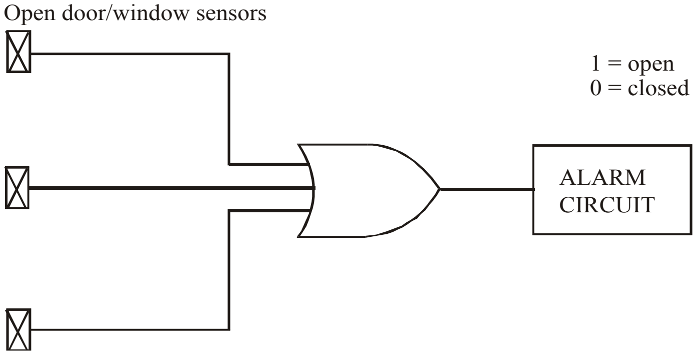


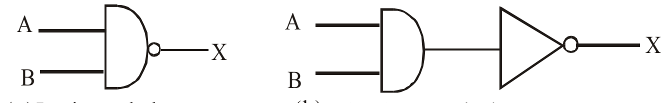
Fig. 3-32. A simplified instrusion detection system using an OR gate

3-17. The NAND and NOR Gate

We have already discussed the NOT, AND and OR gates in the previous articles. Now we will study two other types of logic gates that are used extensively in digital circuitry. These logic gates are the NAND and NOR gates. The NAND and NOR gates actually combine the basic operations AND, OR, and NOT, which make it relatively easy to describe them using Boolean algebra operations. Now we will discuss both these logic gates one by one in the following pages.

3-18. The NAND Gate

The term NAND is a contraction of NOT-AND. It implies an AND function with a complemented (or inverted) output. The standard logic symbol for a 2-input NAND gate is as shown in Fig 3-33 (a). Notice that this symbol is the same as AND gate symbol except for a small circle (or a bubble) on its output. This small circle denotes the invertion operation. Thus the NAND gate operates like an AND gate followed by INVERTER as shown in Fig. 3-33 (b).



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

(*a*) Logic symbol (*b*) NOT-AND equivalent (*c*) Truth table

Fig. 3-33. *2-input NAND gate A.*

Logical Operation of the NAND gate. The logical operation of the NAND gate is described as below. The NAND gate produces a LOW(0) output only when all the inputs are HIGH(1). When any of the inputs is LOW(0), the output will be HIGH(1).

In a 2-input NAND gate operation, output X is LOW(0) if inputs A and B are HIGH(1).The output X is HIGH(1) if either A or B is LOW(0) or if both A and B are LOW(0). This operation can be summarized in the truth table shown in Fig. 3-33(c).

Fig 3-34 illustrates the logical operation of a 2-input NAND gate for all four input combinations, i.e., 00, 01, 10 and 11.

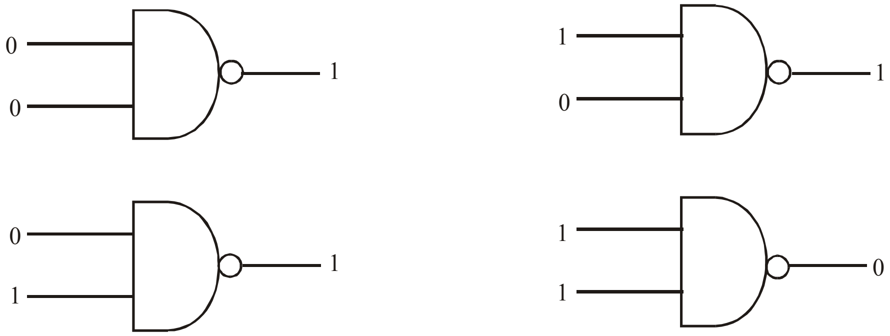


Fig. 3-34. *Illustrating the four possible input combinations and the resulting output for each in a two-input NAND gate operation.*

It is evident from the figure shown above that the output of a NAND gate is 0 only when both the inputs are 1. In all other possible input combinations (i.e., 00, 01 and 10), the output is 1.

3-19. Pulsed Operation of a NAND Gate

As discussed earlier in the case of AND and OR gates, the inputs to the logic gates (of any type) are not stationery voltage levels but are voltage waveforms that change frequently between 1 and 0 logic levels. Now we will study the operation of NAND gate with pulse waveform inputs. Keep it in mind that a NAND gate obeys the truth table operation regardless of whether its inputs are constant levels or levels that change back and forth.

Consider a two waveforms A and B applied at the inputs of a 2-input NAND gate as shown in Fig 3-35. In order to determine the output level, we will look at the inputs with respect to each other with reference to the truth table shown in Fig 3-33 (c).

At $t = t_1$, input A goes 1, B = 1, therefore X goes 0. At $t = t_2$, A = 1, B goes 0, therefore X goes 1. At $t = t_3$, A goes 0, B = 0, therefore X = 1. At $t = t_4$, A goes 1, B = 0, therefore X = 1. At $t = t_4$, A goes 0, B = 0, therefore X = 1. At t = t_5, A = 0, B goes 1, therefore X = 1. At t = t_6, A goes 1, B = 1, therefore X = 0. At t = t_7, A goes 0, B = 0, therefore X goes 1. At t = t_8, A = 0, B goes 0, therefore x = 0. At t = $t_{9,}$ A goes 1, B = 0, Therefore X = 1. The resulting output waveform at X is as shown in Fig.3-35.

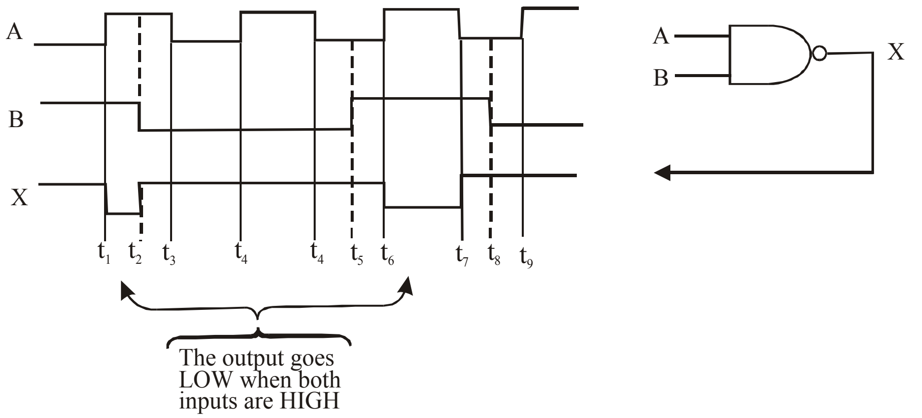

A
B
X
t_1 t_2 t_3 t_4 t_4 t_5 t_6 t_7 t_8 t_9
The output goes LOW when both inputs are HIGH
A
B
X

Fig. 3-35. *Illustrating pulsed operation of a 2-input NAND gate.*

Example 3-14. *Fig* 3-36 *shows the two waveforms applied to the NAND gate inputs. Determine the resulting output waveform.*

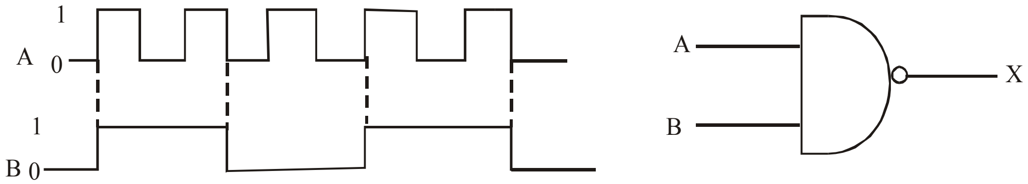


Fig. 3-36.

Solution. Given: A 2-input NAND gate with A and B waveforms.

We know that a NAND gate produces a 0 output when both the inputs A and B are 1. When either of the inputs A or B is 0, or both A and B are 0 the output will be 1. Thus at t = t_1, both A and B goes 1, therefore X goes 0. At t = t_2, A goes 0, B = 1, therefore X goes 1. At t = t_3, A goes 1, B = 1, therefore X goes 0. At t = t_4, A goes 0, B goes 0, therefore X goes 1. Similarly we can determine the output X corresponding to t = t_5, t_6, t_7, t_8, t_9 and t_{10}. The resulting output waveform is as shown in Fig 3-37.

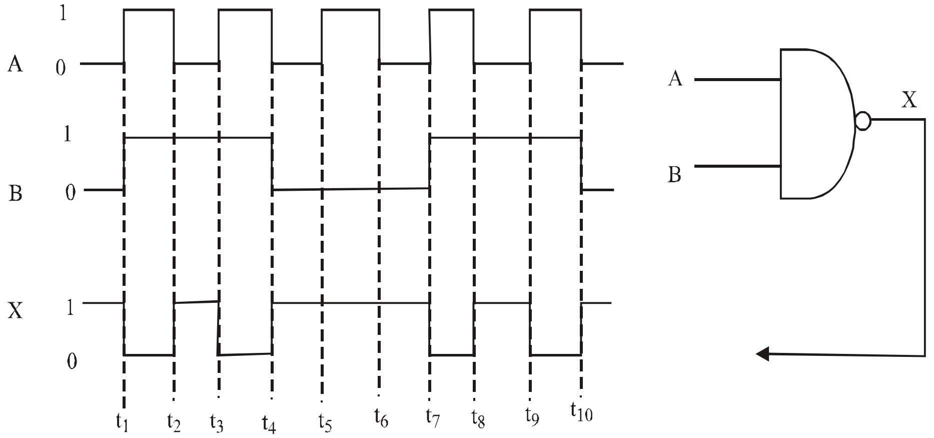

1
A 0
1
B 0
X 1
0
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
A
B
X

Fig. 3-37.

Example 3-15 *Sketch the output waveform for the* 3*-input NAND gate shown in Fig* 3-38 *showing its proper time relationships to the inputs.*

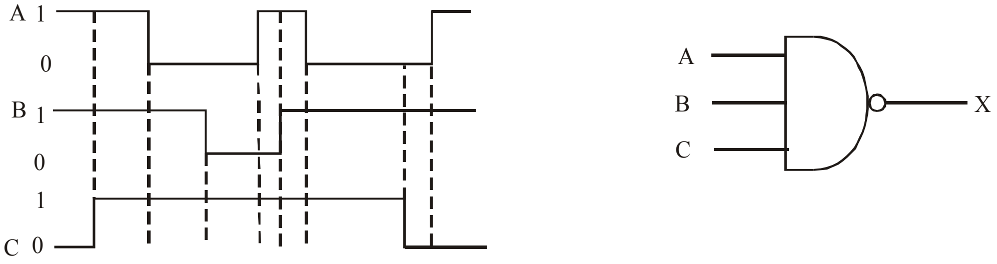
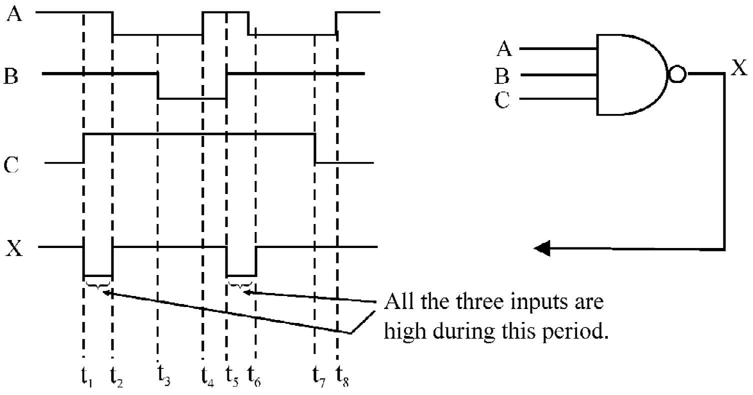


Fig. 3-38.

Solution: Given: A 3-input NAND gate with A, B and C waveforms.

We know that a NAND gate produces 0 output only when all the three inputs, A, B and C are 1. The output is 1, if any of the three inputs A, B, C is 0, or all the three inputs are 0.



All the three inputs are high during this period.

Fig. 3-39.

Thus at t = t_1, A = 1, B = 1, C goes 1, therefore X goes 0. At t = t_2, A goes 0, B = 1, C = 1, therefore X goes 1. At t = t_3, A = 0, B goes 0, C = 1, therefore X = 1. At t = t_4, A goes 1, B = 0, C = 1, therefore X = 1. At t = t_5, A = 1, B goes 1, C = 1, therefore X goes 0. At t = t_6, A goes 0, B = 1, C = 1, therefore X goes 1. At t = t_7, A = 0, B = 1, C goes 0, therefore X = 1. At t = t_8, A goes 1, B = 1, C = 0, therefore X = 1. The resulting output waveform is shown in Fig. 3-39.

A faster way of sketching the waveform is to find the time period during which all the three inputs are 1. During this we know that X is 0. For all other possibilities, X is 1.

3-20. Application of a NAND Gate

Fig 3-40 shows the application of a NAND gate in a drug manufacturing plant that uses two tanks to store a liquid chemical. Each tank has a sensor that detects when the chemical level drops to 25% of full. The sensors produce a 5V level when the tanks are more than one-quarter full. When the level in the tank drops to one-quarter full, the sensor produces a 0V level.

As seen from the figure, the NAND gate with its two inputs is connected to the tank level sensors. The output of the NAND gate is connected to the supply voltage (+ V) through the green LED (a light emitting diode) and a resistor (R). The operation of the system can be explained as follows: As long as both sensor outputs are HIGH(1), indicating that both tanks are more than one-quarter full, the NAND gate output is LOW(0). This turns on the LED. However if either of the two sensor outputs goes LOW(0), the NAND gate output goes HIGH(1), which turns the LED off. Thus a single LED is used to indicate when both tanks are more than one-quarter full.

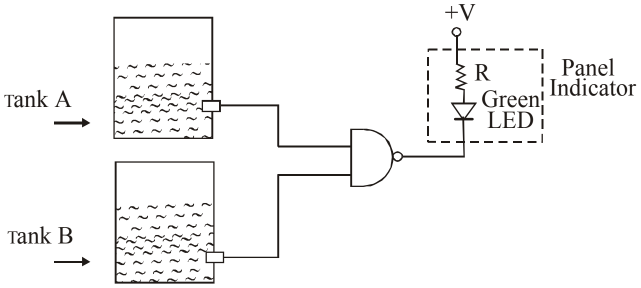


Fig. 3-40. *An electronic system to detect liquid level in a drug manufacturing plant.*

3-21. Boolean Expression for the NAND Gate

We have already discussed in Art 3-18 that in a 2-input NAND gate, output, X is 0 only if inputs A and B are 1. The output, X is 1 if either A or B is 0 or if both A and B are 0. This logical operation of the NAND gate can be expression in the form of a Boolean expression as follows:

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

The above expression is read as X equals AB *bar*. This expression says that the two input variables, A and B are first ANDed and then inverted (or complemented) as indicated by the bar over the AND expression. If we evaluate this expression for all possible values of two input variables, the results are as shown below:

Table 3-5.

A	*B*	$\overline{AB} = X$
0	0	$\overline{0.0} = \overline{0} = 1$
0	1	$\overline{0.1} = \overline{0} = 1$
1	0	$\overline{1.0} = \overline{0} = 1$
1	1	$\overline{1.1} = \overline{1} = 0$

To extend the Boolean expression for the NAND gate, for more than two-input variables, simply use a new letter for each input variable. For example, for a 3-input NAND gate, the Boolean expression is given by the equation:

$$X = \overline{ABC}$$

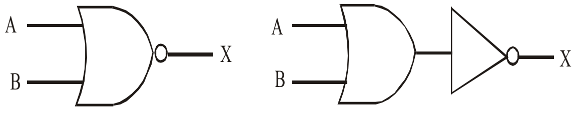
where A, B and C are the input variables. Similarly, for a 4-input NAND gate, the Boolean expression is given by the equation,

$$X = \overline{ABCD}$$

where A, B, C, and D are the input variables.

3-22. The NOR Gate

The term NOR is a contraction of NOT-OR. It implies an OR function with a complemented (or inverted) output. The standard logic symbol for a 2-input NOR gate is as shown in Fig 3-41 (a). Notice that this symbol is the same as OR gate symbol except for a small circle (or a bubble) on its output. This small circle denotes the inversion operation. Thus the NOR gate operates like an OR gate followed by INVERTER as shown in Fig. 3-41 (b).



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

(*a*) Logic symbol (*b*) NOT-OR equivalent (*c*) Truth table

Fig. 3-41. *A 2-input NOR gate.*

Logical operation of NOR gate. The logical operation of a NOR gate is described as below. The NOR gate produces a LOW(0) output when *any* of the inputs is HIGH(1) . The output is HIGH(1) only when all of the inputs are LOW(0).

In a 2-input NOR gate operation, output X is LOW(0) if either input A or input B is HIGH(1) or if both the inputs A and B are HIGH(1). The output X is HIGH(1) if both A and B are LOW(0). This operation can be summarized in the truth table shown in Fig 3-41(c).

Figure 3-42 illustrates the logical operation of a 2-input NOR gate for all the four input combinations, i.e., 00, 01, 10 and 11.

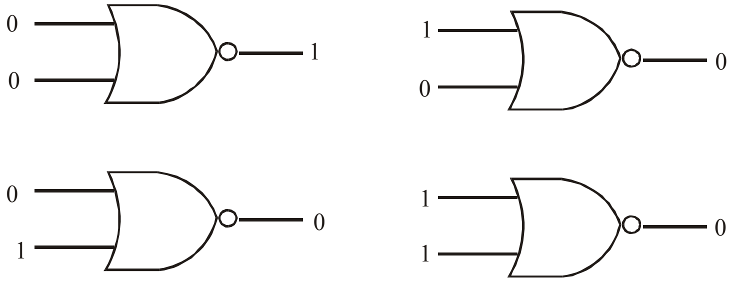


Fig 3-42. *Illustrating the four possible input combinations and the resulting output for each in a two-input NOR gate operation.*

It is evident from the figure shown above that the output of NOR gate is 1 only when both the inputs are 0. In all other possible input combinations (i.e. 01, 10, and 11), the output is 0.

3-23. Pulsed Operation of NOR Gate

Consider waveforms A and B applied at the inputs of a 2-input NOR gate as shown in Fig. 3-43. In order to determine the output level, we will look at the inputs with respect to each other with reference to the truth table shown in Fig. 3-41 (c)

At $t = t_0$, A and B both are 0, therefore X is 1, At $t = t_1$, A goes 1, B is 0, therefore X goes 0, At $t = t_2$, A goes 0, B is 0, therefore X goes 1.At $t = t_3$, A is 0, B goes 1, therefore X goes 0. At $t = t_4$, A is 0, B goes 0, therefore X goes 1, At $t = t_5$, A goes 1, B is 0, therefore X goes 0. At $t = t_6$, A goes 0, B is 0, therefore output X goes 1. The resulting output waveform is as shown in Fig 3-43.

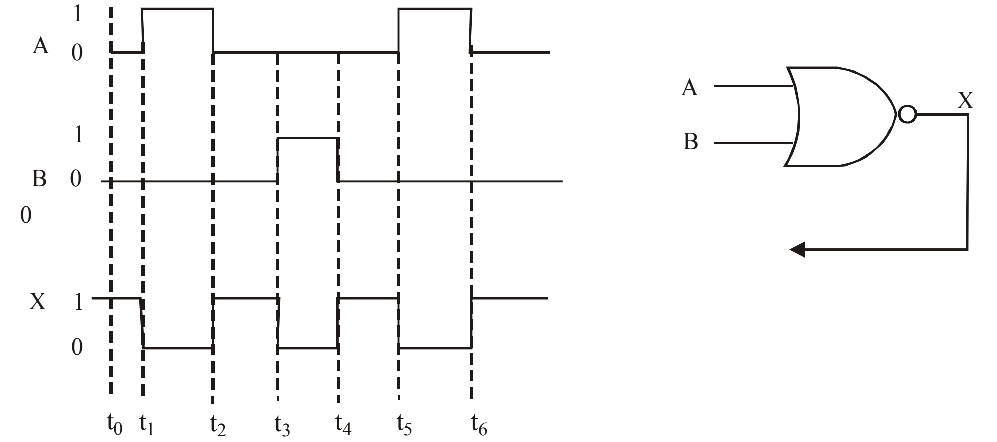


Fig. 3-43. *Illustrating pulsed operation of a 2-input NOR gate.*

Example 3-16. *Sketch the output waveform for a* 3-*input NOR gate shown in Fig* 3-44. *showing the proper time relationships to the inputs.*

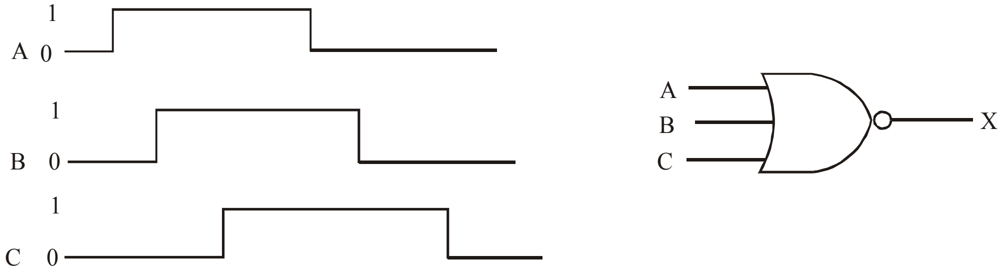


Fig. 3-44.

Solution. Given a 3-input NOR gate with waveforms A, B and C.

We know that in a NOR gate, the output is 0 when any of the inputs is 1. The output is 1, only when all the inputs are 0. With this point in mind, we note that before $t = t_1$, all the three inputs A, B and C are 0. Therefore output is 1. At $t = t_1$, A goes 1, B = 0, C = 0 therefore X goes 0. Between t_1 and t_6, one of the inputs A, B and C is 1, therefore output is 0. At $t = t_6$, A = 0, B = 0, C goes 0, therefore X goes 1.

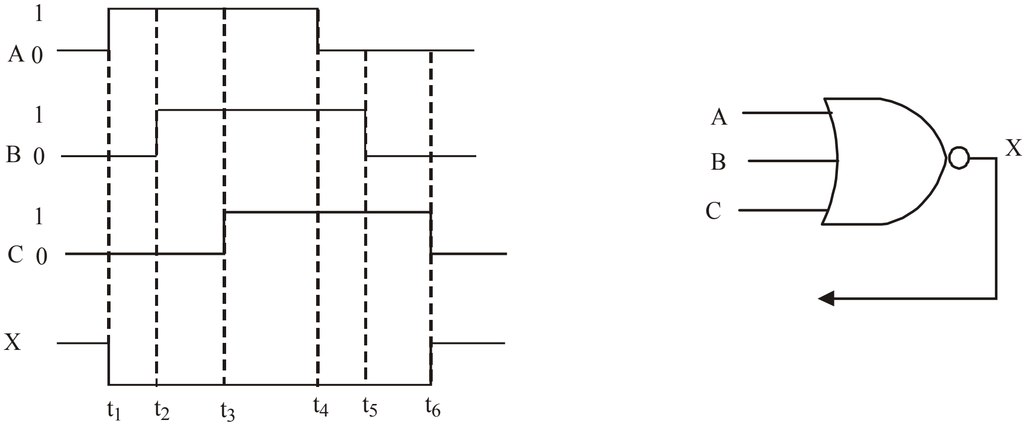


Fig. 3-45.

The resulting output waveform at X is as shown in Fig 3-45.

3-24. Application of a NOR Gate

Fig. 3-46 shows a NOR-gate circuit required to indicate the status of the landing gear prior to landing in an aircraft, as a part of its functional monitoring system. In this circuit, when a landing gear is extended, its sensor produces a LOW(0) voltage. When a landing gear is retracted, its sensor produces a HIGH (1) voltage.

A green LED display turns on if all the three gears are properly extended when the "gear down" switch is activated in preparation for landing. A red LED display turns on if any of the gears fail to extend properly prior to landing.

It is evident from the above diagram that LED is on when the output of NOR-gate 1 is LOW(0). This could be possible only if any of the inputs to NOR-gate 1 is HIGH(1), (i.e., when any of the gears is retracted).

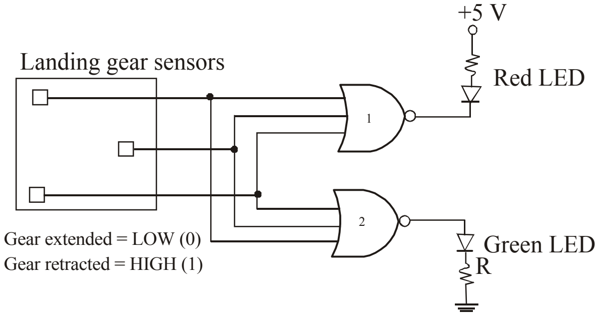


Fig. 3-46. *A NOR-gate circuit to indicate the status of the landing gear prior to landing in an aircraft.*

On the other hand, green LED is on when the output of NOR-gate 2 is HIGH(1). This could be possible only when all the three inputs are low (i.e., when all the gears are extended).

3-25. Boolean Expression for a NOR Gate

We have already discussed in Art 3-22 that in a 2-input NOR gate, the output X is LOW(0) if either input A or input B is HIGH(1) or if both A and B are HIGH(1). The output, X is HIGH(1) if both A and B are LOW(0). This logical operation of the NOR gate can be expressed in the form of a Boolean expression as follows:

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

This equation says that the two input variables are first ORed and then complemented. It is read as complement of A OR B (or A OR B bar). If we evaluate the expression $\overline{A + B}$ for all possible values of two input variables, the results are as shown below.

Table 3-6.

A	B	$\overline{A+B}=X$
0	0	$\overline{0+0}=\overline{0}=1$
0	1	$\overline{0+1}=\overline{1}=0$
1	0	$\overline{1+0}=\overline{1}=0$
1	1	$\overline{1+1}=\overline{1}=0$

To extend the Boolean expression for the NOR gate for more than 2-input variables, simply use a new letter for each input variables, For example, for a 3-input NOR gate, the Boolean expression is given by the equation,

$$X=\overline{A+B+C}\cdot$$

where A, B and C are the input variables. Similarly. For a 4-input NOR gate, the Boolean expression is,

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

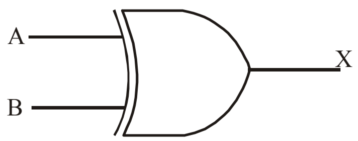
where A, B, C and D are the input variables.

3-26. The Exclusive–OR and Exclusive-NOR Gates

The exclusive –OR and exclusive –NOR gates are actually formed by a combination of other logic gates (i.e., INVERTER, AND and OR gates). However because of their fundamental importance in many applications, the exclusive –OR and exclusive –NOR gates are treated as basic logic gates with their unique symbols. Both these logic gates are discussed one by one in the following pages,

3-27. The Exclusive-OR Gate

The exclusive –OR (abbreviated as XOR) gate has only two-inputs. Unlike the other logic gates we have discussed earlier, the exclusive-OR gate has never more than two inputs. Fig. 3-47 (a) shows the logic symbol of an exclusive –OR gate.



(*a*) Logic symbol

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

(*b*) Truth table

Fig. 3-47. *An Exclusive - OR gate.*

Logic operation. The logical operation of an exclusive-OR gate may be described as below: The output of an exclusive-OR gate is HIGH(1) only when the two inputs are at opposite logic levels. The output is LOW(0) when the two inputs are at same logic levels. This operation can be stated as follows with reference to inputs A and B and output X. In an exclusive–OR gate operation, output X is HIGH(1) if input A is LOW(0) and input B is HIGH(1), or if input A is HIGH(1) and input B is LOW(0). The output X is LOW(0) if A and B are both HIGH(1) or both LOW(0). This operation can be summarized in the truth table shown in Fig. 3-47 (b)

Fig 3-48 shows the four possible input combinations (i.e., 00, 01, 10 and 11) and the resulting output for the exclusive –OR gate.

It is evident from the figure shown above that the output of the exclusive –OR gate is 1 only when the inputs are at opposite logic levels (i.e., 01 or 10). The output is 0 when both the inputs are at same logic levels (i.e., 00 or 11).

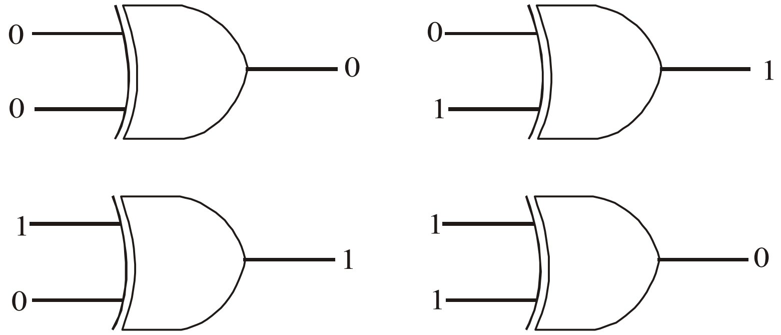


Fig. 3-48. *Illustrating the four possible input combinations and the resulting output for each in a exclusive –OR gate.*

3-28. Pulsed Operation of an Exclusive–OR Gate

Consider waveforms A and B applied at the inputs of an exclusive -OR gate as shown in Fig. 3-49.

We know that in exclusive –OR gate the output is HIGH(1) if the input A is LOW(0) and input B is HIGH(1), or if input A is HIGH(1) and input B is LOW(0). The output X is LOW(0) if A and B are both HIGH(1) or both LOW(0).

In other words, the output of an exclusive –OR gate is HIGH(1) when the input A and input B are at opposite logic levels. The output is LOW(0) if the input A and input B are at the same logic levels. Let us apply these points to sketch the output waveform.

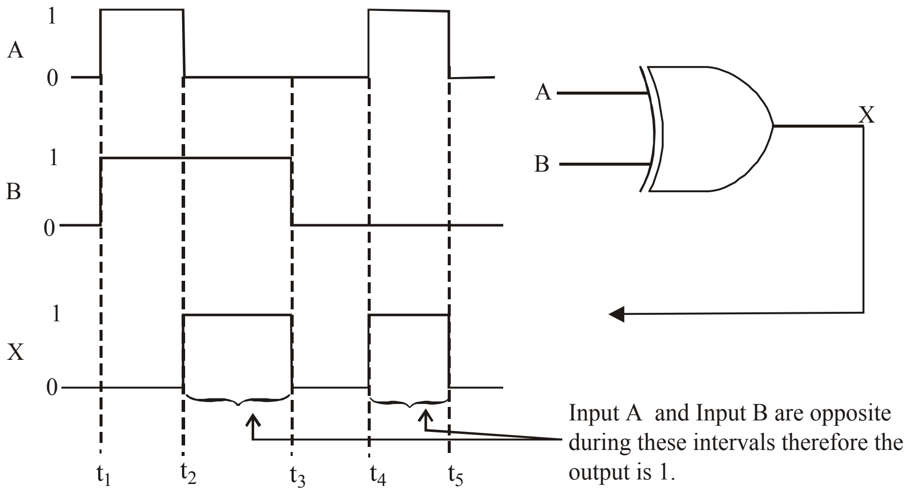

A
1
0
B
1
0
X
1
0
t1
t2
t3
t4
t5
A
B
X
Input A and Input B are opposite during these intervals therefore the output is 1.

Fig. 3-49. *Illustrating the pulsed operation of an exclusive -OR gate.*

At t_1, input A goes 1, input B also goes1, Therefore output X is 0, At t_2, A goes 0, B is 1, therefore output is 1. At t_3, A is 0, B goes 0, therefore X goes 0. At t_4, A goes 1, B is 0, therefore X goes 1. At t_5, A goes 0, B is 0, therefore X goes 0. The resulting output waveform is as shown in figure 3-50.

Another way of sketching the output waveform is to identify the time intervals of the waveforms A and B, during which the inputs are at different logic levels. The output X is 1 during these time intervals. For the rest, the output X is 0. Refer to the Fig. 3-49.

Example 3-17 *Sketch the output waveform for an Exclusive –OR gate with the input waveforms as shown in Fig.* 3-50.

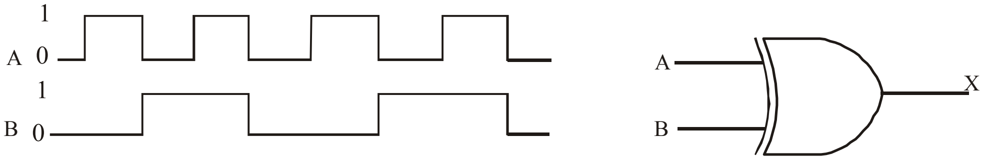


Fig. 3-50.

Solution. Given an exclusive –OR gate with two input waveforms.

We know that an exclusive –OR gate produces a HIGH (1) output whenever its two inputs are at opposite logic levels. On the other hand, the exclusive-OR gate produces a LOW(0) output whenever the two inputs are at the same logic levels. Using this, we can identify the time intervals from the given input waveforms, during which the two inputs are at opposite logic levels or at same logic levels. We can sketch an output 1 and 0 respectively for these time intervals as shown in Fig 3-51.

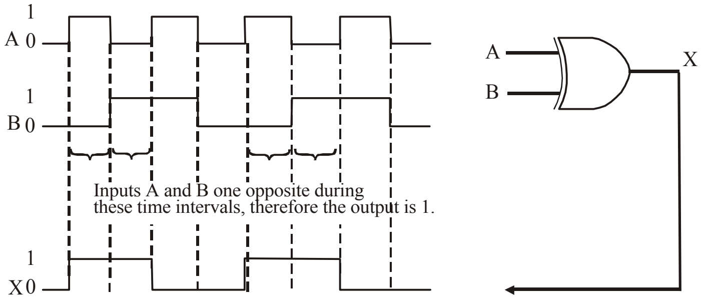

1
A 0
1
B 0
Inputs A and B one opposite during
these time intervals, therefore the output is 1.
1
X 0
A
B
X

Fig. 3-51.

3-29. Application of an Exclusive–OR Gate

Fig 3-52 shows an exclusive –OR gate being used to detect the failure in either one of the identical circuits operating in parallel. As long as both the circuits are operating properly, the outputs of both the circuits are always the same. Therefore the output of the exclusive –OR gate is LOW (0). However, if one of the circuits fails, the outputs of the two circuits will be at opposite levels, for some time. Therefore the exclusive –OR gate will produce a HIGH(1) output indicating failure is one of the circuits.

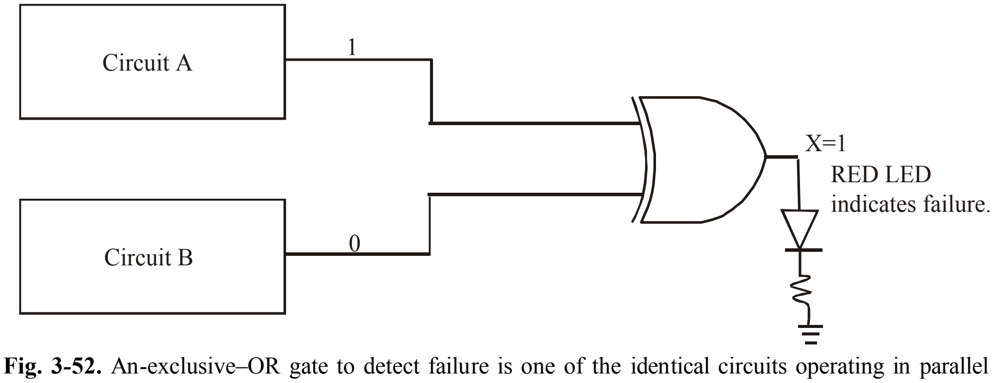


Fig. 3-52. An-exclusive–OR gate to detect failure is one of the identical circuits operating in parallel

3-30. Boolean Expression for an Exclusive–OR Gate

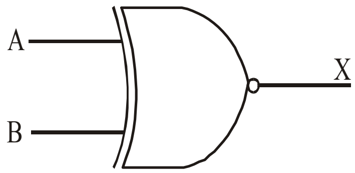
The Boolean expression for an exclusive –OR gate is given by the equation,

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

Note that unlike the INVERTER, AND and OR gates we have two terms in the Boolean expression of the exclusive –OR gate.

3-31. The Exclusive –NOR Gate

The exclusive –NOR (abbreviated as XNOR) gate has also only two inputs like exclusive –OR gate. Fig 3-53(a) shows the logic symbol of an exclusive –NOR(XNOR) gate. The bubble on the output of the XNOR gate symbol indicates that its output is opposite to the output of the XOR gate.



(*a*) Logic symbol

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

(*b*) Truth table

Fig. 3-53. *The exclusive –NOR gate.*

Logical Operation. The logical operation of an exclusive –NOR (X NOR) gate may be described as below: The output of an XNOR gate is HIGH(1) only when the two inputs are at the same logic level. The output is LOW(0) when the two inputs are at opposite logic levels. This operation can be stated as follows with reference to inputs A and B and output X.

In an exclusive-NOR gate operation, output X is LOW(0) if input A is LOW(0) and input B is HIGH(1), or if A is HIGH(1) and B is LOW(0), The output X is HIGH(1) if A and B are both HIGH(1) or both LOW(0).

This operation can be summarized in the truth table shown in Fig 3-53 (b)

Fig 3-54 shows the four possible input combinations (i.e., 00, 01, 10 and 11) and the resulting output for the exclusive –NOR gate.

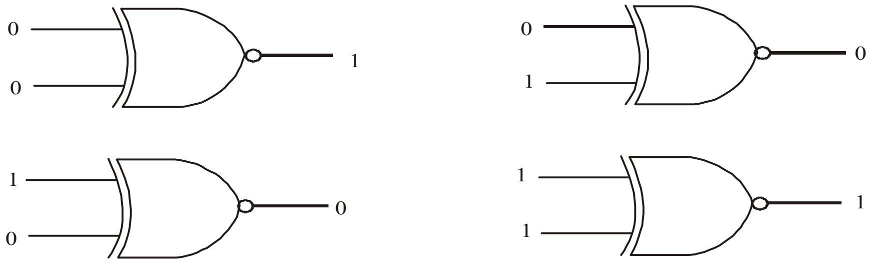


Fig. 3-54. *Illustrating the four possible input combinations and the resulting output for each in an exclusive –NOR gate.*

It is evident from the figure shown above that the output of exclusive-NOR gate is 1 only when the inputs are at same logic levels (*i.e.*, both 0 or both 1). The output is 0 when both the inputs are at opposite logic levels (i.e. 01 or 10).

3-32. Pulsed Operation of an Exclusive–NOR Gate

Consider waveforms A and B applied at the inputs of an exclusive NOR gate as shown in Fig 3-55. We know that in exclusive-NOR gate, the output X is LOW(0) if input A is LOW(0) and input B is HIGH(1), or if A is HIGH(1) and B is LOW(0). The output X is HIGH(1) if A and B are both HIGH(1) or both LOW(0).

In other words, the output of an exclusive-NOR gate, is HIGH(1), when the input A and input B are both at the same logic levels. The output is LOW(0), when the input A and input B are at different logic levels. Let us apply these points to sketch the output waveform.

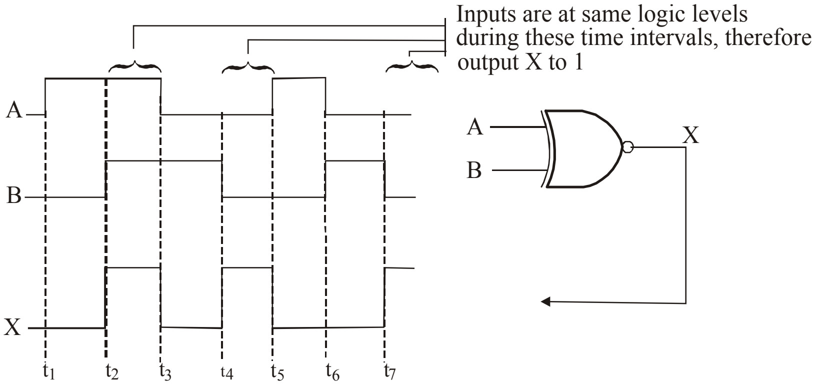

Inputs are at same logic levels
during these time intervals, therefore
output X to 1
A
B
X
t_1
t_2
t_3
t_4
t_5
t_6
t_7
A
B
X

Fig. 3-55. *Illustrating Pulsed operation of an exclusive-NOR gate.*

At t_1, input A goes 1, B is 0, therefore X is 0, At t_2, A is 1, B goes 1, therefore X goes 1. At t_3, A goes 0, B is 1, therefore X goes 0. At t_4, A is 0, B goes 0, therefore X goes 1. At t_5, A goes 1, B is 0. Therefore X goes 0. At t_6, A goes 0, B goes 1, therefore X is 0, At t_7, A is 0, B goes 0, therefore X goes 1. The resulting output waveform is as shown in figure.

Another way of sketching the output waveform is to identify the time intervals of the waveforms A and B during which the inputs are at same logic levels. The output X is 1 during these intervals. For the rest, the output X is 0. The resulting output waveform is shown in Fig. 3-54.

Example 3-18. *Sketch an output waveform for an exclusive-NOR gate with the input waveforms as shown in Fig* 3-56.

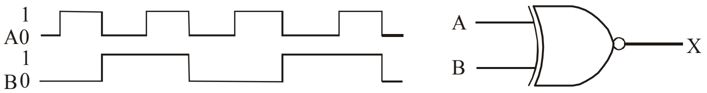


Fig. 3-56.

Solution. Given the exclusive –NOR gate with two input waveforms.

We know that an exclusive –NOR gate produces a HIGH(1) output whenever the two inputs are at same logic levels. On the other hand, the exclusive–NOR gate produces a LOW(0) output whenever the two inputs are at opposite logic levels. Using this, we can identify the time intervals from the given input waveforms during which the two inputs are at the same and opposite logic levels. Sketch an output 1 and 0 respectively for these time intervals as shown in Fig. 3-57.

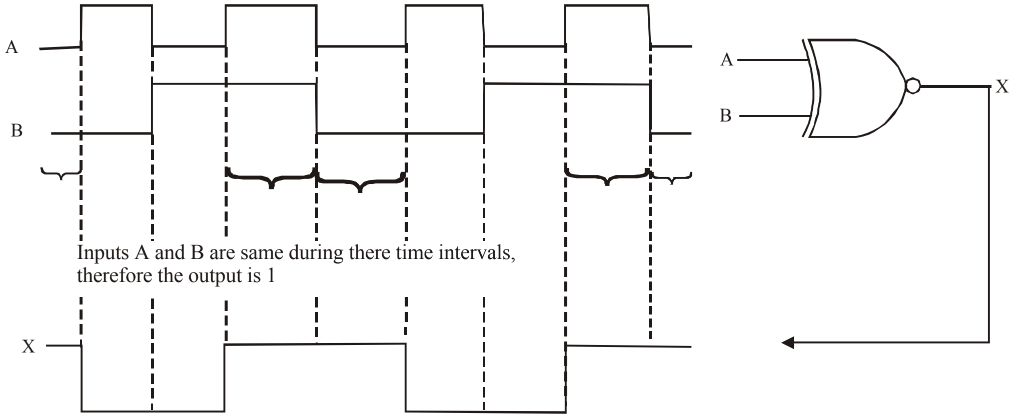

A
B
Inputs A and B are same during there time intervals,
therefore the output is 1
X
A
B
X

Fig. 3-57.

3-33. Application of an Exclusive-NOR Gate

The exclusive –NOR gate can also be used to detect failure in either one of the identical circuits operating in parallel (i.e., the same application as discussed for exclusive –OR gate) as shown in Fig 3-58. As long as both the circuits are operating properly the outputs of both the circuits are always the same. Therefore the output X of the exclusive NOR gate is HIGH(1). However, if one of the circuits fails, the outputs of the two circuits will be at opposite levels for some time. Therefore, the exclusive –NOR gate will produce a LOW(0) output indicating a failure in one of the circuits.

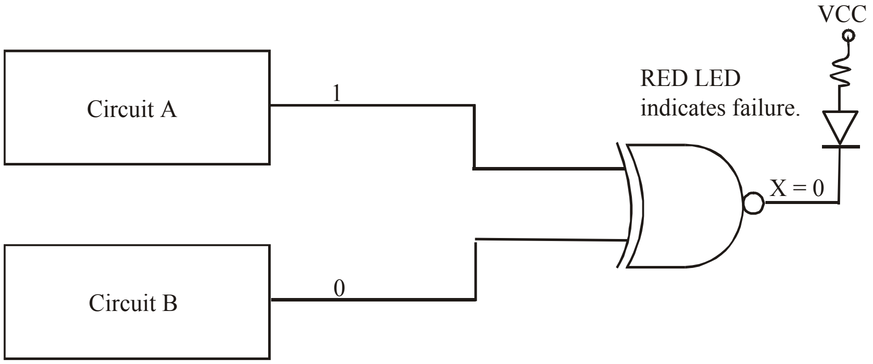


Fig. 3-58. *An exclusive-NOR gate to detect failure in one of the identical circuits operating in parallel.*

3-34. Boolean Expression for an Exclusive-NOR Gate

The Boolean expression for an exclusive –NOR gate is given by the equation,

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

Note that like exclusive– OR gate, we have two terms in the Boolean expression of the exclusive NOR gate.

3-35. Determination of Boolean Expression for a Logic Circuit

We have already discussed in last articles that the INVERTER (or NOT gate), AND and OR gates are the basic building blocks of a digital system. A digital system consists of logic circuits. Further any logic circuit, no matter how complex it may be, is completely described using the Boolean operations. For example, consider the circuit shown in Fig. 3-59

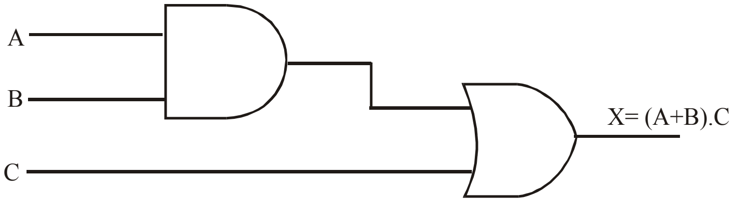


Fig. 3-59. *Logic circuit with its Boolean expression.*

This circuit has three inputs, *A*, *B*, *C*, and a single output, *X*. Utilizing the Boolean expression for each gate, we can easily determine the expression for the output.

The expression for the AND gate output in written as *A*. *B*. This AND output is connected as an input to the OR gate along with *C*, another input. The OR gate operates on its inputs so that its output is the OR sum of the inputs. Thus, we can express the OR output as $X = A.\ B + C$. This final expression could also be written as $X = C + A.\ B$, since it does not matter which term of the OR sum is written first.

Occasionally, there may be confusion as to which operation is an expression is performed first. The expression $A.\ B+ C$ can be interpreted in two different ways: (1) $A.\ B$ is ORed with C or (2) A is ANDed with the term $B + C$. In order to avoid this confusion, the students are advised to follow the following approach.

If an expression contains both AND and OR operations, the AND operations are performed first, unless there are *parentheses* in the expression, in which case the operation inside the parentheses is to be performed first. This is the actually the same rule that is used-in ordinary algebra to determine the order of operations.

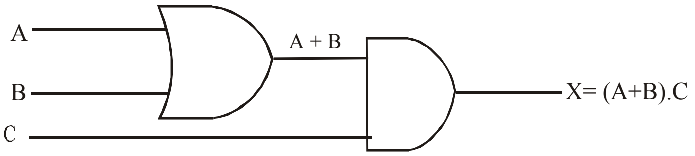


Fig. 3-60. *Logic circuit diagram whose output expression requires parentheses.*

In order to illustrate this point further, let us consider the logic circuit shown in fig 3-60. The expression for the OR gate output is simply $A+B$. This output serves as an input to the AND gate along with another input, C. Thus, we express the output of the AND gate as $X= (A + B)$. C. Note the use of parentheses here to indicate that A and B are ORed first, before their OR sum is ANDed with C. Without the parentheses it would be interpreted incorrectly, since $A + B.\ C$ means that A is ORed with the product $B.\ C$.

3-36. Boolean Expression for a Logic Circuit Containing INVERTERs.

Whenever an INVERTER is present in a logic circuit diagram, its output expression is simply equal to the input expression with a bar over it. Fig. 3.61 shows two examples using INVERTER. In Fig 3-61(a), input A is fed through an INVERTER, whose output is fed to an OR gate together with B, so that the OR Output is equal to $\overline{A}+B$. Note the bar is over the A alone, indicating that A is first inverted and then ORed with B.

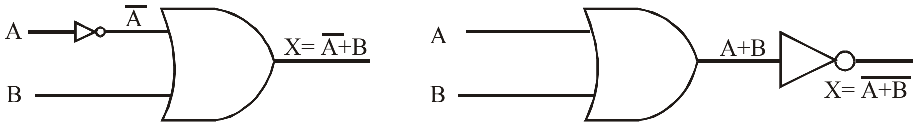


Fig. 3-61. *Logic circuit containing inverters.*

Now consider the circuit shown in fig 3-61 (b). In this circuit, the output of the OR gate is equal to $A + B$ and is fed through the INVERTER. The INVERTER output is therefore equal to $(A+B)$, since it inverts the complete input expression. Note that the bar covers the entire expression $(A+B)$. This is important because, as will be shown later, the expressions $(\overline{A+B})$, and $(\overline{A}+\overline{B})$ are not equivalent. The expression $(\overline{A+B})$ means that A is ORed with B and then their OR sum is inverted, whereas the expression $(\overline{A}+\overline{B})$ indicates that A is inverted and B is inverted and the results are then ORed together.

Example 3-19. W*rite the Boolean expression for output X for the logic circuit shown in Fig*. 3-62.

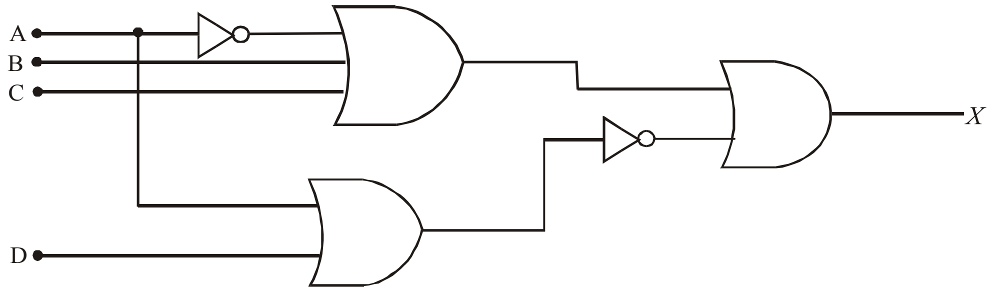


Fig. 3-62.

Solution. Given: The logic circuit containing INVERTER, AND and OR gates of Fig 3-62.

In order to determine the Boolean expression for output X, determine the output of each logic gate while starting from the left-most inputs and work towards the final output writing the expression for each gate.

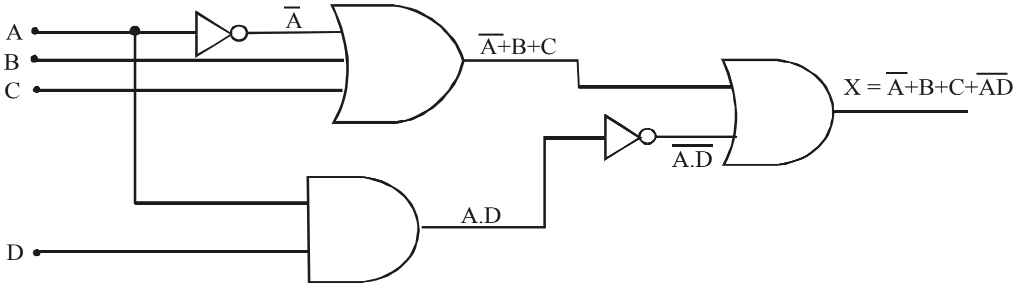


Fig. 3-63.

As seen from circuit shown in Fig 3-63, we find that the Boolean expression for the output,

$$X = \bar{A} + B + C + \overline{AD} \quad \textbf{Ans.}$$

Example 3-20. *Write the Boolean expression for output X of the logic circuit shown in Fig.* 3-64.

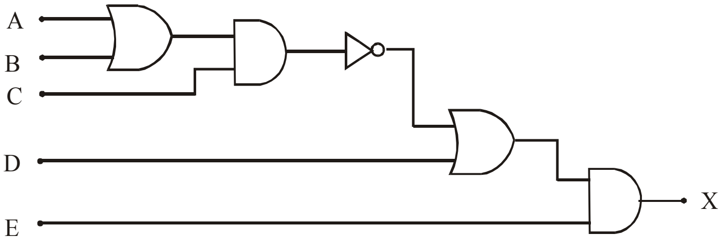


Fig. 3-64.

Solution. Given, the logic circuit containing INVERTER, AND and OR gates of Fig. 3-61.

We know that to determine the Boolean expression for the final output X, we start from left-most inputs and work towards the light as shown in Fig 3-65.

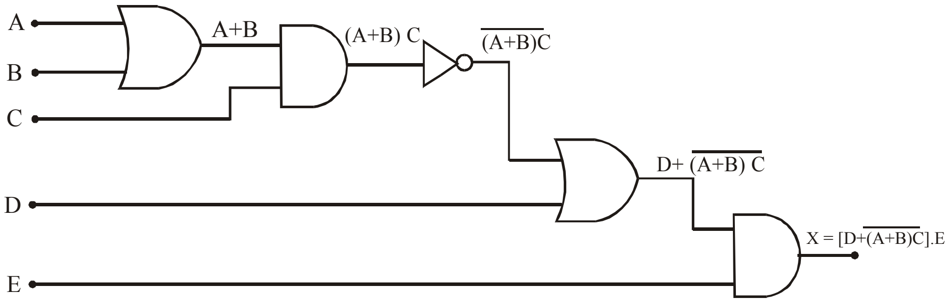


Fig. 3-65.

Writing the Boolean expression for each gate, we find that the Boolean expression for the final output,

$$X=\left[D+\overline{(A+B)C}\right].E \textbf{ Ans.}$$

Example 3-21. *Determine the Boolean expression for output X for the logic circuit shown in Fig. 3-66.*

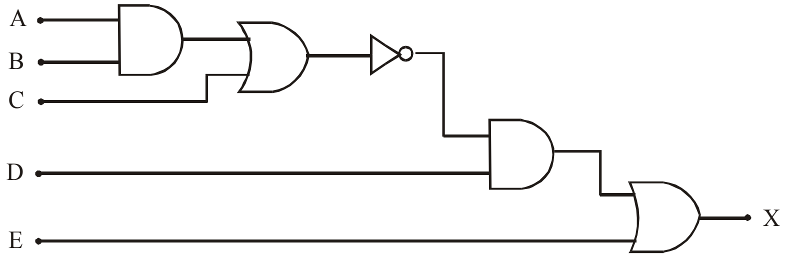


Fig. 3-66.

Solution. Given, the logic circuit containing INVERTER, AND and OR gates of Fig 3-66.

We know that to determine the Boolean expression for the final output X, we start from left-most inputs and work towards the light as shown in Fig. 3-67

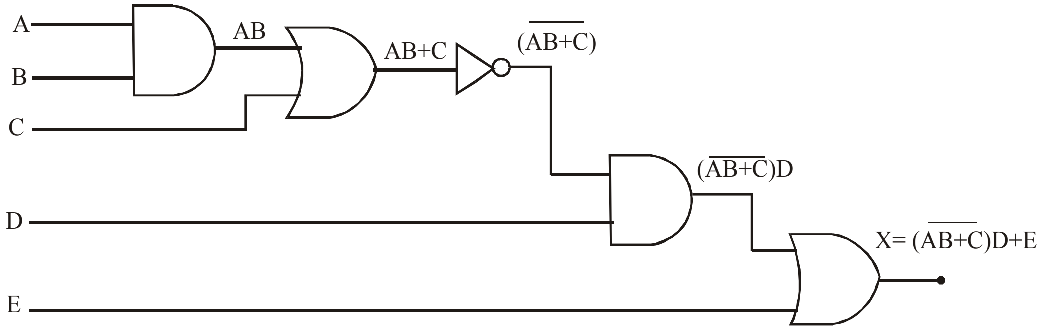


Fig. 3-67.

Writing the Boolean expression for each gate, we find that the Boolean expression for the final output,

$$X=(\overline{AB+C})D+E \quad \textbf{Ans.}$$

3-37. Evaluating the Boolean Expression for the Logic Circuit Output

Once the Boolean expression for a circuit output has been obtained, the output logic level can be determined for any set of input levels. For example, suppose that we want to know the logic level of the output X for the circuit in Fig. 3-68 (a) for the case where $A = 0$, $B = 1$, $C = 1$, and $D = 1$. As in ordinary algebra, the value of X can be found by substituting the values of the variables into the expression and performing the indicated operations as follows:

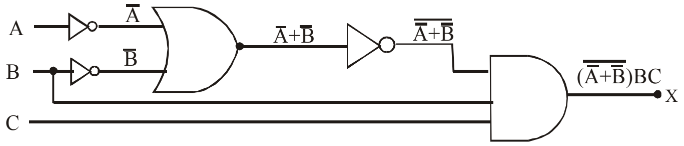

A
B
C
$\overline{A}$
$\overline{B}$
$\overline{A}+\overline{B}$
$\overline{\overline{A}+\overline{B}}$
$(\overline{\overline{A}+\overline{B}})BC$
X

Fig. 3-68.

We know that the Boolean expression for the final output X can be written as,

$$X = (\overline{\overline{A}+\overline{B}})BC \qquad ....(1)$$

Let us suppose we want to know the logic level of the output X for the case where $A = 0$, $B = 1$ and $C = 1$. Substituting the values of the variables A, B, and C into the expression.

$$\begin{aligned} X &= (\overline{\overline{0}+\overline{1}})1.1 \\ &= (\overline{1+0})1.1 \\ &= (\overline{1})1.1 \\ &= 0.1.1 \\ &= 0 \end{aligned}$$

In general, we must follow the following rules when evaluating the Boolean expression of a logic circuit,

1. First, perform all inversions of single terms; i.e., is, $\overline{0} = 1$ or $\overline{1} = 0$.
2. Then perform all operations within parentheses.
3. Perform an AND operation before an OR operation unless parentheses indicate otherwise.
4. If an expression has a bar over it, perform the operations of the expression first and then invert the result.

3-38. Determination of a Truth Table for a Logic Circuit

We have already discussed in the last article the determination and evaluation of a Boolean expression for a given logic circuit. Once we have evaluated a Boolean expression for all the possible combinations of the input variables, we can summarize the results in the form of a truth table.

In order to illustrate the concept, consider a circuit shown in Fig. 3-69. We know that the Boolean expression for this circuit can be written as,

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

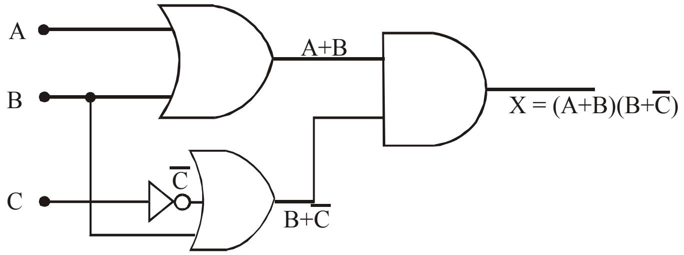


Fig. 3-69.

Let us evaluate this Boolean expression for all the possible values of input variables, A, B, and C.

1. When $A = 0$, $B = 0$, $C = 0$, the final output,

$$X = (A+B)(B+\overline{C}) = (0+0)(0+\overline{0}) = (0+0)(0+1) = 0.1 = 0$$

2. When $A = 0$, $B = 0$, $C = 1$, the final output,

$$X = (A+B)(B+\overline{C}) = (0+0)(0+\overline{1}) = (0+0)(0+0) = 0.0 = 0$$

Similarly, we can consider all the other possible combinations of input variables and determine the logic level of the final output. The results are as shown in a table 3-7.

Table 3-7.

Inputs			*Output*
A	***B***	***C***	***X***
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	1

3-39. Determining Output Logic Level from a Circuit Diagram

As a matter of fact, the output logic level for given input levels can also be determined directly from the circuit diagram *without* using the Boolean expression. This technique is useful for technicians during the troubleshooting or testing of a logic system. It also tells them what each gate output is supposed to be as well as the final output. In order to illustrate this point, let us consider the circuit shown in fig 3-70. The input levels for this circuit are $A = 1$, $B = 0$, $C = 1$, and $D = 0$.

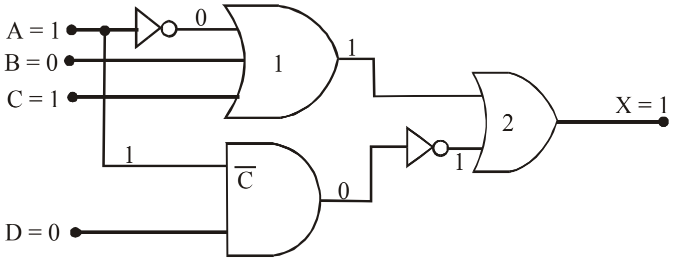


Fig. 3-70. *Determining the output logic from a circuit diagram.*

The procedure is to determine the output logic level of each logic gate while starting from the left-most inputs and work towards the final output.

Let us start from OR gate 1. It has three inputs, 0, 0 and 1. Note that the INVERTER changes the input $A = 1$ to a 0. This condition produces 1 at the OR gate output because $0 + 0 + 1 = 1$. The AND gate has inputs of 1 and 0 which produces a 0 output because $1.0 = 0$. This 0 is inverted to 1 and applied to an OR gate 2 along with the 1 from the OR gate 1 output. The 1 and 1 inputs to the OR gate 2 produces an output of 1 because $1 + 1 = 1$.

Example 3-22. *Determine the output logic level for a digital circuit shown in Fig.* 3-71, *having* $A = 0$, $B = 1$ *and* $C = 1$.

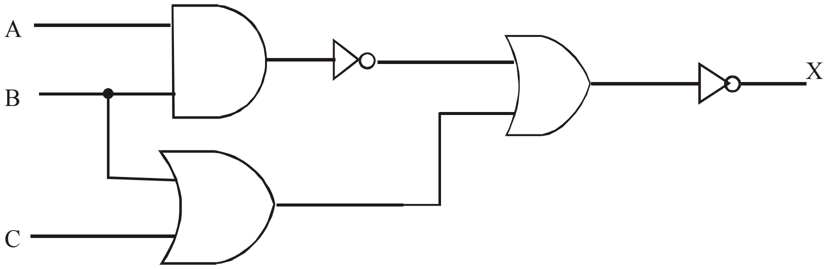


Fig. 3-71.

Solution. Given: The digital circuit shown in Fig. 3-71 with $A = 0$, $B = 1$ and $C = 1$

Let us determine the output logic level of each logic gate while starting from the left-most inputs and work towards the output. The resulting outputs are shown as in Fig. As seen, the final output, $X = 0$ **Ans.**

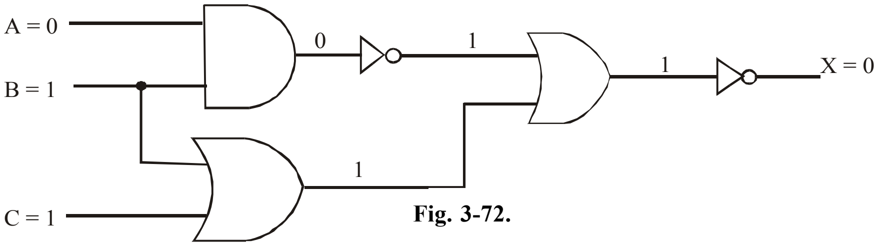


Fig. 3-72.

Example 3-23. *Determine the output logic level for a logic circuit shown in Fig.* 3-73 *when* $A = 1$, $B = 0$, $C = 1$ *and* $D = 1$.

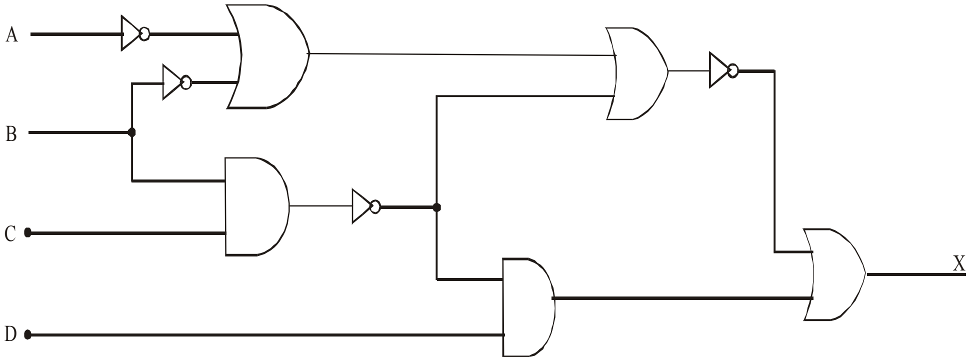


Fig. 3-73.

Solution. Given: The logic circuit shown in Fig. 3-73 with $A = 1$, $B = 0$, $C = 1$ and $D = 1$

Let us determine the output logic level of each logic gate while starting from the left-most inputs and work towards the output. The resulting outputs are as shown in Fig. 3.74. As seen the final output, $X = 1$ **Ans.**

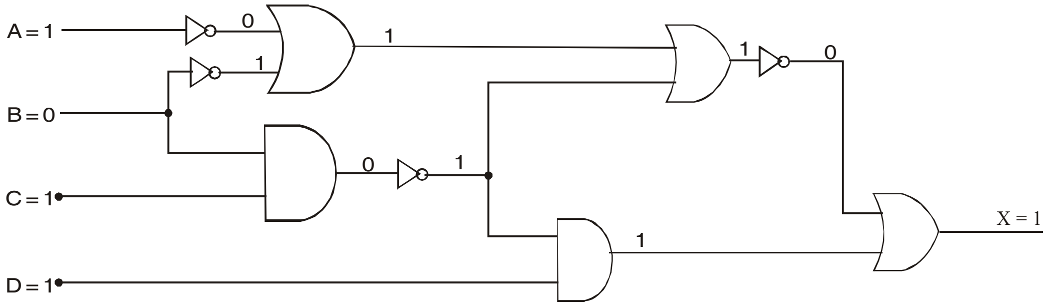
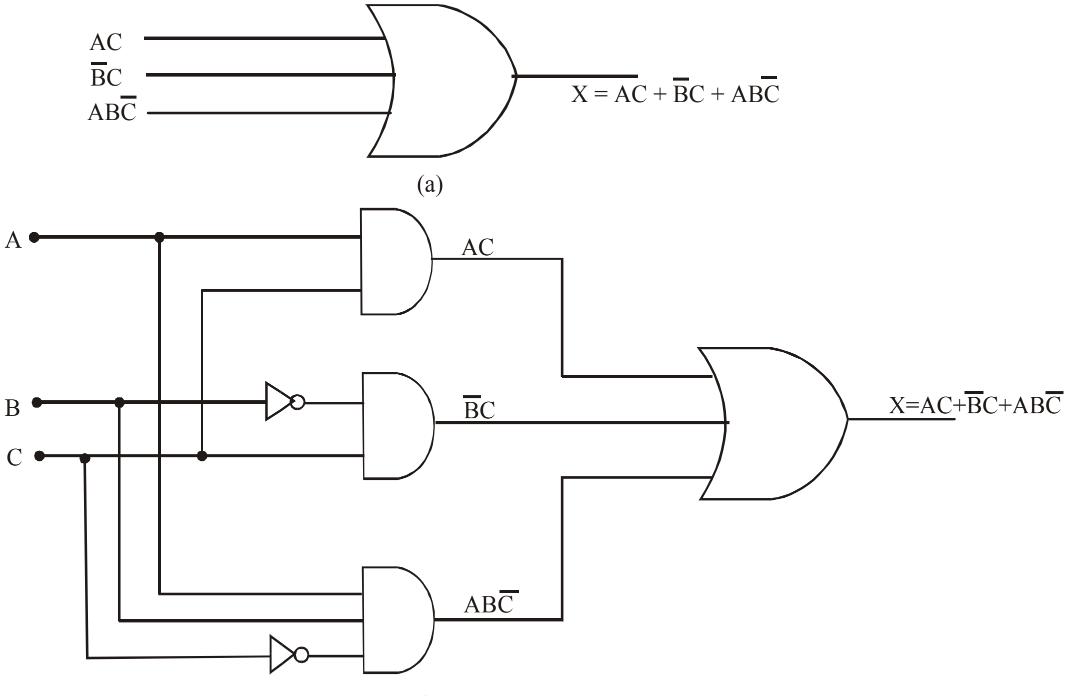


Fig. 3-74.

3-40. Implementing Logic Circuits From Boolean Expressions

It will be interesting to know that if the operation of a circuit is defined by a Boolean expression, a logic circuit can be implemented directly from that expression. For example, suppose we need a circuit that is defined by $X = A.\ B.\ C.$ Then we immediately know that all that is needed is a three input AND gate. If we need a circuit that is defined by $X = A + B$, we will use a two-input OR gate with an INVERTER on one of the inputs. The same reasoning used for these simple cases can be extended to more complex logic circuits.

Now suppose that we want to construct a circuit whose output is $Y = AC + \overline{B}C + AB\overline{C}$. This Boolean expression contains three terms $(AC, \overline{B}C, AB\overline{C})$, which are ORed together. This tells us that a three-input OR gate is required with inputs that are equal to $AC, \overline{B}C,$ and $AB\overline{C}$, respectively. This is illustrated in Fig. 3.75. where a three input OR gate is drawn with inputs labeled as AC, $\overline{B}C$, and $AB\overline{C}$.


AC
$\overline{B}C$
$AB\overline{C}$
$X = AC + \overline{B}C + AB\overline{C}$
A
B
C
AC
$\overline{B}C$
$AB\overline{C}$
$X=AC+\overline{B}C+AB\overline{C}$

(a)

(b)

Fig. 3-75. *Illustrating the implementation of logic circuits from Boolean expressions.*

Notice that, that each OR gate input is an AND product term, which means that an AND gate with appropriate inputs can be used to generate each of these terms. This is shown in Fig. 3-75. which is the final circuit diagram. Notice the use of INVERTERS to produce the $\overline{B}$ and $\overline{C}$ terms required in the expression.

This general approach can always be followed, although we shall find that there are some other more efficient techniques that can be employed. From now onwards, however, this straightforward method will be used to minimize the number of new logic circuits that are to be studied.

Example 3-24. *Sketch the logic circuit for the Boolean expressions*:

(*a*) $(\overline{A}+\overline{B}).C$ (*b*) $\overline{A}B+\overline{C}$ (*Dip. I.E.T.E., June* 1997)

Solution.

(*a*) Given: The Boolean expression,

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

In order to implement this expression as a logic circuit, we find that it requires three input variables (*A, B* and *C*). The term $\overline{A}$ is obtained by using an INVERTER with *A* as an input as shown in Fig. 3-76. Similarly, the term $\overline{B}$ is obtained by using another INVERTER with *B* as its input. Next the term $(\overline{A}+\overline{B})$ is obtained by ORing the outputs of two INVERTERs. Finally an AND gate is required to produce $(\overline{A}+\overline{B})$ *C*. Notice the AND gate inputs are $(\overline{A}+\overline{B})$ and *C*.

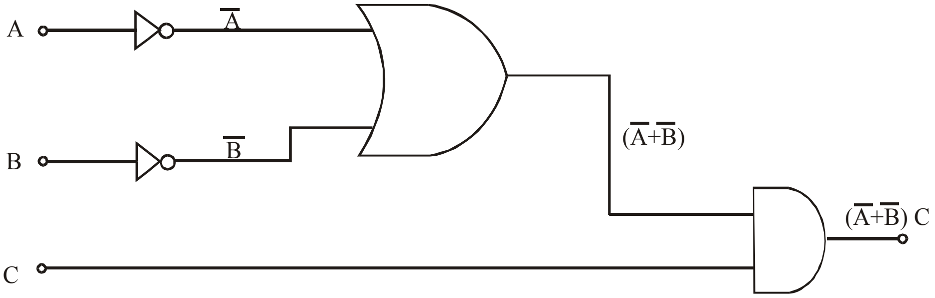

A
$\overline{A}$
B
$\overline{B}$
C
$(\overline{A}+\overline{B})$
$(\overline{A}+\overline{B})$ C

Fig. 3-76.

(*b*) Given: The Boolean expression,

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

Proceeding in the same way as in part (a), we can get $\overline{A}$ and $\overline{C}$ by using INVERTERs. The term $\overline{A}B$ is obtained by using an AND gate with $\overline{A}$ and B as its inputs. Finally we use an OR gate to produce $(\overline{A}B+\overline{C})$ with $\overline{A}B$ and $\overline{C}$ as its inputs. The complete logic circuit is shown in Fig 3-77.

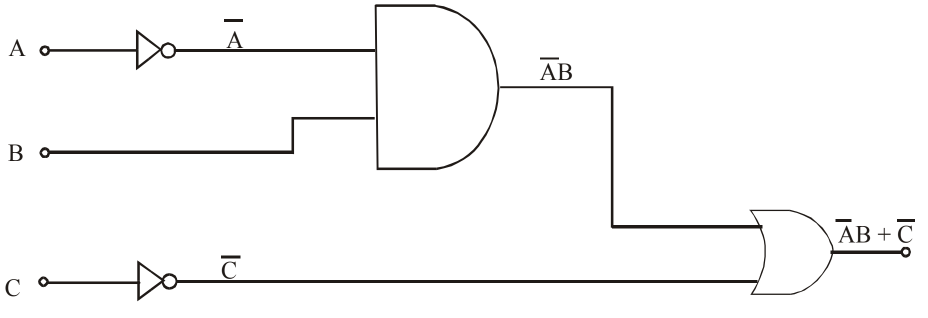


Fig. 3-77.

Example 3-25. *Construct a logic circuit that can implement the Boolean expression*:

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

(*Anna University, Nov./Dec. 2005*)

Solution. Given : The Boolean expression is,

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

In order to implement this expression as a logic circuit, we find that it requires two input variables, A and B. The $\overline{A}$ and $\overline{B}$ are obtained by using INVERTERs and the terms $A\overline{B}$ and $\overline{A}B$ by using two AND gates as shown in Fig 3-78. Finally the output X, is obtained by using an OR gate with ($\overline{A}B$) and ($A\overline{B}$) as its inputs.

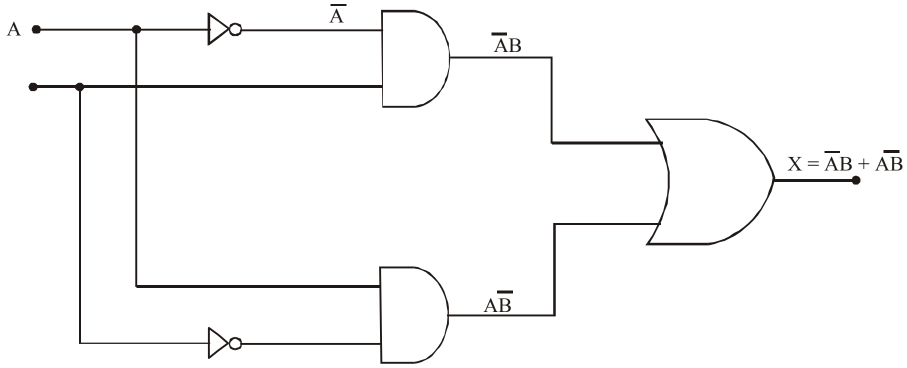


Fig. 3-78.

Example 3-26. *Construct a logic circuit that can implement the Boolean expression*:

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

Solution. Given: The Boolean expression,

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

In order to implement this expression as a logic circuit, we find that it requires two input variables, A and B. The $\overline{A}$ and $\overline{B}$ are obtained by using two INVERTERs and the terms $\overline{A}\overline{B}$ and AB are obtained by using AND gates as shown in the Fig 3.79. Finally, the output X is obtained by using an OR gate with $(\overline{A}\overline{B})$ and (AB) as its inputs.

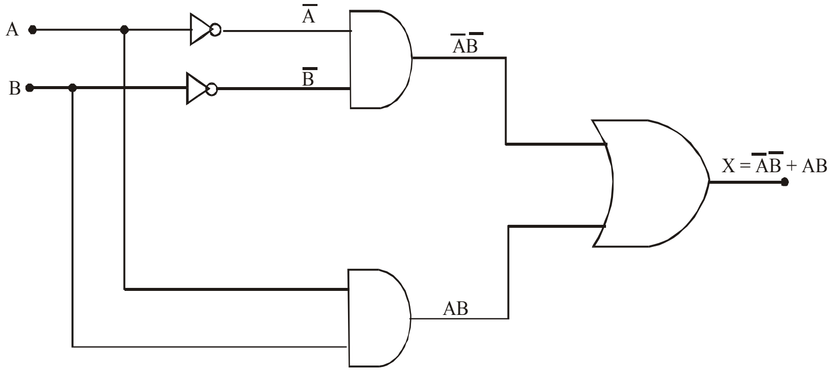

A
B
$\overline{A}$
$\overline{B}$
$\overline{A}\overline{B}$
$X = \overline{A}\overline{B} + AB$
AB

Fig. 3-79.

Example 3-27. *Construct a logic circuit that can implement the Boolean expression.*

$$X = AB + \overline{B}C$$

Solution. Given: The Boolean expression,

$$X = AB + \overline{B}C$$

In order to implement this expression as a logic circuit, we find that it requires three variables A, B and C. The $\overline{B}$ is obtained by using an INVERTER and the terms AB and $\overline{B}C$ are obtained by using AND gates as shown in Fig. 3-80. Finally, the output X is obtained by using an OR gate with (AB) and ($\overline{B}C$) as its inputs.

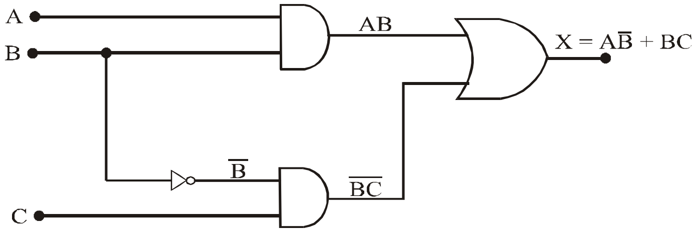


Fig. 3-80.

Example 3-28. *Construct a logic circuit using INVERTER, AND and OR gates for the Boolean expression,*

$$X = \overline{W + P\overline{Q}}$$

Solution. Given: The Boolean expression,

$$X = \overline{W + P\overline{Q}}$$

In order to implement this expression as a logic circuit, we find that it requires three variables W, P and Q. The $\overline{Q}$ is obtained by using an INVERTER, $P\overline{Q}$ by using an AND gates. The term $W + P\overline{Q}$ is obtained by using an OR gate with W and $P\overline{Q}$ as its inputs. Finally the output X is obtained by using an INVERTER at the output of OR gate as shown in Fig. 3-81.

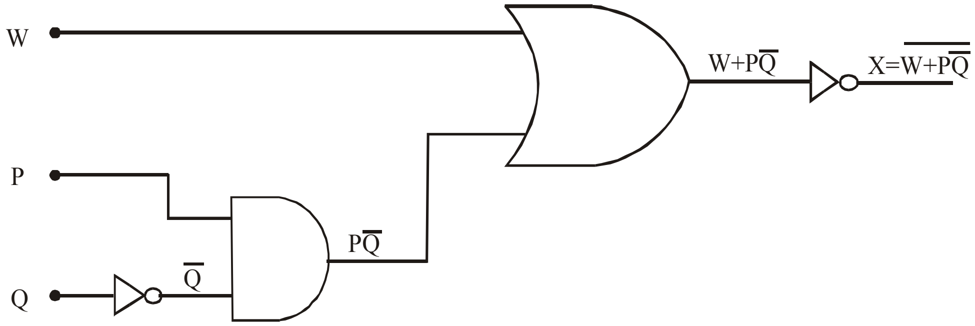


Fig. 3-81.

Example 3-29. *Construct a logic circuit using INVERTER, AND and OR gates for the Boolean expression,*

$$X = MN(P + \overline{N})$$

Solution. Given: The Boolean expression,

$$X = MN\,(P + \overline{N})$$

In order to implement the expression as a logic circuit, we find that it requires three variables *M*, *N* and *P*. The $\overline{N}$ is obtained by using an INVERTER. The term $(P+\overline{N})$ is obtained by using an OR gate with *P* and $\overline{N}$ as its inputs. The term *MN* is obtained by using an AND gate with *M* and *N* as its inputs. Finally, the output *X* is obtained by using an AND gate with (*MN*) and $(P+\overline{N})$ as its inputs.

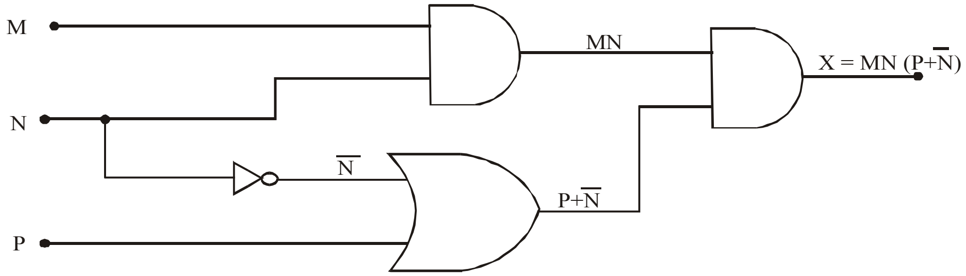


Fig. 3-82.

Example 3-30. *Construct a logic circuit using INVERTER, AND and OR gates for the Boolean expression,*

$$X = \overline{AB(C+D)}$$

Solution. Given: The Boolean expression,

$$X=\overline{AB\,(C+D)}$$

In order to implement this expression as a logic circuit, we find that it requires four varibles, *A*, *B*, *C* and *D*. The term (*AB*) is obtained by using an AND gate with *A* and *B* as its inputs. The term (*C* + *D*) is obtained by using an OR gate with *C* and *D* as its inputs. Next, the term *AB* (*C* + *D*) is obtained by using an AND gate with (*AB*) and (*C* + *D*) with its inputs. Finally, the output *X* is obtained by using an INVERTER as shown in Fig. 3-83.

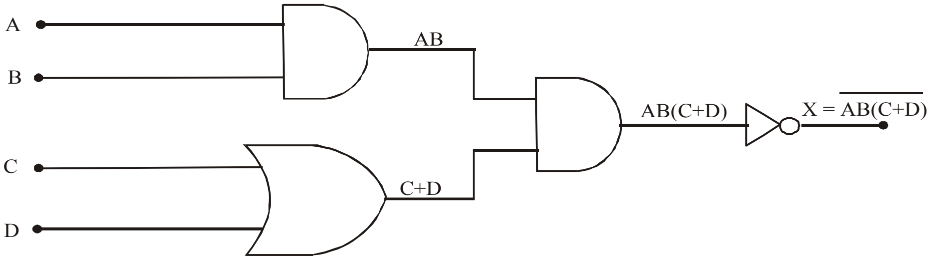


Fig. 3-83.

Example 3-31. *Construct a logic circuit using INVERTER, AND and OR gates for the Boolean expression,*

$$X = (\overline{A+B+\overline{C}D\overline{E}})+\overline{B}C\overline{D}$$

Solution. Given: The Boolean expression,

$$X=(\overline{A+B+\overline{C}D\overline{E}})+\overline{B}C\overline{D}$$

In order to implement this expression as a logic circuit, we find that it requires five variables *A*, *B*, *C*, *D* and *E*. The $\overline{B}, \overline{C}, \overline{D}$ and $\overline{E}$ are obtained by using INVERTERS. The term *A* + *B* is obtained by using an OR gate with A and B as its inputs. The term $\overline{C}D\overline{E}$ is obtained by using an AND gate with $\overline{C}, D\,and\,\overline{E}$ as its inputs. The term $\overline{B}C\overline{D}$ is obtained by using another 3-input AND gate with

$\overline{B}$, C and $\overline{D}$ as its inputs. Next the term $\overline{A+B+\overline{C}D\overline{E}}$ is obtained by using an INVERTER, Finally, the output, X is obtained by using an OR gate as shown in Fig. 3-84.

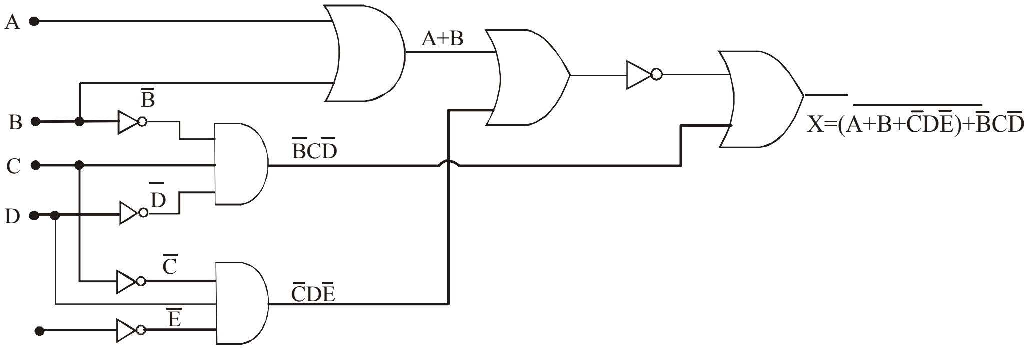

A
B
C
D
A+B
$\overline{B}$
$\overline{B}C\overline{D}$
$\overline{D}$
$\overline{C}$
$\overline{C}D\overline{E}$
$\overline{E}$
$X=\overline{(A+B+\overline{C}D\overline{E})}+\overline{B}C\overline{D}$

Fig. 3-84.

Example 3-32. I*mplement the following function using a quad 2 input NOR gates*: $(\overline{A}B + C).\overline{D}$

(*Anna University, Nov./Dec. 2005*)

Solution. Given: The Boolean expression,

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

In order to implement this expression as a logic circuit, we find that it requires four input variable, A, B, C and D. The $\overline{A}$ is obtained by using INVERTER and the term $\overline{A}B$ is obtained by AND gates. The term $\overline{A}B + C$ is obtained by using OR gate. Finally ,the output X is obtained by using OR gate with $(\overline{A}B + C)$ and D as its input as shown in Fig. 3.85

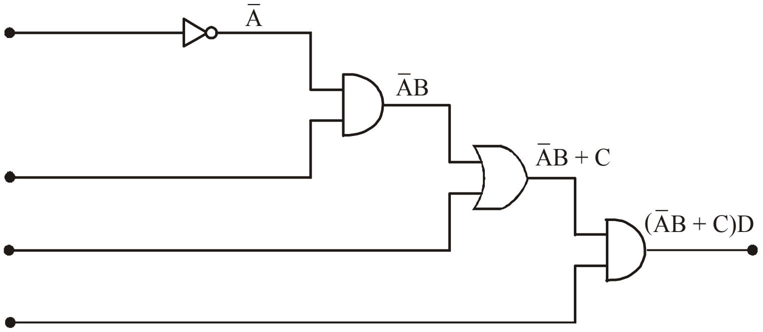


Fig. 3-85

Example 3-33. *Draw the logic circuit for the expression* $F = \overline{x}\,\overline{y}z + \overline{x}yz + x\overline{y}$

(*PTU., May 2009*)

Solution. Given: The Boolean expression.

$$F = \overline{x}\,\overline{y}z + \overline{x}yz + x\overline{y}$$

In order to implement this expression as a logic circuit, we find that it requires three variable x, y and z. The term $\overline{x}$ and $\overline{y}$ an obtained by using three inverter. The terms $\overline{xy}z$, $\overline{x}yz$ and $\overline{x}y$ are obtained by using AND gates as shown in the fig 3.86. Finally the output f is obtained by using an OR gate with $\overline{xy}z$, $\overline{x}yz$ and $\overline{x}y$ as its inputs.

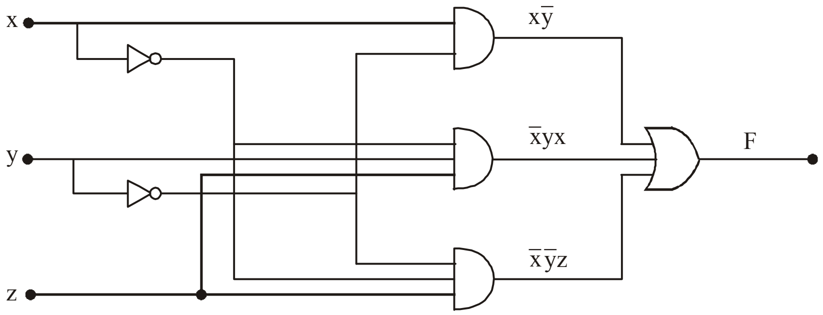


Fig. 3.86

Example 3-34. *Draw the logic circuit for the expression* $F = \bar{A}B + A\bar{B}\bar{C}$

(*PTU., Dec 2009*)

Solution. Gives: the Boolean expression $F = \bar{A}B + A\bar{B}\bar{C}$

In order to implement the expression as a logic circuit, we find that it requires three variable A, B and C. The term $\bar{A}$, $\bar{B}$ and $\bar{C}$ an obtained by using three inverters. The term $\bar{A}B$ and $A\bar{B}\bar{C}$ an obtained by using AND gates as shown in fig 3.87 finally output f is obtained by using an OR gate with $\bar{A}BC$ and $A\bar{B}\bar{C}$ as its inputs.

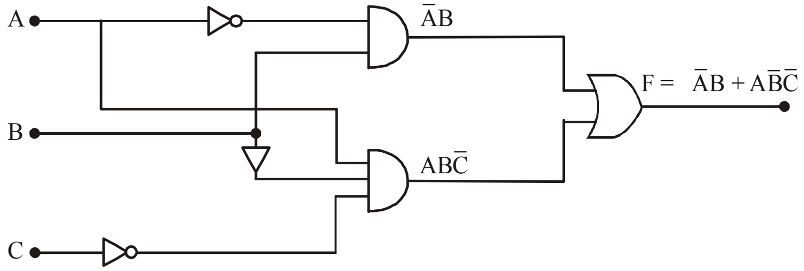


Fig. 3.87

Example 3-35. *Draw the logic diagram for* $X = AB + \bar{B}C$

(*Anna University, Nov./Dec. 2007*)

Solution. Given: The Boolean expression

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

In order to implement this expression as a logic circuit, we find that it requires three input variable, A, B and C. The $\bar{B}$ is obtained by using INVERTER and the terms AB and $\bar{B}C$ are obtained by using AND gate as shown is fig 3.88 finally, the output X is obtained by using and OR gate with AB and BC as its inputs.

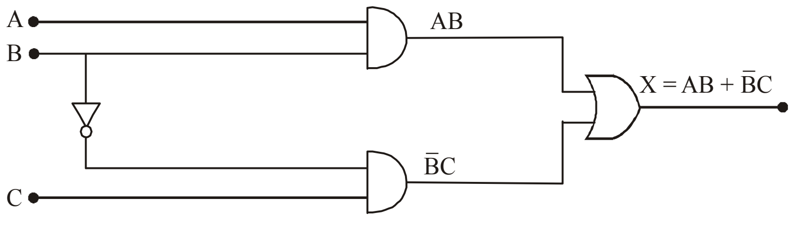


Fig. 3.88

Example 3-36. *Construct the truth table for* $F = x\bar{y} + \bar{x}y$

(*PTU., May 2009, Assam Engineering College, 2006*)

Solution. *Given:* the expression $F = x\overline{y} + \overline{x}y$

We know that for two variable x and y the truth table is

Input		*output*
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.89. *Truth Table.*

3-41. The Universality of NAND and NOR Gates

The universality of NAND gates means that it can be used as an INVERTER and that combinations of NAND gates can be used to implement the AND, OR and NOR operations. Similarly, the NOR gate can be used to implement the INVERTER, AND, OR and NAND operations. Now we will study one by one in the following pages as how the NAND and NOR gates can be used as universal gates.

3-42. The NAND Gate a Universal Logic Gate

The NAND gate is a universal logic gate because it can be used to implement the INVERTER (or NOT), the AND and the OR operation.

An INVERTER can be made from a NAND gate by connecting all of its inputs together and creating, in effect, a single input as shown in Fig 3-90(a) for a 2-input gate. In this configuration, the NAND simply acts as INVERTER because its output is,

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

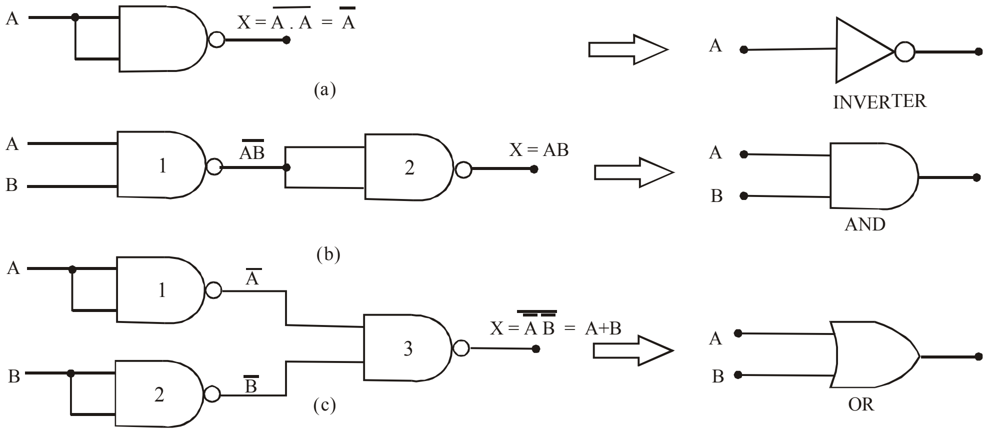

A
$X = \overline{A\,.\,A} = \overline{A}$
(a)
A
INVERTER
A
B
1
$\overline{AB}$
2
X = AB
A
B
AND
(b)
A
1
$\overline{A}$
B
2
$\overline{B}$
3
$X = \overline{\overline{A}\,\overline{B}} = A+B$
(c)
A
B
OR

Fig. 3-90.

An AND operation can be performed by connecting 2 NAND gates as shown in Fig 3-90(b). Here the NAND gate 2 is used as an INVERTER so that the final output,

$$X = \overline{\overline{AB}} = AB$$

This is the desired AND function.

An OR operation can be performed by connecting 3 NAND gates as shown in Fig 3-90(c), Here NAND gates 1 and 2 are used as INVERTERs to invert the inputs, so that the final output is,

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

This is the desired OR operation.

Using DeMorgan's theorem (explained in chapter 4), it is possible to show that

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

$$\therefore \quad X = A+B$$

3-43. The NOR Gate as a Universal Gate

Like the NAND gate, the NOR gate can also be used to implement the INVERTER (or NOT), the AND and the OR operation.

An INVERTER can be made from a NOR gate by connecting all of its inputs together and creating, in effect, a single input as shown in Fig 3-91(a). for a 2-input gate. In this configuration, the NOR gate simply acts as an INVERTER because its output is,

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

An OR operation can be performed by connecting a 2 NOR gates as shown in Fig. 3-91(b), Here the NOR gate 2 is used as an INVERTER, so that the final output,

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

An AND operation can be performed by connecting 3 NOR gates as shown in Fig 3-91(c). Here the NOR gates 1 and 2 are used as INVERTERs to invert the inputs, so that the final output is,

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

Using DeMorgan's theorem (explained in chapter 4), it is possible to show that,

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

$$\therefore \quad X = AB$$

This is the desired AND function.

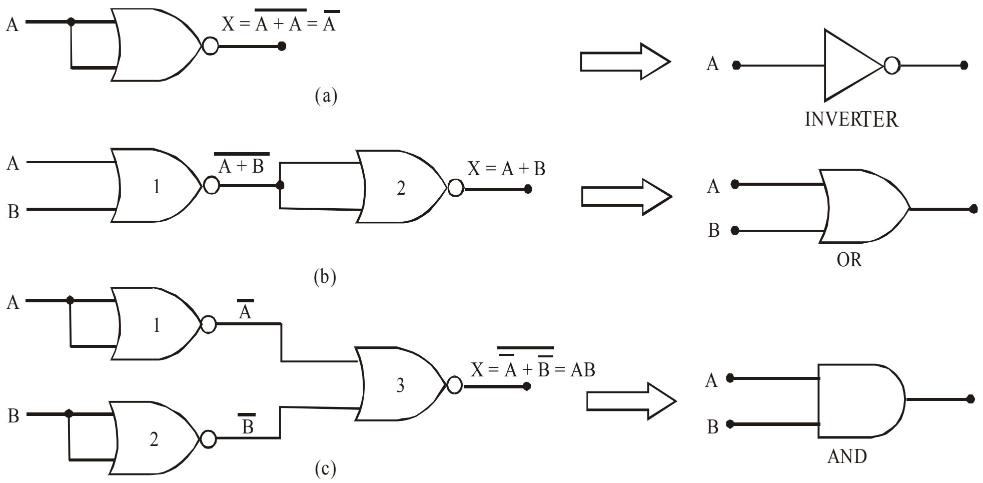


Fig. 3-91.

Example 3-37. *Sketch a NAND-NAND logic circuit for the Boolean equation,*

$$y = A\overline{B} + AC + BD$$

(*Nagpur University, Oct/Nov. 1997*)

Solution. Given: The Boolean equation,

$$y = A\overline{B} + AC + BD$$

Let us first of all, implement this circuit using the basic logic gates, i.e. using AND, OR and INVERTER gates. The $\overline{B}$ is obtained by INVERTing B and $A\overline{B}$ by ANDing the variables A and $\overline{B}$. Similarly AC is obtained by ANDing variables A and C. Next the term BD is obtained by ANDing the variables B and D Finally, the terms $A\overline{B}$, AC and BD are ORed to get Y output. Fig 3-92 shows the logic circuit for the given Boolean expression using basic logic gates.

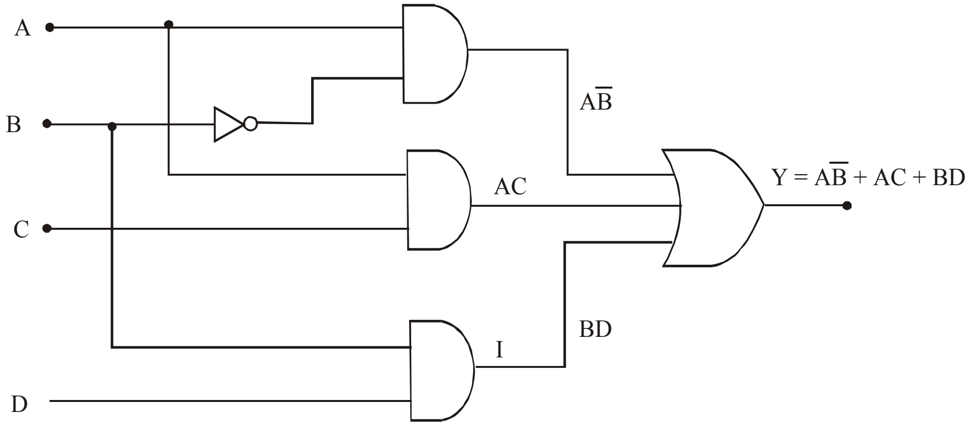
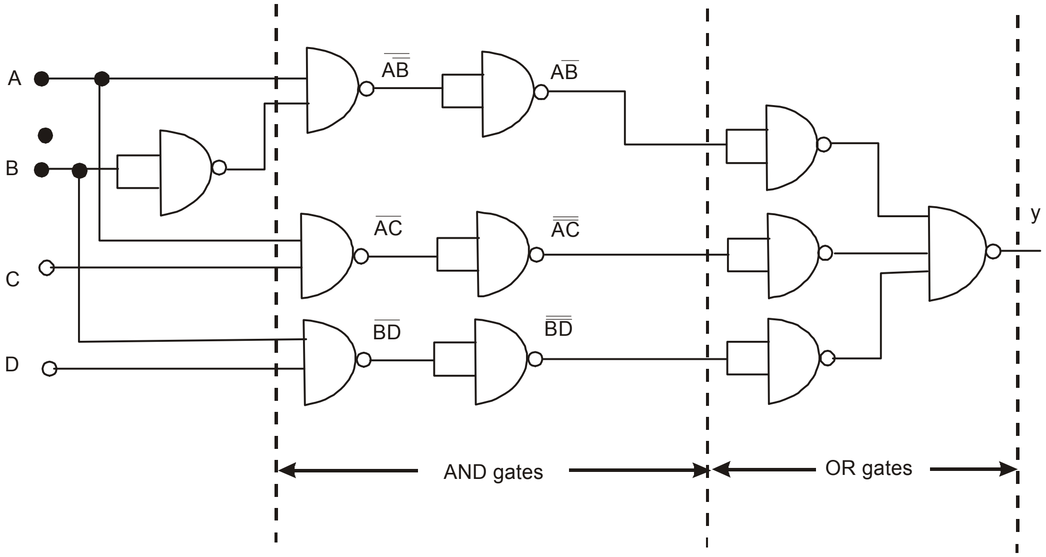


Fig. 3-92.

Now we can replace each logic gate by its NAND gate equivalent. Notice that the INVERTER is replaced by a single NAND gate each AND gate is replaced by two NAND gates while an OR gate is replaced by three NAND gates. The resulting circuit is as shown in Fig 3-93.



A
B
C
D
$\overline{A\overline{B}}$
$A\overline{B}$
$\overline{AC}$
$\overline{\overline{AC}}$
$\overline{BD}$
$\overline{\overline{BD}}$
y
AND gates
OR gates

Fig. 3-93.

Removing the two INVERTERs, on the same line we can simplify the circuit shown in Fig 3-93, to the one shown in Fig 3-94.

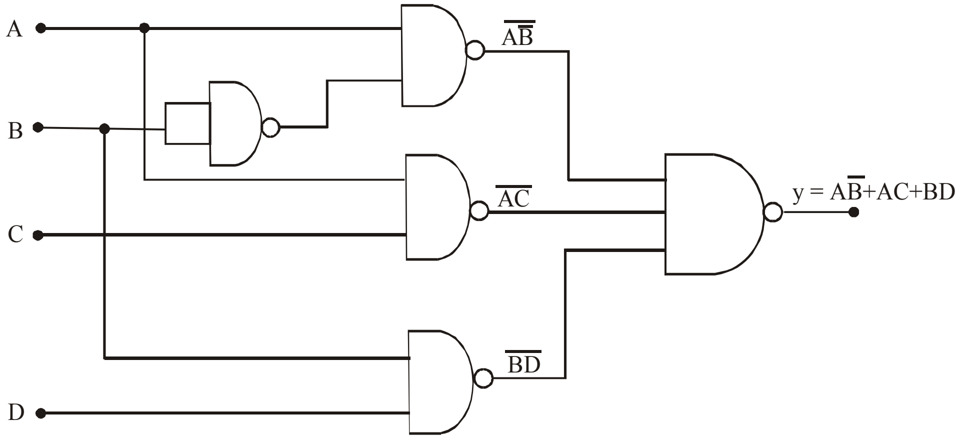


Fig. 3-94.

Example 3-38. *Using graphical procedure, obtained a NOR gate realization of the Boolean expression.*

$$F\ (a,\ b,\ c,\ d)\ =\ \bar{a}d + a\bar{d} + (b + \bar{c})$$

Also explain the steps to be followed in the said procedure.

(*VTU., July/Aug. 2004 Jan/Feb. 2006*)

Solution. Given: The Boolean expression.

$$F\ (a,\ b,\ c,\ d) = \bar{a}d + a\bar{d} + (b + \bar{c})$$

Let us first of all, implement this circuit using the basic logic gates, i.e using AND OR and INVERTER gates. The $\bar{a}$, $\bar{c}$, and $\bar{d}$ is obtained by INVERTERING a, c and d. The term $a\bar{d}$ by ANDING the variable a and $\bar{d}$. Next the term (b + c) is obtained by ORed variable b and $\bar{c}$. ANDING the variable ad and (b + c) we get the term $a\bar{d}$ (b + c). finally the output is obtained by ORed the variable $\bar{a}d$, and $a\bar{d}$(b + $\bar{c}$) as shown in fig. 3.95.

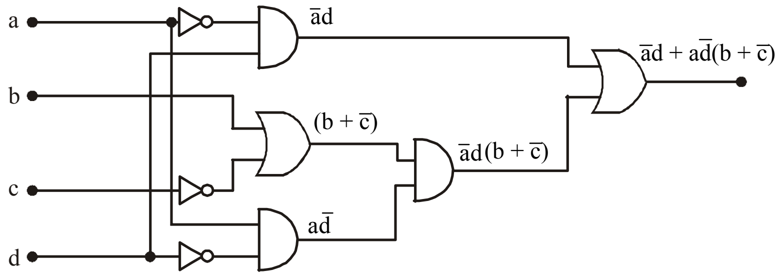


Fig. 3-95

Now we can replaced each logic gate by its NOR gate equivalent. Notice that the INVERTER is replaced by a single NOR gate each AND gate is replaced by three NOR gates while an OR gate is replaced by two NOR gates as shown in Fig 3-96.

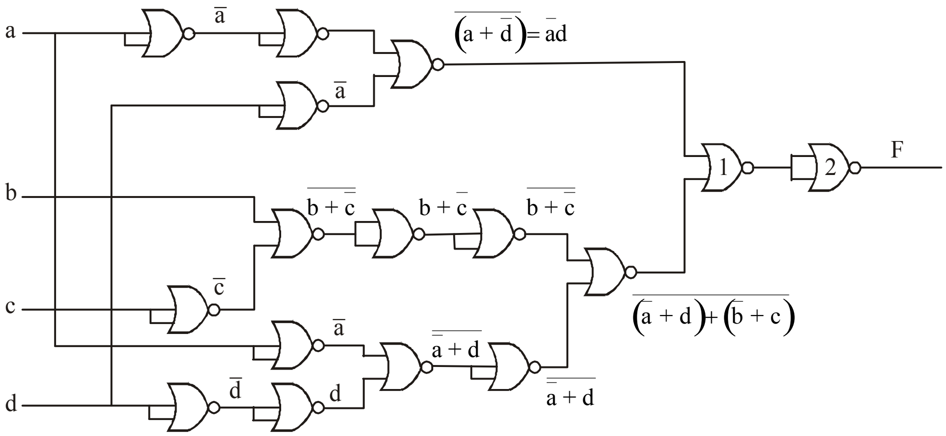


Fig. 3-96

Output at gate No 1 is $\overline{(\bar{a}d) + [(\bar{a}+d) + (\overline{b+\bar{c}})]}$ and the output at gate no 2

$\overline{(\bar{a}d) + [(\bar{a}+d) + (\overline{b+\bar{c}})]} = \bar{a}d + a\bar{d}\,(b+\bar{c})$. Remaining the two inverter on the same line we can simplify the circuit as shown in Fig. 3.97.

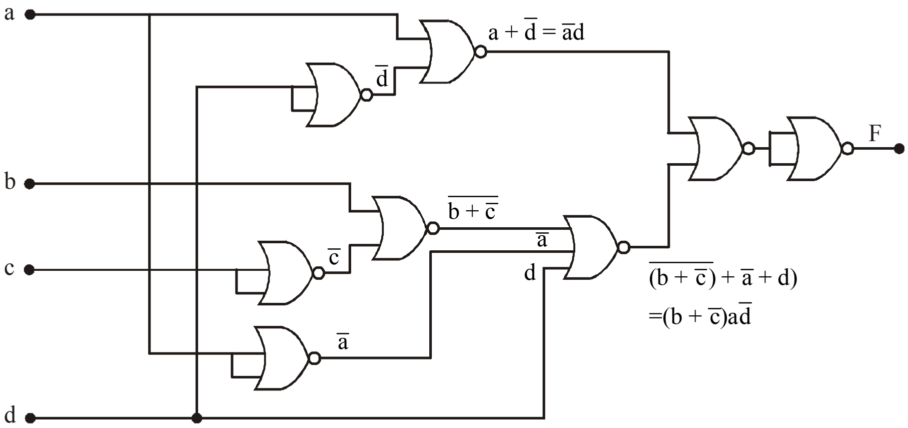


Fig. 3-97

Example 3.39. *$F = (A\bar{B} + \bar{A}B)\,(C + \bar{D})$ with only NOR get.*

(*Anna University, Nov./Dec.2007*)

Solution. *Given:* The Boolean equation,

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

Let us first of all implement this circuit using the basic logic gates i.e. using AND, OR and INVERTER gates Fig. 3-98 shows the logic circuit for the Boolean expression using basic logic gates

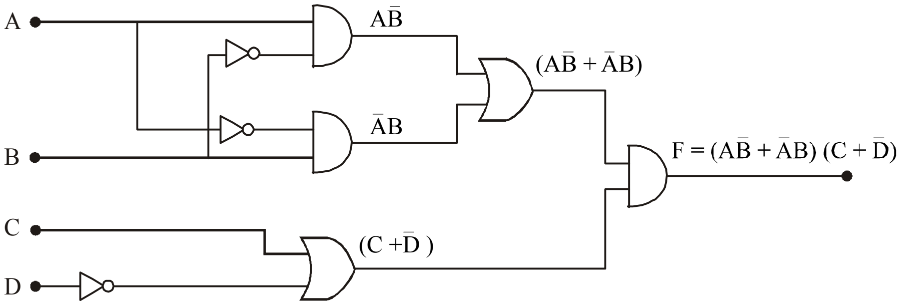


Fig. 3-98

Now we can replace each logic gate by its NOR gate equivalent Notice that INVERTER is replaced by single NOR. While AND is replace by three NOR and OR gate is replace by two NOR gate. The resulting circuit is shown in Fig 3-99.

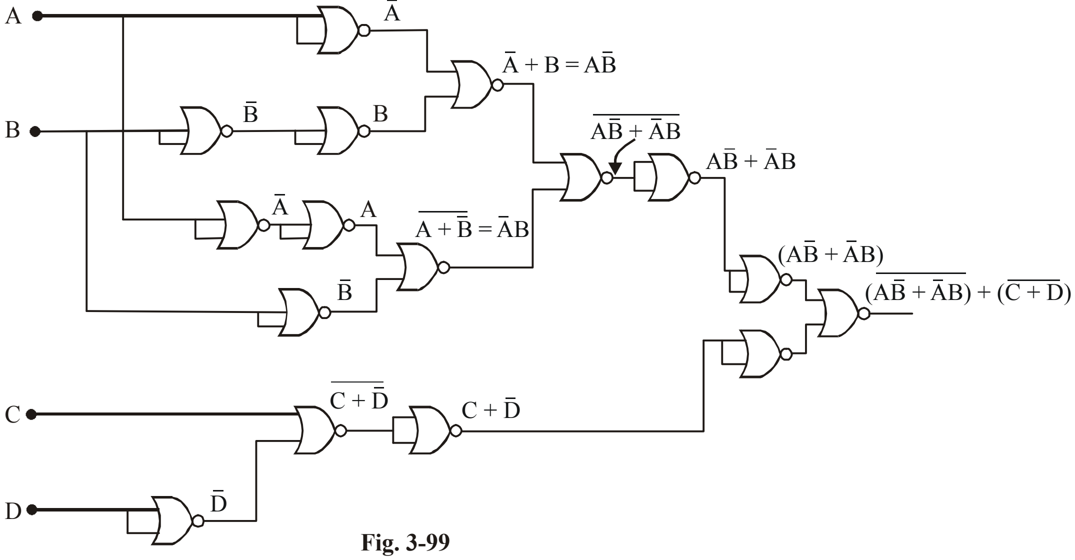

A
B
C
D
$\bar{A}$
$\overline{\bar{A} + B} = A\bar{B}$
$\bar{B}$
B
$\overline{A\bar{B} + \bar{A}B}$
$A\bar{B} + \bar{A}B$
$\bar{A}$
A
$\overline{A + \bar{B}} = \bar{A}B$
$\bar{B}$
$(A\bar{B} + \bar{A}B)$
$\overline{(A\bar{B} + \bar{A}B) + (\overline{C + D})}$
$\overline{C + \bar{D}}$
$C + \bar{D}$
$\bar{D}$

Fig. 3-99

Removing the two INVERTER on the same line we can simplify the circuit. The resulting circuit is shown in Fig. 3-100

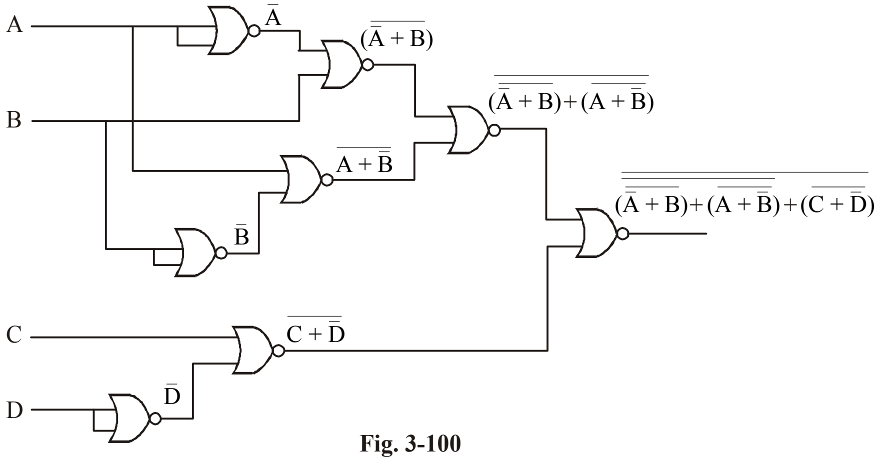

A
B
C
D
$\bar{A}$
$\overline{(\bar{A} + B)}$
$\overline{\overline{(\bar{A} + B)} + \overline{(A + \bar{B})}}$
$\overline{A + \bar{B}}$
$\bar{B}$
$\overline{\overline{\overline{(\bar{A} + B)} + \overline{(A + \bar{B})}} + \overline{(C + \bar{D})}}$
$\overline{C + \bar{D}}$
$\bar{D}$

Fig. 3-100

Example 3-40. *Obtain a NAND gate realization of the Boolean expression*

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

Solution. Given: the Boolean Expresion

$$f(\mathrm{A,B,C}) = (\mathrm{A} + \overline{\mathrm{B}} + \mathrm{C})\,(\overline{\mathrm{A}} + \overline{\mathrm{B}} + \overline{\mathrm{C}})\,(\overline{\mathrm{A}} + \mathrm{B})$$

Let us first of all implement the circuit using the basic logic gates i.e. using AND, OR and INVERTER gates as shown in Fig 3-10.

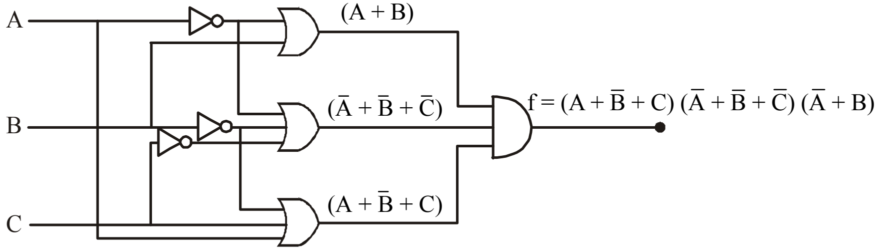

A
B
C
$(\overline{A} + B)$
$(\overline{A} + \overline{B} + \overline{C})$
$(A + \overline{B} + C)$
$f = (A + \overline{B} + C)\,(\overline{A} + \overline{B} + \overline{C})\,(\overline{A} + B)$

Fig. 3-101

Now we can replace each logic by its NAND gate equivalent. Notice that the INVERTER is replaced by a single NAND gate each AND gate is replaced by two NAND gates while on OR gate is replaced by three NAND gates. The resulting circuit is as shown in Fig 3-102.

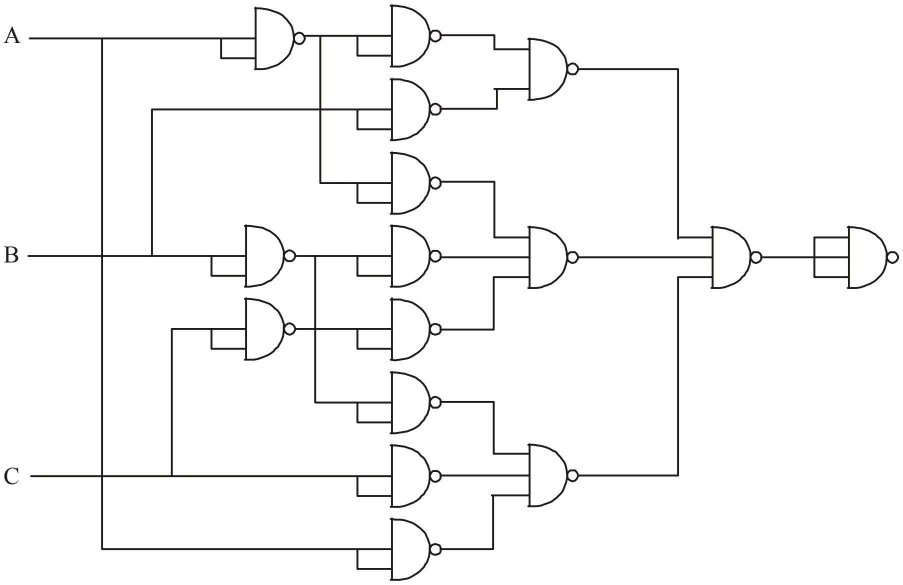


Fig. 3-102

Example 3-41. *With sketch the expression X = AB + CD using.*

(*i*) *NAND gate only*

(*ii*) *NOR gate only*

(*Gujrat Technological University., Dec 2009*)

Solution. *Given:* The Boolean equation,

$$\mathrm{X} = \mathrm{AB} + \mathrm{CD}$$

Let us first of all, implement this circuit using the basic logic gates, i.e using AND and OR gates. The term AB and CD are obtained by using AND gate. The final output X is obtained by using OR gate as shown in fig. 3.103.

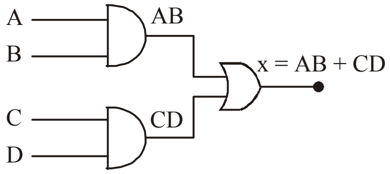


Fig. 3-103

(*i*) NAND gate only

Realizing the expression using NAND gates, we replace the each logic gate by it NAND gate equivalent AND gate is replaced by two NAND gate while the OR gate is replaced by three NAND gates. The resulting circuit is as shown in Fig 3-104.

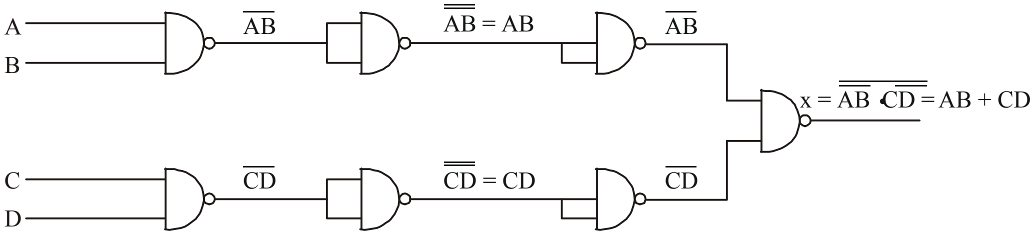

A
B
$\overline{AB}$
$\overline{\overline{AB}} = AB$
$\overline{AB}$
x = $\overline{\overline{AB}\cdot\overline{CD}}$ = AB + CD
C
D
$\overline{CD}$
$\overline{\overline{CD}} = CD$
$\overline{CD}$

Fig. 3-104

Removing the two gates on the same line we can simplicity the circuit shown in Fig 3-105.

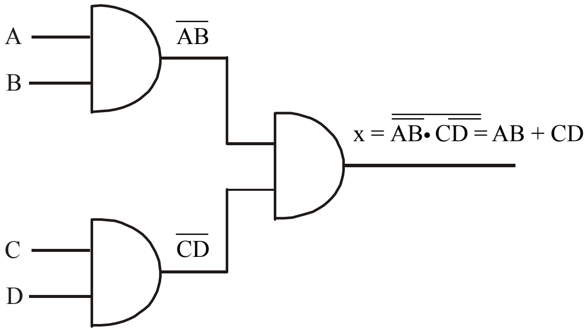

A
B
$\overline{AB}$
x = $\overline{\overline{AB}}\cdot\overline{\overline{CD}}$ = AB + CD
C
D
$\overline{CD}$

Fig. 3-105

(*ii*) NOR Gate only

We can replace each logic gate of the Fig 3-105 by its NOR gate equivalent. Each AND gates are replaced by three NOR gate and OR gate are replaced by two NOR gate. The resulting circuit is shown in Fig 3-106.

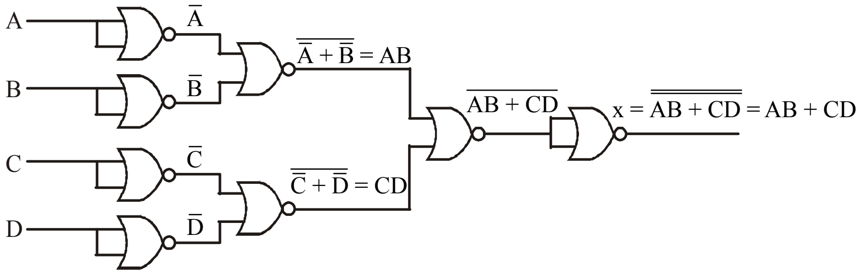

A
$\overline{A}$
B
$\overline{B}$
$\overline{\overline{A}+\overline{B}} = AB$
$\overline{AB + CD}$
x = $\overline{\overline{AB + CD}}$ = AB + CD
C
$\overline{C}$
$\overline{\overline{C}+\overline{D}} = CD$
D
$\overline{D}$

Fig. 3-106

Example 3-42. *Simplify the expression F=AB + AC + ABC (AB + C). Implement using minimum number of NAND gates.*

(*PTU, Dec. 2009*)

Solution. *Given:* The Boolean expression.

$$\begin{aligned} F &= AB + AC + ABC\,(AB + C) \\ &= AB + AC + ABC + ABC \\ &= AB + AC + ABC \\ &= AB + AC\,(1 + B) \end{aligned}$$

Applying Rule 4. (chapter No 4.) to the term $1 + B$ we get

$$= AB + AC$$

Let us first of all implement this circuit using the basic logic gates, i.e. using AND and OR gates. The terms AB and AC are obtained by AND gates. The output function F is obtained by OR gate. Fig 3-107 shown the logic circuit for the given Boolean expression using basic logic gates.

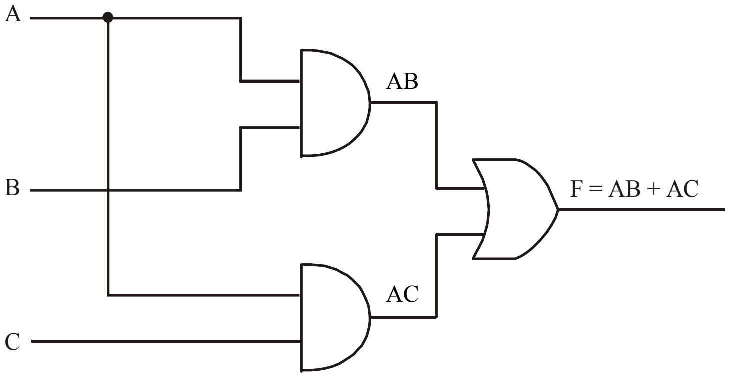


Fig. 3-107

Now we can replace each logic gate by its NAND gate equivalent. Notice that the AND gates is replaced by two NAND gates while an OR gates is replaced by three NAND gates. The resulting circuit is as shown in Fig 3-108.

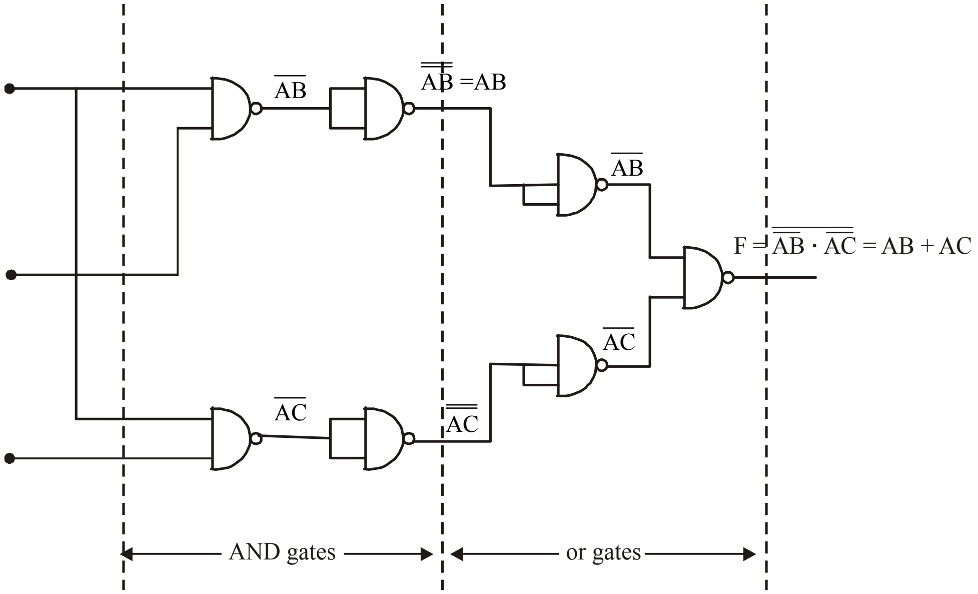

$\overline{AB}$
$\overline{\overline{AB}}$ =AB
$\overline{AB}$
F = $\overline{\overline{AB} \cdot \overline{AC}}$ = AB + AC
$\overline{AC}$
$\overline{AC}$
$\overline{\overline{AC}}$
AND gates
or gates

Fig. 3-108

Removing the two NAND gate on the some line. We can simplify the circuit shown in Fig 3-109.

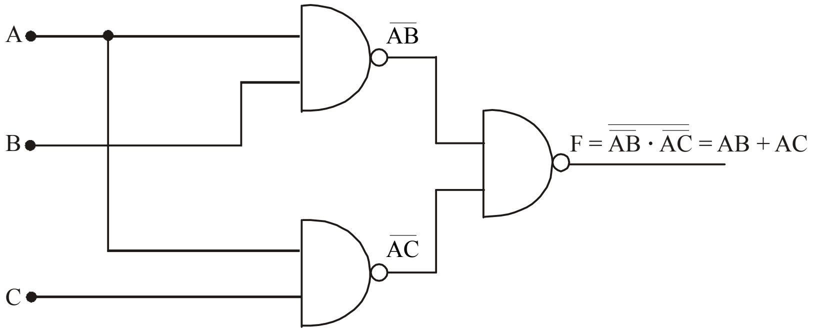

A
B
C
$\overline{AB}$
$\overline{AC}$
$F = \overline{\overline{AB} \cdot \overline{AC}} = AB + AC$

Fig. 3-109

Example 3-43. *In the combinational logic circuit shown in Fig* 3-110., *the input square wave is phase-inverted at the output when the control input is 1. It is passed on as it is when the control input is zero. Determine the logic gate used in the given figure to achieve the desired operation.*

(*Civil Services Elect. Engg.* 1996)

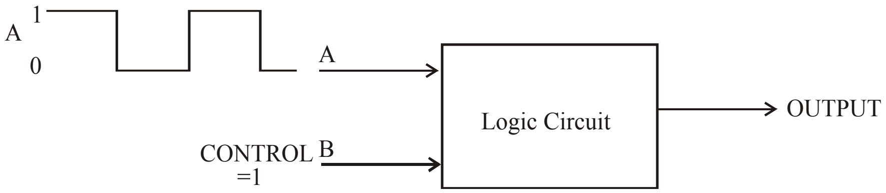


Fig. 3-110.

Solution. Given: A logic circuit with a square waveform at one input and a logic HIGH at the other input.

To determine what logic gate is used to achieve the desired operation, we will have to examine the logic gate available to us. These are the AND gate, OR gate, NAND gate, NOR gate, XOR gate and XNOR gate. Since the question says the squares waveform is inverted at the output when the control input is 1, and it is passed as it is when the control input is zero, we will have to rule out the possibility of an AND gate and an OR gate because these logic gates don't invert the waveforms. Let us continue with the remaining four logic gates i.e., NAND gate, NOR gate, XOR gate and XNOR gate.

We know that the Boolean expression for a NAND gate is given by the expression,

$$X = \overline{AB}$$

In the present situation B is the control input. Now if B = 1, then $X = \overline{A.1} = \overline{A}$, i.e. the square wave input is inverted at the NAND gate output. But if B = 0, then $X = \overline{A.0} = \overline{0} = 1$. This means the circuit will produce HIGH output, if control input B is 0, which is not a desired function. Therefore, the logic circuit with in the box of a given circuit is **not** a NAND gate.

Let us not examine the Boolean expression for a NOR gate. We know that the Boolean expression for a NOR gate is,

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

If $B = 1,$ then $X = \overline{A+1} = \overline{1} = 0$

If $B = 0,$ than $X = \overline{A+0} = \overline{A}$

So we find that if control input (B) is 1, the circuit output is 0 (LOW) and if control input (B) is 0, the circuit output is the opposite of the input i.e. the inverted square waveform. But this is again not a desired operation. Therefore we *rule out* the possibility of a NOR gate also.

Let us now examine the Boolean expression for an XOR gate. We know that Boolean expression for an XOR gate is,

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

If $B = 1$, then $X = A.\overline{1} + \overline{A}.1 = A.0 + \overline{A} = \overline{A}$

If $B = 0$, then $X = A.\overline{0} + \overline{A}.0 = A.1 + 0 = A$

So we find that if the logic circuit makes use of an XOR gate, then if the control input B is 1, the logic gate output is opposite to the input A i.e. the output will be an inverted square wave. On the other hand if B is 0, the logic gate output is the same as input A i.e. the square waveform is passed as it is. This is the desired operation of the given logic circuit. Therefore the logic gate which we need is an XOR gate. **Ans.**

3-44 Tristate Gates

In digital electronics **Three-state**, **Tri-State**, or **3-state logic** allows an output port of assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit. This allows multiple circuit to share the same output line or lines (such as a bus).

Tristate gates have three possible output states logic '1' state, the logic '0' state, and a high impedance state. High impedance state is controlled by an ENABLE input. The ENABLE input decides whether the gate is active or in high impedance state. When active it can be '0' or '1' depending upon input conditions. One of the main advantages of these gates is that their inputs and outputs can be connected in parallel to a common bus line. Fig. 3-11 shows the circuit symbol of a tristate NAND gate with active HIGH ENABLE input, its truth table is shown in Table 3-8. Here Z is high impedance state.

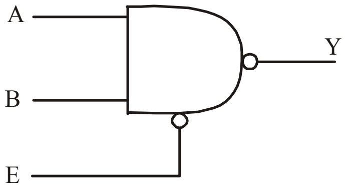


Fig. 3-111

Table 3-8

A	*B*	*E*	*Y*
0	0	0	Z
0	0	1	1
0	1	0	Z
0	1	1	1
1	0	0	Z
1	0	1	1
1	1	0	Z
1	1	1	0

Fig. 3-112 shows the NAND gate with active LOW ENABLE input.. its truth table is shown in Table 3-9. Here Z is high impedance state.

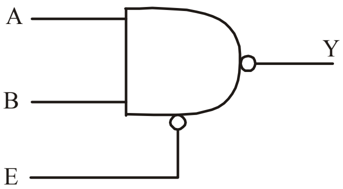


Fig. 3-112

Table 3-9

A	*B*	*E*	*Y*
0	0	0	1
0	0	1	Z
0	1	0	1
0	1	1	Z
1	0	0	1
1	0	1	Z
1	1	0	0
1	1	1	Z

When tristate devices are paralleled then only one of them is enabled at a time. Fig. 3-113 shows a paralleling of tristate inverters having HIGH enable inputs.

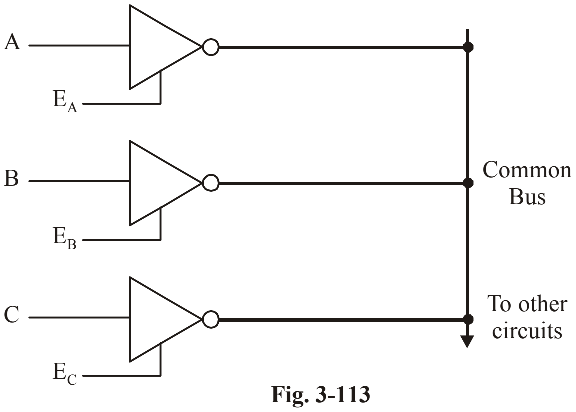
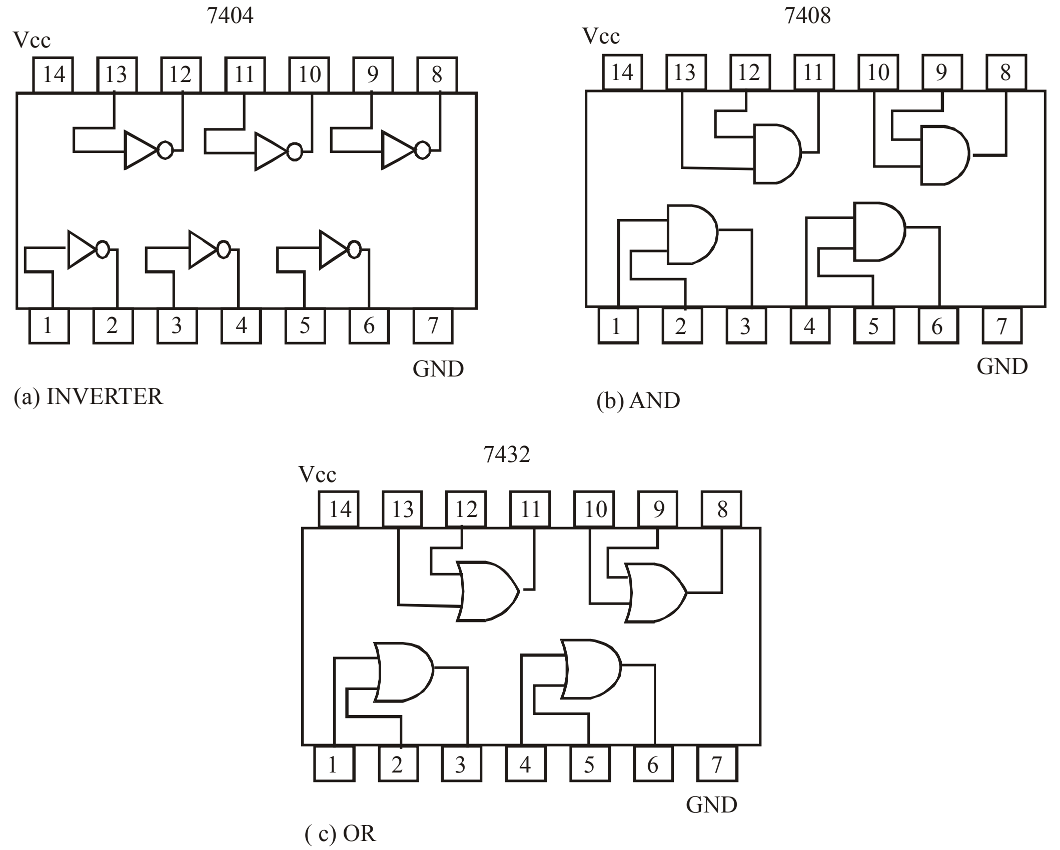


Fig. 3-113

3-45. Using Integrated Circuit Logic Gates

As a matter of fact, all the logic gates (which have been discussed in this chapter) are available in various configurations in TTL (Transistor-Transistor Logic) and CMOS (Complementary Metal-Oxide Semiconductor) Logic families. Thus TTL IC 7404 and CMOS IC 74C04 are hex (six) INVERTERs. The TTL IC 7400 and CMOS IC 74 C00 are quad (four) two-input NAND gates. The TTL IC 7402 and CMOS IC 74C 02 are quad two-input NOR gates. Other popular NAND and NOR gates are available in three-; four – and eight – input configurations. The TTL IC 7486 and CMOS IC 74C86 all quad Exclusive – OR (XOR) gates. Similarly the TTL IC 74266 and CMOS IC 74C266 are quad Exclusive-NOR (XNOR) gates.

You can consult a TTL or CMOS data manual for the availability and pin configuration of these ICs. The pin configurations for the TTL hex INVERTER, the quad AND and the quad OR gates are given in Fig 3-114. Similarly the pin configurations for the TTL quad NAND, quad NOR, gates are shown in Fig 3-115.



(a) INVERTER

(b) AND

(c) OR

Fig. 3-114.

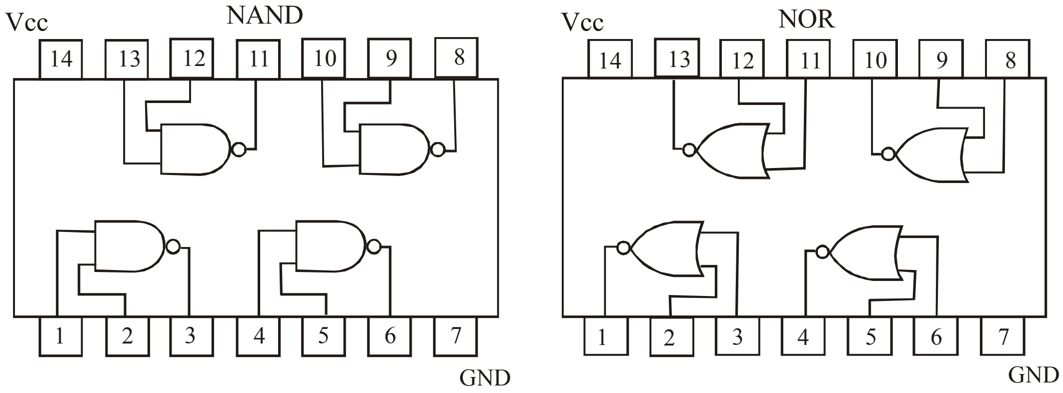


Fig. 3-115.

SUMMARY

In this chapter you have learnt that :

1. Boolean algebra is a powerful mathematic tool used in design and analysis of digital/logic circuits.
2. The INVERTER, AND and OR are the basic Boolean operations.
3. An INVERTER (or a NOT) gate produces an output that is the opposite logic level as the input. An AND gate produces a HIGH output when all inputs are HIGH. An OR gate produces a HIGH output when any input is HIGH.
4. A NAND gate is the same as an AND gate with its output connected to an INVERTER. A NOR gate is the same as an OR gate with its output connected to an INVERTER.
5. NAND gates can be used to implement any of the basic Boolean expression. Similarly NOR gates can also be used for this purpose.
6. An Exclusive –OR gate has the Boolean expression, $X = \overline{A}B + A\overline{B}$. Its output X will be HIGH only when its inputs *A* and *B* are at opposite logic levels.
7. An Exclusive-NOR gate has the Boolean expression, $X = \overline{A}\overline{B} + AB.$ Its output X will be HIGH only when its inputs *A* and *B* are at same logic levels.

DESCRIPTIVE QUESTIONS

1. If you were to build a truth table for an eight-input AND gate, how many different combinations of inputs would you have? Sketch the table with these input combinations. Indicate the output level for each input combination.
2. Show how a two-input NOR gate can be constructed from a two-input NAND gate.
3. Show how a two-input NAND gate can be constructed from a two-input NOR gate.
4. Describe the truth table for a 5-input AND gate.
5. Write the output expression for a NAND gate with inputs *A*, *B*, *C* and *D*.
6. Write the output expression for a NOR gate with inputs *A*, *B*, *C* and *D*.
7. Write the output expression for a 3-input NOR gate.
8. How does a NAND gate differs from an AND gate?
9. How does an exclusive–OR gate differ from an OR gate in its logical operation?
10. Suppose that you have an unknown 2-input gate. That is either OR gate or an AND gate. What combinations of input levels should you apply to the gate's inputs to determine which type of gate it is?
11. Draw the symbol of NOR gate. Write its truth table, logic equation and logic operation. Draw and explain its switch equivalent.
    (*Nagpur University, 2008*)
12. Implement EX OR gate using only four two input NAND gates.
    (*Nagpur University, 2008*)
13. Draw the symbol, truth table and the equation of the three basic gates and two universal gates and realize all the five gates using either of the universal gates.
    (*Anna University, May/Jun. 2006*)
14. What are Universal gates? Realize the basic gates using them. (*VTU., Jan. /Aug. 2004*)

15. What is a universal gate? Consider a gate which takes two inputs A and B and produce and output $\overline{A} \cdot B$.

(*VTU, Jan./Feb. 2005*)

16. Would you consider it a universal gate? Discuss.

(*VTU, Jan./Feb. 2005*)

17. Explain the EX-OR function.

(*VTU, Jan./ Feb. 2004*)

18. How the realization of the following
(*i*) NOR using NAND gates
(*ii*) X-NOR using NAND gates

(*VTU, Dec,/Jan. 2008*)

19. Explain the difference between combination circuit realization OR and XOR. Use truth table to describe how these operation differ.

(*PTU. May 2007*)

20. Draw the logic symbol of a XOR gate and give its truth table.

(*Anna University, Nov./Dec. 2007*)

21. Find the relation between the I/Ps and O/P shown, in Fig. 3-116 Name the operation performed.

(*Anna University, Apr./May 2008*)

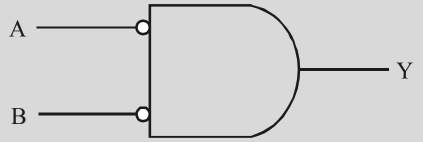


Fig. 3-116.

22. Realize AND, OR, NOT using only NAND gates.

(*GBTU/MTU, 2006-07*)

23. Using NOE gate only realize EX-OR gates.

(*GBTU/MTU, 2006-07*)

24. Realize NOT, OR, AND gates using NOR approach.

(*GBTU/MTU, 2006-07*)

25. Using discrete elements design AND and OR gates.

(*Jamia Millia Islamia University, 2007*)

26. Why NAND and NOR gates are called universal gates? Explain with appropriate example.

(*Jamia Millia Islamia University, 2007*)

27. Is it possible to use a 3 input NAND gate as a 2 input AND gate? If yes, how?

(*Nagpur University, 2004*)

28. lllustrate how a NAND gate can be used to realize the following gates:
1. NOT gate
2. NOR gate

(*RGPV Bhopal. June 2008*)

29. How will you determine NANDgate delay in the laboratory?

(*GBTI/MTU, 2006-07*)

TUTORIAL PROBLEMS

1. Fig. 3-117 shows a waveform applied to the input of an INVERTER. Sketch the output waveform corresponding to the input.

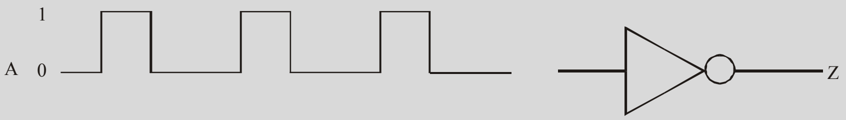
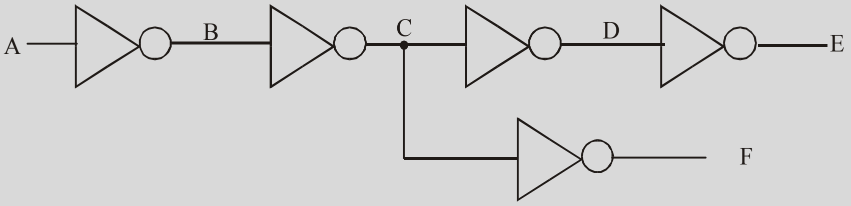


Fig. 3-117.

2. A network of cascaded INVERTERS is shown in Fig. 3-118. If a logic 1 is applied to a point A. determine the logic levels at a points B through F.



(**Ans.** B = 0, C = 1, D = 0, E = 1, F = 0)

Fig. 3-118.

3. Determine the output, X for a 2-input AND gate with the input waveforms shown in Fig. 3-119.

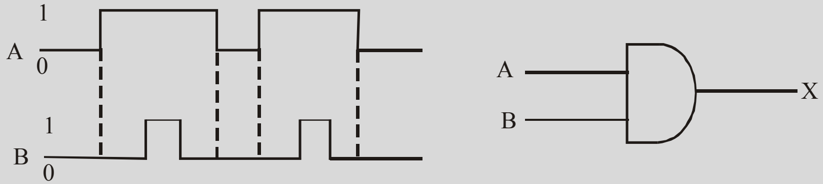


Fig. 3-119.

4. Determine the output, X for a 2-input AND gate with the input waveforms shown in Fig. 3-120.

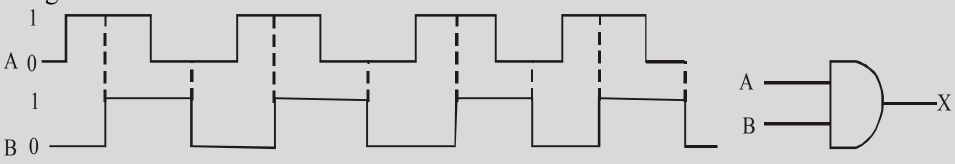


Fig. 3-120.

5. Fig 3-121, shows the input waveforms applied to a 3-input AND gate. Show the output waveform in proper relation to the inputs with a timing diagram.

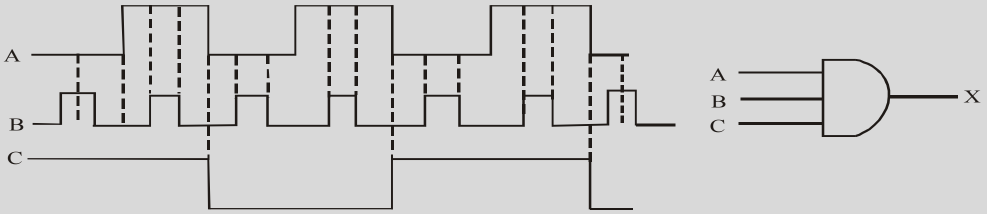


Fig. 3-121.

6. Fig 3-122 shows the input waveforms applied to a 4-input AND gate. Show the output waveform in proper relation to the inputs with a timing diagram.

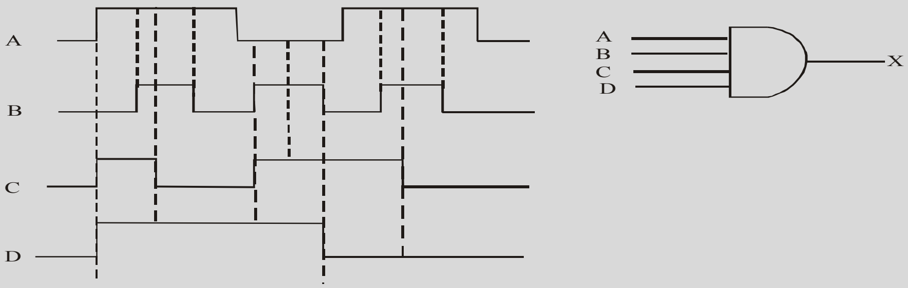


Fig. 3-122.

7. Sketch the output waveform at *X* for the 2-input OR gate shown in Fig. 3-123.

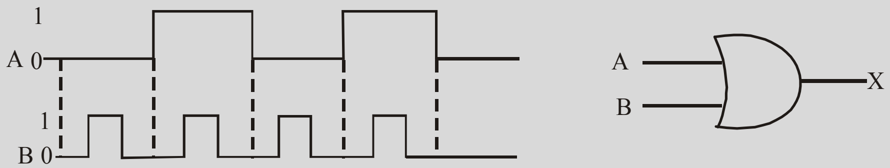


Fig. 3-123.

8. Determine the output, *X* for a 2-input OR gate with the input waveforms shown in Fig. 3-124.

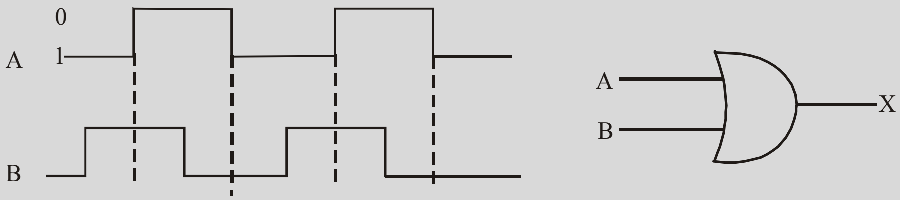


Fig. 3-124.

9. Fig 3-125 shows the input waveform at *A* for a 2-input AND gate. Sketch the input waveform at *B* that will produce the output at *X* as shown in the figure.

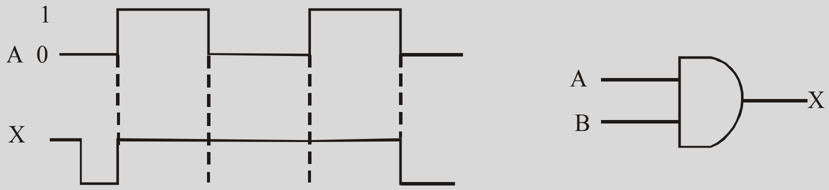


Fig. 3-125.

10. Fig 3-126 shows the input waveform at *A* for a 2-input OR gate. Sketch the input waveform at *B* that will produce the output at *X* as shown in the figure.

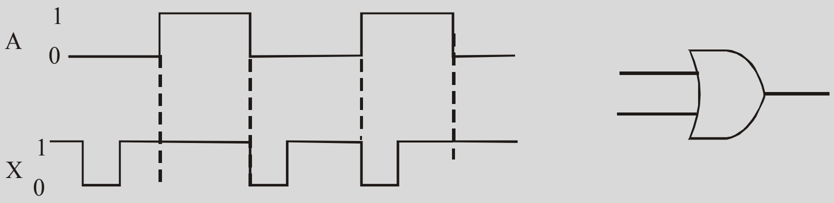


Fig. 3-126.

11. Determine the output for a 2-input OR gate when the input waveforms are as shown in Fig. 3-127 and draw a timing diagram.

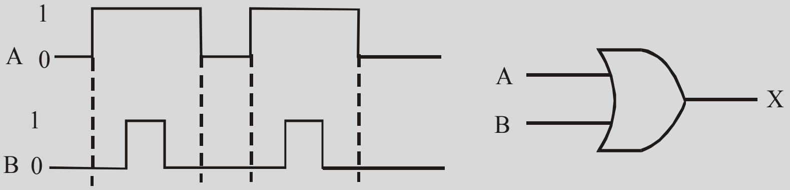


Fig. 3-127.

12. Fig. 3-121 shows the input waveforms applied to a 3-input OR gate. Show the output waveform in proper relation to the inputs with a timing diagram.

13. Fig. 3-122 (Problem 6) shows the input waveforms applied to a 4-input OR gate. Show the output waveform in proper relation to the inputs with a timing diagram.

14. For the set of input waveforms shown in Fig. 3-128 determine the output waveform *X*, for the 2-input NAND gate and draw the timing diagram.

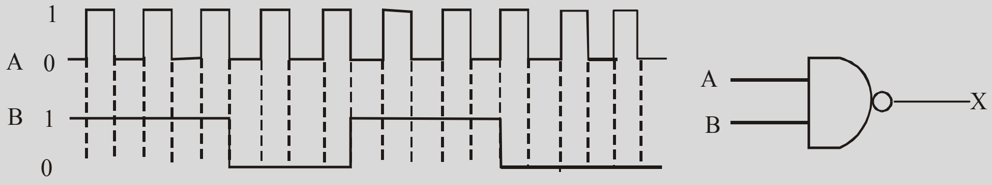


Fig. 3-128.

15. Determine the 3-input NAND gate output for the input waveforms shown in Fig 3-129 and draw the timing diagram.

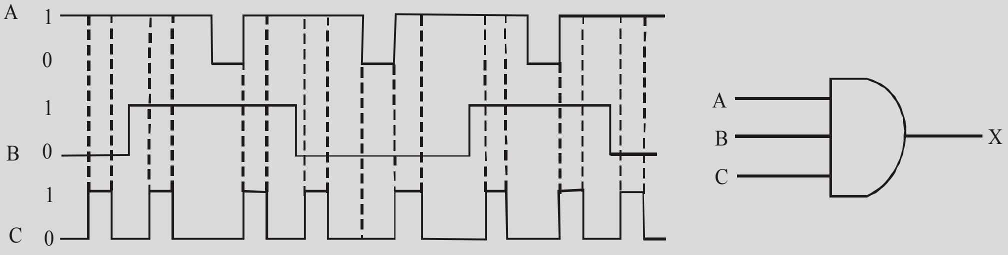


Fig. 3-129.

16. Fig 3-130 shows the waveforms at a 4-input NAND gate. Show the output waveform in proper relation to the inputs with a timing diagram.

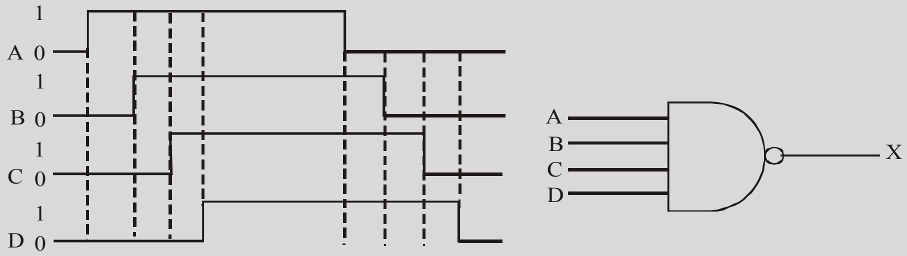


Fig. 3-130.

17. Repeat Problem 12 for a 2-input NOR gate.

18. Repeat Problem 13 for a 3-input NOR gate.

19. Determine the 2-input NOR gate output for the input waveforms shown in Fig 3-131 and draw the timing diagram.

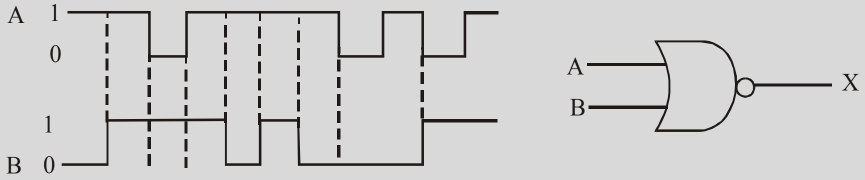


Fig. 3-131.

20. Determine the output of an exclusive –OR gate for the input waveforms shown in Fig. 3-132 and draw the timing diagram.

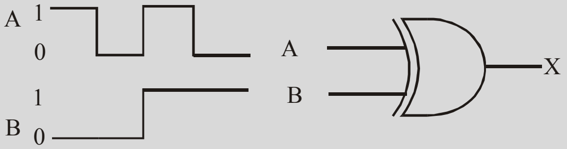


Fig. 3-132.

21. Determine the output of an exclusive – NOR gate for the input waveforms shown in Fig 3-133 and draw the timing diagram.

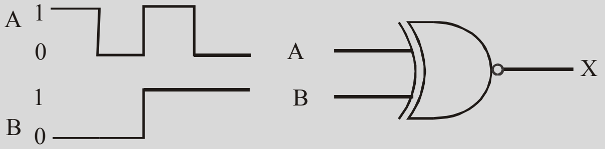


Fig. 3-133.

MULTIPLE CHOICE QUESTIONS

1. The number of table entries needed for a five-input logic circuit is:

(*a*) 4 (*b*) 8 (*c*) 16 (*d*) 32

2. Which of the following input combination will produce a HIGH at the output of a two-input AND gate

(*a*) both inputs are LOW

(*b*) one of the input is HIGH and the other one is LOW

(*c*) both inputs are HIGH

(*d*) none of these

3. The output of a two-input NAND gate is HIGH if

(*a*) both inputs are LOW

(*b*) one of the input is HIGH and the other one is LOW

(*c*) both inputs are HIGH

(*d*) none of these.

4. An exclusive –OR function is expressed as

(*a*) $\bar{A}\bar{B} + AB$ (*b*) $\bar{A}B + A\bar{B}$

(*c*) $(\bar{A} + B)(A + \bar{B})$ (*d*) $(\bar{A} + \bar{B}) + (A + B)$

5. The AND operation can be produced with

(*a*) two NAND gates (*b*) three NAND gates

(*b*) one NOR gate (*d*) three NOR gates

6. The OR operation can be produced with

(*a*) two NOR gates (*b*) three NAND gates

(*c*) four NAND gates (*d*) both (a) & (b)

7. All Boolean expressions can be implemented with

(*a*) NAND gates only

(*b*) NOR gates only

(*c*) combinations of NAND and NOR gates

(*d*) combinations of AND gates, OR gates and INVERTERS

(*e*) any of these.

8. To get an output 1 from circuit of Fig. 3-134. The inputs ABC must be

(*a*) 010 (*b*) 100 (*c*) 101 (*d*) 110

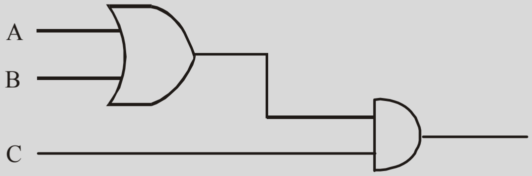
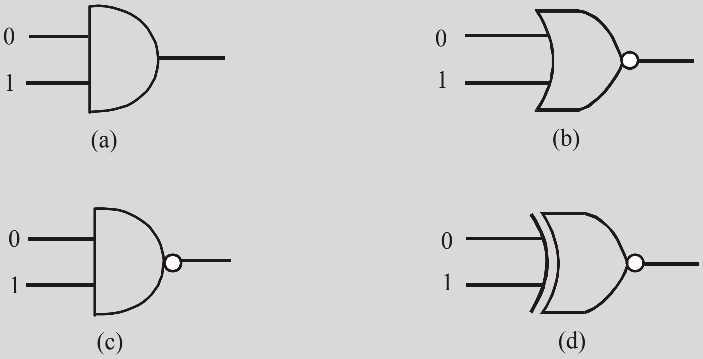


Fig. 3-134.

9. Which of the following logic gates will have an output of 1?



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

Fig. 3-135.

10. The logic gate which produces a 0 or low-level output when one or both of the inputs are 1 is called ....gate.

(*a*) AND (*b*) OR (*c*) NOR (*d*) NAND

ANSWERS

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

4

BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

OUTLINE

Objectives

Upon completion of this chapter, you should be able to:

- Write Boolean equations for combinational logic applications.
- Utilize Boolean Algebra laws and rules for simplifying combinational logic circuits.
- Apply DeMorgan's theorem to complex Boolean equations to arrive at simplified equivalent equations.
- Design single-gate logic circuits by utilizing the universal capability of NAND and NOR gates.
- Troubleshoot combinational logic circuits.
- Implement sum-of-products expressions utilizing AND-OR-INVERT gates.
- Utilize the Karnaugh mapping procedure to systematically reduce complex Boolean equations to their simplest form.
- Describe the steps involved in solving a complete system design application.

4-1. Introduction

Boolean algebra is a convenient and systematic way of expressing and analysing the operation of logic circuits. In the last chapter, we have already introduced the Boolean operations and expressions in terms of their relationships to INVERTER, AND, OR, NAND, NOR, exclusive-OR (XOR) and exclusive-NOR (XNOR) gates.

This chapter covers the laws, rules and theorems of Boolean algebra and their application to digital circuits. We will learn how to simplify the logic circuits using the methods of Boolean algebra and Karnaugh maps.

4-2. Laws and Rules of Boolean Algebra

Boolean algebra is an important mathematical tool for designing and analysing digital systems. A basic knowledge of Boolean Algebra is indispensable to the study and analysis of logic circuits. As in other areas of mathematics, there are certain well developed rules and laws that must be followed in order to apply Boolean Algebra properly. Let us now study some of the important laws and rules of Boolean algebra. First we will study the laws of Boolean algebra and then the rules.

4-3. Laws of Boolean Algebra

Broadly speaking there are 3 basic categories laws of Boolean algebra: (1) the **Cummutative laws,** (2) the **Associative laws** and, (3) the **Distributive laws**. Each of these categories is illustrated with two or three Boolean variables, but the number of Boolean variables is not limited to this.

1. **Cummutative laws.** There are 2 laws under this category- (*i*) Cummutative law of addition and (*ii*) Cummulative law of multiplication.

   (*i*) **Cummutative law of addition.** This law for 2 variables is written algebraically as

$$A+B = B+A \qquad \ldots(3\text{-}1)$$

This law states that the order in which the variables are ORed makes no difference. Remember the addition and the OR operation are the same when applied to logic circuits. Fig 4-1. Illustrates the cummulative law as applied to the OR gate. It shows that it doesn't matter to which input each variable is applied.

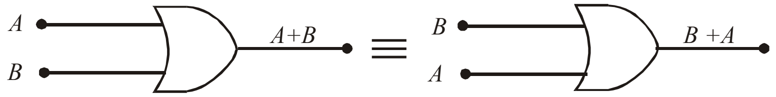


Fig. 4-1. *Illustrating cummutative law of addition*

(*ii*) **Cummutative law of multiplication.** This law for 2 variables is written algebraically as,

$$AB = BA$$

This law states that the order in which the variables are ANDed makes no difference. Fig 4-2 illustrates the law as applied to the AND gate.

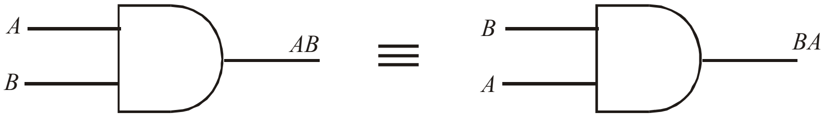


Fig. 4-2. *Illustrating cummutative law of multiplication.*

2. **Associative laws.** Like cummutative laws, there are 2 laws under this category: (*i*) associative law of addition and (*ii*) associative law of multiplication.

(*i*) **Associative law of addition.** This law for 3 variables is written algebraically as,

$$A+(B+C) = (A+B)+C \qquad \text{...(3-3)}$$

This law states that in ORing of more than 2 variables, the result is the same regardless of the grouping of the variables. Fig. 4-3 illustrates this law as applied to OR gates.

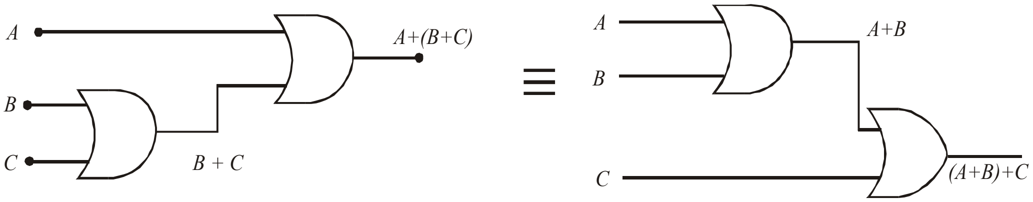


Fig. 4-3. *Illustrating associative law of addition.*

(*ii*) **Associative law of multiplication.** This law for 3 variables is written algebraically as:

$$A\ (BC) = (AB)\ C \qquad \text{...(3-4)}$$

This law states that it makes no difference in what order the variables are grouped when ANDing more than 2 variables. Fig. 4-4 illustrates this law when applied to AND gates.

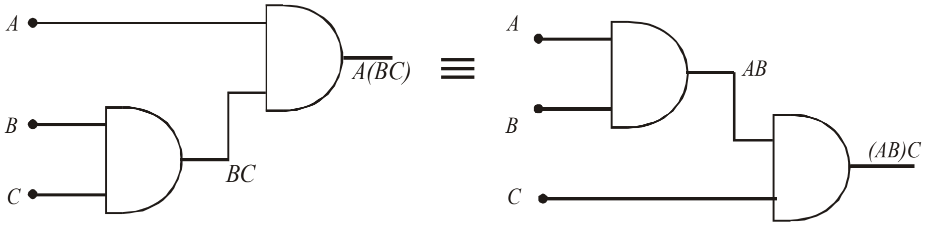


Fig. 4-4. *Illustrating associative law of multiplication.*

3. **Distributive Law.** This law is the same as in ordinary algebra. The distributive law for 3 variables is written as follows:

$$A\ (B+C) = AB+AC \qquad \text{... (3-5)}$$

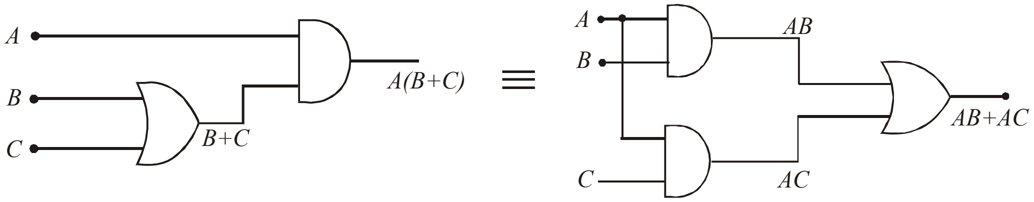


Fig. 4-5. *Illustrating a distributive law.*

This law states that ORing 2 or more variables and ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of **factoring** in which a common variable A is factored out of the product terms. Fig 4-5 illustrates the distributive law in terms of gate implementation.

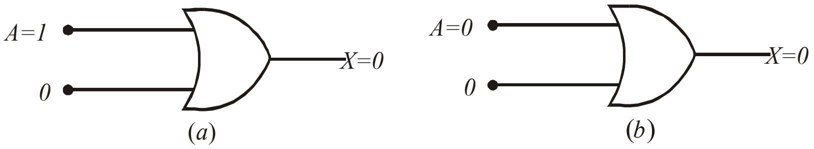
4-4 Rules of Boolean Algebra

Table 4-1 lists the 12 basic rules that are useful in manipulating and simplifying Boolean expressions. We will view the rules 1 through 9 in terms of their application to logic gates. Whereas rules 10 through 12 will be derived in terms of simpler rules and the laws previous discussed.

Table 4-1. Twelve Basic rules of Boolean algebra

1. $A.0 = 0$	7. $A.A = A$
2. $A.1 = A$	8. $A.\overline{A} = 0$
3. $A + 0 = A$	9. $\overline{\overline{A}} = A$
4. $A + 1 = 1$	10. $A + AB = A$
5. $A + A = A$	11. $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	12. $(A + B)(A + C) = A + BC$

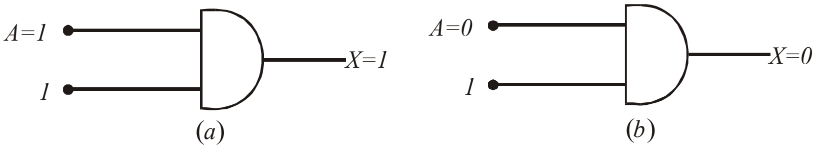
1. **A . 0 = 0** This rule states a variable ANDed with 0 is always equal to 0. In other words, if at any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Fig 4-6 (a) & (b) where the lower input is fixed at 0.



(*a*) (*b*)

Fig. 4-6. *Illustrating Rule 1: A . 0 = 0.*

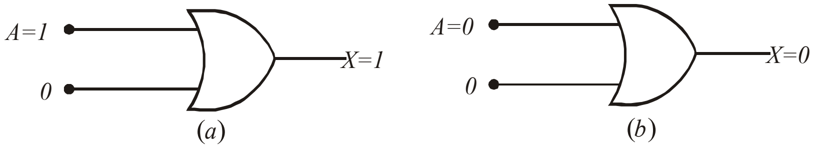
2. **A . 1= A** This rule states that a variable ANDed with 1 is always equal to the variable. If the variable *A* is 0, the output of the AND gate is 0. If the variable *A* is 1, the output of the AND gate is 1, because both input are now 1s. This rule is illustrated in Fig 4-7 (a) and (b) where the lower input is fixed at 1.



(*a*) (*b*)

Fig. 4-7. *Illustrating Rule 2: A . 1 = A.*

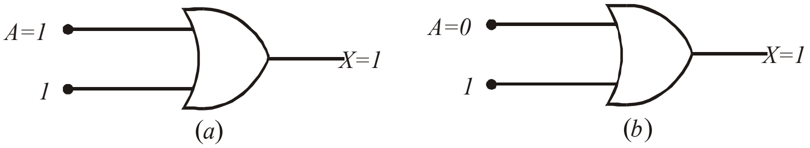
3. **A + 0 = 0** This rule states that a variable ORed with 0 is always equal to the variable. If *A* is 1, the output is 1, which is equal to *A*. If A is 0, the output is 0, which is also equal to *A*. This rule is illustrated in Fig 4-8 (a) and (b) where the lower input is fixed at 0.



(*a*) (*b*)

Fig. 4-8. *Illustrating Rule 3: A + 0 = 0*

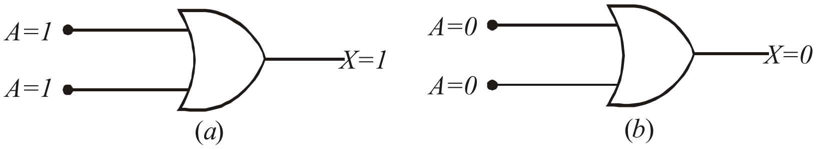
4. **A + 1 = 1** This rule states that a variable ORed with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input. This rule is illustrated in Fig. 4-9 (a) and (b) where the lower input is fixed at 1.



(*a*) (*b*)

Fig. 4-9. *Illustrating Rule 4: A + 1 = 1*

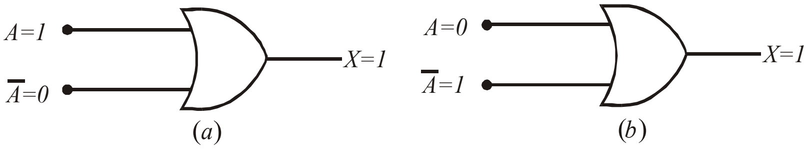
5. **A + A = A** This rule states that a variable ORed with itself is always equal to the variable. If *A* is 0, then 0 + 0 = 0, and if *A* is 1, then 1+1 = 1. This is illustrated in Fig. 4-10 (a) and (b) where both inputs are the same variable.



(*a*) (*b*)

Fig. 4-10. *Illustrating Rule 5: A + A = A*

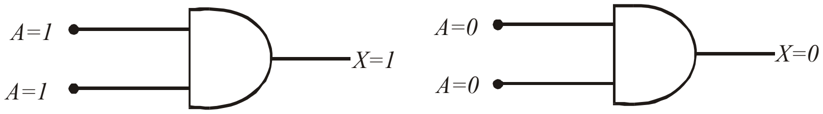
6. $\mathbf{A+\overline{A}=1}$ This rule states that a variable ORed with its complement is always equal to 1. If A is 0, then $0+\overline{0}=0+1=1$. If *A* is 1, then $1+\overline{1}=1+0=1$. This is illustrated in Fig 4-11 (a) and (b) where one input is the complement of the other.



(*a*) (*b*)

Fig 4-11. *Illustrating Rule 6:* $\mathrm{A}+\overline{\mathrm{A}}=1$

7. **A . A = A** This rule states that a variable ANDed with itself is always equal to the variable. If A is 0, then 0.0 = 0, and if *A* is1, then 1 . 1 = 1. This is illustrated in Fig 4-12 (a) and (b)



(*a*) (*b*)

Fig. 4-12. *Illustrating Rule 7: A . A = A*

8. $\mathbf{A.\overline{A}=0}$ This rule states that a variable ANDed with its complement is always equal to 0. Either A or $\overline{A}$ will always be 0, and when a 0 is applied at the input of an AND gate, the output will be 0. This rule is illustrated in Fig 4-13 (a) and (b),

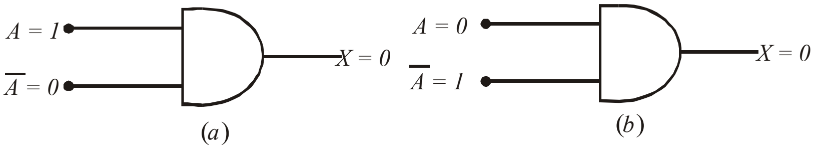


Fig. 4-13. *Illustrating Rule 8:* $A.\overline{A}=0$

9. $\mathbf{\overline{\overline{A}} = A}$ This rule states that a double complement of a variable is always equal to the variable. If we start with the variable A and complement (or invert) it once, we get $\overline{A}$. If we then take $\overline{A}$ and complement it again, we get A, which is the original variable, This rule is illustrated in Fig 4-14 (a) and (b).

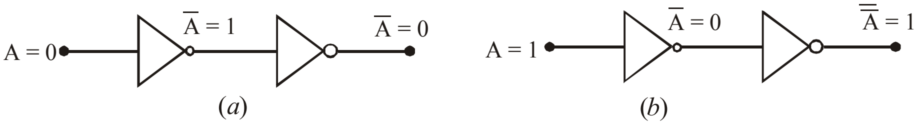


Fig. 4-14. *Illustrating Rule 9:* $\overline{\overline{A}}.A$

10. **A+AB = A** This rule can be proved by applying the distributive law, rule 4 and rule 2 as follows:

$A+AB = A\ (1+B)$ — Factoring (distributive law)

$= A.1$ — Rule 4: $(1+B) = 1$

$= A$ — Rule 2: $A.1= A$

11. $\mathbf{A + \overline{A}B = A + B}$ This rule can be proved as follows:

$A + \overline{A}B = (A + AB) + \overline{A}B$ — Rule 10 : $A = A + AB$

$= (AA + AB) + \overline{A}B$ — Rule 7 : $A = AA$

$= AA + AB + A\overline{A} + \overline{A}B$ — Rule 8 : Adding $A\overline{A} = 0$

$= (A + \overline{A})(A + B)$ — Factoring (distributive law)

$= 1.(A + B)$ — Rule 6 : $A + \overline{A} = 1$

$= A + B$ — Rule 2 : drop the 1

12. **(A+B) (A+C) = A+BC** This rule can be proved as follows:

$(A+B)\ (A+C) = AA + AC + AB + BC$ — Distributive law

$= A + AC + AB + BC$ — Rule 7: $AA = A$

$= A\ (1+C) + AB + BC$ — Factoring (distributive law)

$= A.1 + AB + BC$ — Rule 4: $1 + C = 1$

$= A\ (1+B) + BC$ — Factoring (distributive law)

$= A.\ 1 + BC$ — Rule 4: $1 + B = 1$

$= A + BC$ — Rule 2: $A\ .\ 1 = A$

Example 4-1. *Using the Truth table, show that the Boolean expression $A + AB = A$.*

Solution. Given the Boolean expression $A + AB = A$. From the given expression, we find that we have 2 variables i.e. A and B. These can have 4 possible combinations i.e. 00, 01, 10 and 11. This is shown in the first two columns of the truth table shown in Table 4-2. The third column for AB is obtained by ANDing the first two columns. The fourth column of the truth table is obtained by ORing the first column and the third column. An inspection of the first and the fourth column indicates that there are the same. This verifies that $A + AB = A$.

Table 4-2.

A	*B*	*AB*	*A+AB*
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

These two columns are equal

Example 4-2. *Using the truth table, show that the Boolean expression,* $A + \bar{A}B = A + B$

Solution. Given: The Boolean expression $A + \bar{A}B = A + B$

From the given expression, we find that we have two variables, i.e. A and B. These two variables can have 4 possible combinations, i.e. 00, 01, 10 and 11. This is shown in the first two columns of the truth table shown in Table 4-3. The third column $(\bar{A})$ is obtained by complementing the first column. The fourth column $(\bar{A}B)$ is obtained by ANDing the second and the third column. The fifth column is obtained by adding the first and fourth column. The sixth column is obtained by adding the first two columns.

An inspection of the fifth and sixth columns indicate that these are the same. This verifies that $A + \bar{A}B = A + B \cdot$

Table 4-3.

A	*B*	$\bar{A}$	$\bar{A}B$	$A+\bar{A}B$	*A+B*
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

Check these columns are equal.

Example 4-3. *Using the truth table, show that the Boolean expression, (A+B)(A+C) = A+BC.*

Solution. Given the Boolean expression,

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

From the given expression, notice that there are 3 variables, A, B and C. These three variables can have 8 possible combinations listed in Table 4-4 in the first three columns. The fourth column is obtained by ORing the first two. The fifth column is obtained by ORing the first and third column. The sixth column is obtained by

Table 4-4.

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

Check these two colums are qual.

ANDing the fourth and fifth column. The seventh column is obtained by ANDing the second and third columns. Finally the eighth column is obtained by ORing the first and seventh column. Notice that the sixth and the eighth columns are the same. This verifies that *(A+B)(A+C) =A+BC*

Example 4-4. *Using the Boolean laws and rules simplify the expression,*

$$y \;=\; A\overline{B}D \;+\; A\,\overline{B}\,\overline{D}$$

Solution. Given : The expression

$$y \;=\; A\overline{B}D \;+\; A\overline{B}\overline{D}$$

To simplify this expression, factor out the common variables $A\overline{B}$ by applying distributive law (3–5),

$$\therefore \qquad y \;=\; A\overline{B}(D+\overline{D})$$

Applying Rule 5 $(A+\overline{A}=1)$ to the term $D+\overline{D}$ we get,

$$y = A\overline{B}(D+\overline{D}) = A\overline{B}.1$$

Applying Rule 2 (A.1 = A) to the right term, we get,

$$y = A\overline{B} \text{ **Ans.**}$$

Example 4-5. *Using the Boolean Laws and rules, simplify the expression,*

$$x = ACD + \overline{A}BCD$$

Solution. Given : The Boolean, expression,

$$x = ACD + \overline{A}BCD$$

We know that the given expression can be simplified by applying the Boolean laws and rules repeatedly. Thus, applying the distributive law (3–5) to the terms on the right, we get,

$$x = ACD + \overline{A}BCD$$
$$= (A + \overline{A}B)CD$$

Now applying Rule 11 : $(A+\overline{A}B=A+B)$, we get,

$$x=(A+\overline{A}B)CD$$
$$=(A+B)CD \quad \textbf{Ans.}$$

Example 4-6. *Using the Boolean laws and rules, simplify the logic expression,s*

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

(*Anna University., Nov./Dec. 2007*)

Solution. Given : The logic expression,

$$Z=(\overline{A}+B)(A+B) \qquad ...(i)$$

To simplify this expression, let us apply the distributive law to the two terms in parenthesis,

$$Z=\overline{A}A+\overline{A}B+AB+BB \qquad ...(ii)$$

Applying Rule 8 : *$(\overline{A}A=0)$* to the first term and Rule 7 : ($BB = B$) to the last term, equation (*ii*) can be written as,

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

Factoring out the variable B from all the three terms of the above equation, we get,

$$Z=(\overline{A}+A+1)B$$

Applying Rule 4 and 6 $(\overline{A}+A+1=1)$ to the term in parenthesis, we get,

$$Z = 1.B$$

Applying Rule 2 : *(1.B = B)* to the right side, we get,

$$Z = B \quad \textbf{Ans.}$$

Example. 4-7. *Using Boolean algebra techniques, simplify the expression* :

$$Z = AB + A(B + C) + B(B + C)$$

Solution. Given : The expression,

$$Z = AB + A(B + C) + B(B + C)$$

Applying the distributive law to the second and third terms in the expression on the right :

$$Z = AB + AB + AC + BB + BC$$

Applying Rule 7 : ($BB = B$) to the fourth term on the right,

$$Z = AB + AB + AC + B + BC$$

Applying Rule 5 : *(AB + AB = AB)* to the first two terms,

$$Z = AB + AC + B + BC$$

Applying Rule 10 : *(B + BC = B)* to the last two terms

$$Z = AB + AC + B$$

Applying Rule 10 : *(AB + B = B)* to the first and third terms

$$Z = B + AC \quad \textbf{Ans.}$$

Note : This approach is not necessarily the only approach to reach at the result.

Example 4-8. *Using laws and rules of Boolean algebra, Simplify the Boolean expression,*

$$Z=[A\overline{B}(C+BD)+\overline{A}\overline{B}]C$$

Solution. Given : The boolean expression,

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

Applying the distributive law to the terms on the right within brackets :

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

Applying Rule 8 $(\bar{B}B=0)$ to the second term on the right within parenthesis :

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

Applying Rule 1 *(A.0.D = 0)* to the second term on the right within paranthesis :

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

Applying Rule 3 (drop the zero) within parenthesis,

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

Applying the distributive law,

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

Applying Rule 7 *(CC = C)* to the first term on the right,

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

Factoring out $\bar{B}C$ from the first and second terms on the right,

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

Applying Rule 6 $(A+\bar{A}=1)$,

$$Z=1.\bar{B}C$$

Applying Rule 2 (drop the 1),

$$Z=\bar{B}C \text{ } \mathbf{Ans.}$$

Example 4-9. *Using Boolean laws and rules simplify the expression*;

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

(*Jamia University., 2007*)

Solution. Given the Boolean expression,

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

Factoring BC out of the first and last terms on the right,

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

Applying Rule 6 : $(\bar{A}+A=1)$ to the firm term within parenthesis and factoring $A\bar{B}$ out from the second and last terms on the right,

$$Z=1.BC+A\bar{B}(\bar{C}+C)+\bar{A}\bar{B}\bar{C}$$

Applying Rule 2 : (drop the 1) to the first term and Rule 6 $(\bar{C}+C=1)$ to the term within parenthesis,

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

Applying Rule 2 : (drop the 1) to the second term on the right,

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

Factoring $\overline{B}$ out from the second and third terms on the right,

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

Applying Rule 11 : $(A + \overline{A}\,\overline{C} = A + \overline{C})$ to the term in parenthesis,

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

Using the distributive and commutative laws, we get,

$$Z = BC + A\overline{B} + \overline{B}\,\overline{C}$$ **Ans.**

Example 4-10. *Prove the following using Boolean Theorems.*

(*i*) $(x + \overline{x}\,\overline{y})(\overline{x} + \overline{y}) + yz = \overline{y} + z$

(*ii*) $\overline{w}\,\overline{y}\,\overline{z} + wz + \overline{y}z + xyz = \overline{w}\,\overline{y} + wz + xz$

(*VTU., July/Aug. 2004*)

Solution. (*i*) Given: The Boolean expression

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

Let us solve the left side of the expression, applying the distributive law to the learn within brackets:

$$x\overline{x} + x\overline{y} + \overline{x}\,\overline{y} + \overline{x}\,\overline{y} + yz$$

Applying Rule 8: $A\overline{A}$ = 0 in the first term we get

$$x\overline{y} + \overline{x}\,\overline{y} + \overline{x}\,\overline{y} + yz$$

Factoring out the variable $\overline{y}$ from the first second and third terms in the expression, we get

$$\overline{y}\,(x + \overline{x} + \overline{x}) + yz$$

Applying Rule 6: $A + \overline{A}$ = 1, we get,

$$\overline{y} + yz$$

Applying Rule 11: $A + \overline{A}B$ = A + B, we get,

$$\overline{y} + z$$ **Ans.**

(*ii*) Given the Boolean expression.

$$\overline{w}\,\overline{y}\,\overline{z} + wz + \overline{y}\,z + xyz = \overline{w}\,\overline{y} + wz + xz\,.$$

Factoring out the variable $\overline{y}$ from the first and third term in the left side of the equation,

$$\overline{y}(\overline{w}\,\overline{z} + z) + wz + xy\,z$$

Applying Rule 11: $A + \overline{A}B = A + B$ in the first term, we get

$$\overline{y}\,(\overline{w} + z) + wz + xy\,z$$

Rearranging the above equation,

$$\overline{w}\,\overline{y} + \overline{y}\,z + wz + xy\,z$$

Factoring out the variable *z* firm the second and fourth term we get,

$$\overline{w}\,\overline{y} + wz + (\overline{y} + xy)\,Z$$

Applying Rule 11: $A + \overline{A}B = A + B$ in the third term, we get,

$$\overline{w}\,\overline{y} + wz + (\overline{y} + x)\,Z$$

$$\overline{w}\,\overline{y} + wz + \overline{y}z + xz.$$

Example 4-11. *Reduce the Boolean Expression* :

$$F = AB + \overline{AC} + A\overline{B}C\,(AB + C)$$ (*Nagpur University., 2004*)

Solution. Given: The boolean expression

$$F = AB + \overline{AC} + A\overline{B}C\,(AB + C)$$

Let us apply the distributive law,

$$= AB + \overline{AC} + A\overline{B}BC\ A\overline{B}CC$$

Applying Rule 8: $(\overline{B}B = 0)$ and Rule 7 ($CC = C$), we get.

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

Factoring out the variable A from the first and last term,

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

Applying Rule 12: $(B + \overline{B}C) = (B + \overline{B})(B + C)$

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

Applying Rule 6: $B + \overline{B} = 1$

$$= A \cdot 1(B + C) + \overline{AC}$$

$$= AB + AC + \overline{AC}$$

Applying Rule 6: $AC + \overline{AC} = 1$

$$= AB + 1$$

Applying Rule 7: $AB + 1 = AB$

$= AB$ **Ans.**

Example 4-12. *Reduce the following using Boolean Algebra*

(*i*) $Z = A\,[B + C(AB + AC)]$

(*ii*) $Z = \overline{A}\,\overline{B}C + (\overline{A + B + C}) + \overline{A}\,\overline{B}\,\overline{C}D.$ (*Jamia University, 2007*)

Solution. (*i*) Given : The Boolean expression

$$Z = A\,[B + C\,(AB + AC)]$$

Applying the distributive law in the expression on the right

$$= AB + AC\,(ABC + AC)$$

$$= AB + ABC + AC$$

Factoring out the variable AB from the first and second term of the above equation

$$= AB\,(1 + C) + AC$$

Applying Rule 4: (1 + C = 1) to the first term

$$= AB + AC$$

Factoring out the variable *A* from the all term, we get

$Z = A\,(B + C)$ **Ans.**

(*ii*) Given : The Boolean Expression

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

Applying De Morgan's theorem 2 in the second term of expression, we get

$$= \overline{A}\,\overline{B}C + \overline{A} \cdot \overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}D$$

Factoring $\overline{A}\,\overline{B}\,\overline{C}$ out of the second and third term on the right,

$$= \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C}\,(1 + D)$$

Applying Rule 4: (A + 1 = 1),

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

Factoring $\overline{A}\,\overline{B}$ out of the all term, we get

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

Applying Rule 6: $(C + \overline{C} = 1)$, we get,

$$= \overline{A}\,\overline{B}\cdot 1$$

$$Z = \overline{A}\,\overline{B}\ \textbf{Ans.}$$

Example 4-13. *Prove the following indentities using Boolean theorems: Also implement with the proper Landware.*

(*i*) $A + BC = (A + B)\cdot(A + C)$

(*ii*) $\overline{A}B + A\overline{B} + \overline{A}\,\overline{B} + AB = 1$

(*Jamia University., 2007*)

Solution. The logic expression

(*i*) Given : $A + BC = (A + B)\cdot(A + C)$

Applying distributive law, we get

$$(A + B)\,(A + C) = AA + AC + AB + BC$$

Applying Rule 7: ($AA = A$) to the first term.

$$= A + AC + AB + BC$$

Using factoring (distributive law)

$$= A\,(1 + C) + AB + BC$$

Applying Rule 4: $1 + C = 1$

$$= A\cdot 1 + AB + BC$$

Again using factoring, we get

$$= A\,(1 + B) + BC$$

Applying Rule 4: $1 + B = 1$, we get

$$= A + BC$$

$$A + BC = (A + B)\,(A + C)\ \textbf{Ans.}$$

In order to implement this expression as a logic circuit, we find that it requires three variables A, B and C. The term BC *i*, obtained by using AND gate. Finally the output *i*, obtained by using OR gate as shown in Fig. 4-15.

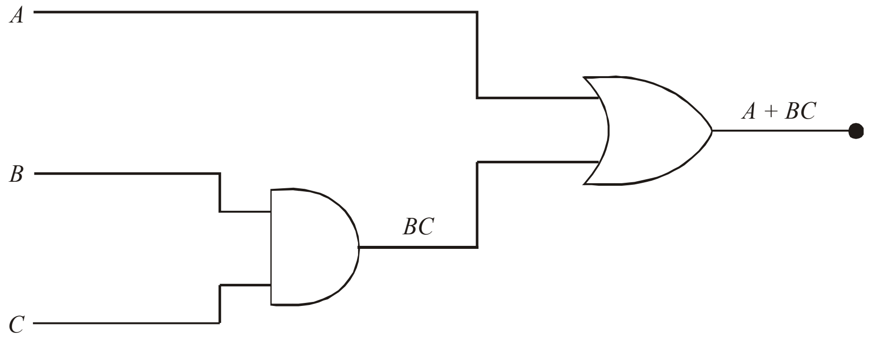


Fig. 4-15.

(*ii*) Given: The logic expression,

$$\overline{A}B + A\overline{B} + \overline{A}\,\overline{B} + AB = 1$$

Rearranging the term, we get

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

Factoring out the variable *B* from the first two term and variable $\overline{B}$ from the last two term we get,

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

Applying Rule 6: $A + \overline{A} = 1,$ we get

$$= B \cdot 1 + \overline{B} \cdot 1$$

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

Applying Rule 6: $(B + \overline{B} = 1),$ we get

$$= 1.$$

$\therefore \quad \overline{A}B + A\overline{B} + \overline{A}\,\overline{B} + AB = 1.$ **Ans.**

In order to implement this to expression as a logic circuit we find that it requires two variables *A* and *B*. The logic circuit i, shown in Fig. 4-16.

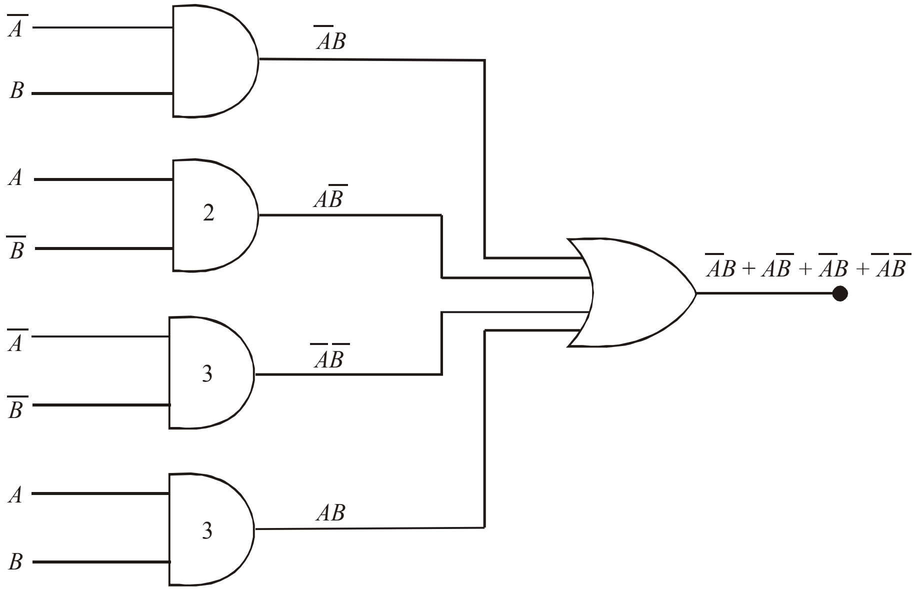


Fig. 4-16.

Example 4-14. *Simplify*

$$Z = A \cdot B + ABC + \overline{A}B + A\overline{B}C$$

(*RGPV., June 2008*)

Solution. Given: The logic expression,

$$Z = AB + ABC + \overline{A}B + A\overline{B}C$$

Factoring out the variable AC from the second and fourth term of the above equation, we get,

$$= AB + \overline{A}B + AC\,(B + \overline{B})$$

Applying Rule 6: $(B + \overline{B} = 1)$ to the third term

$$= AB + \overline{A}B + AC \cdot 1$$

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

Factoring out the variable B from the first and second term of the above equation, we get

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

Applying Rule 6: $A + \overline{A} = 1$ to the first term, we get

$$\begin{aligned} &= B \cdot 1 + AC \\ Z &= AC + B \ \textbf{Ans.} \end{aligned}$$

Example 4-15. *Prove that* $A + \overline{A}B = A + B$.

(*Gujarat Technological University, Dec., 2009*)

Solution. Given: The expression

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

Applying Rule 12: In the left side of expression

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

Applying Rule 6: $A + \overline{A} = 1$ in the left side of expression, we get

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

Example 4-16. *Prove using Boolean Algebra*

(*i*) $(AB + C + D)\,(C + \overline{D})(C + \overline{D} + E) = AB\overline{D} + C$

(*ii*) $\overline{A}B(D + \overline{C}D) + (A + \overline{A}CD)\,B = B$

Solution. (*i*) Given: $(AB + C + D)\,(C + \overline{D}) + (C + \overline{D} + E) = AB\overline{D} + C$

Applying the distributive law to the second and last term inthe expression on the left, we get

$$(AB + C + D)\,(C + C\overline{D} + CE + \overline{D} + \overline{D}E)$$

Factoring out C and $\overline{D}$ from the second parenthesis, we get

$$(AB + C + D)\,[C\,(1 + \overline{D}) + CE + \overline{D}(1 + E)]$$

Applying Rule 4: $(A + 1 = 1)$ to the term $1 + \overline{D}$ and $1 + E$ we get

$$(AB + C + D)\,[C + CE + \overline{D}]$$

Factoring out C from the second parenthesis, we get

$$(AB + C + D)\,[C + (1 + E) + \overline{D}]$$

Applying Rule 4: $(A + 1 = 1)$ to the term $1 + E$, we get

$$(AB + C + D)(C + \overline{D})$$

Applying the distribution law, we get

$$ABC + C + CD + AB\overline{D} + C\overline{D} + D\overline{D}$$

Applying Rule 8: $(A \cdot \overline{A} = 0)$ to the term $D\overline{D}$, we get

$$ABC + C + CD + AB\overline{D} + C\overline{D}$$

Example 4-17. *Simplify*

(*a*) $\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

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

(*c*) $(B + BC)(B + \overline{B}C)(B + D)$

(*Nagpur University, 2008*)

Solution. (*a*) Given : The logic expression,

$$\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Factoring out the variable AB from the third and fourth terms of the above equation, we get,

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

Applying Rule 6 $(\overline{C} + C = 1)$ to the term in parenthesis, we get,

$$\overline{A}BC + A\overline{B}C + AB.1$$

$$\overline{A}BC + A\overline{B}C + AB$$

Factoring out the variable *A* from the second and third terms of the above equation, we get,

$$\overline{A}BC + A(\overline{B}C + B)$$

Applying Rule 11 $(B + \overline{B}C = B + C)$ to the last term, we get,

$$\overline{A}BC + A(B + C)$$

$$\overline{A}BC + AB + AC$$

Factoring out the variable *B* from the first and second terms of the above equation, we get,

$$B(\overline{A}C + A) + AC$$

Applying Rule 11 $(A + \overline{A}C = A + C)$ to the last term, we get,

$$B(A + C) + AC$$

$$AB + BC + AC \textbf{ Ans.}$$

(*b*) Given: The logic expression,

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

Applying the distributive law to the second and third parenthesis in the expression, we get,

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

Applying Rule 8 $(A\bar{A} = 0)$, we get

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

Applying the distributive law to in the expression, we get,

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

Applying Rule 8 $(A\bar{A} = 0)$, we get,

$$A\bar{B} \quad \textbf{Ans.}$$

(*c*) Given: The logic expression,

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

Applying the distributive law to the first and second parenthesis in the expression, we get,

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

Applying Rule 7 $(A \cdot A = A)$ and rule 8 $(A.\bar{A} = 0)$, we get,

$$(B + BC)(B + D)$$

Applying the distributive law, we get,

$$BB + BD + BBC + BCD$$

Applying Rule 7 $(A \cdot A = A)$, we get,

$$B + BD + BC + BCD$$

Factoring out the variable B from the first and second terms and BC from the third and fourth terms of the above equation, we get,

$$B(1 + D) + BC(1 + D)$$

Applying Rule 4: $(D + 1 = 1)$, we get,

$$B + BC$$

Factoring out the variable B from the first and second terms of the above equation, we get,

$$B(1 + C)$$

Applying Rule 4: $(C + 1 = 1)$, we get,

$$B \quad \textbf{Ans.}$$

Example 4-18. *Simplify the Boolean functions.*

(*a*) $xy + \bar{x}yz$

(*b*) $y(w\bar{z} + wz) + xy$

(*Mahatma Gandhi University, Dec. 2007*)

Solution. (*a*) Given : The Boolean functions

$$xy + \bar{x}yz$$

Factoring out the variable y from the above equation, we get,

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

Applying Rule 11 $(A + \bar{A}B = A + B)$, we get,

$$= y(x + z) \quad \textbf{Ans.}$$

(*b*) Given: The Boolean functions

$$y(w\bar{z} + wz) + xy$$

Factoring out the variable *w* from the above equation, we get,

$$= yw(\overline{z} + z) + xy$$

Applying Rule 6 $(A + \overline{A} = 1)$ in the first term, we get,

$$= yw.1 + xy$$
$$= yw + xy$$

Factoring out the variable *y* from the above equation, we get,

$$= y\,(w + x)$$ **Ans.**

Example 4-19. *Prove the following Boolean identity.*

(*GBTU/MTU, 2007*)

(*a*) $A + B.C = (A + B)\,(A + C)$

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

(*c*) $(A + B)\,(A + \overline{B})(\overline{A} + C) = AC$

Solution. (*a*) Given: The Boolean identity

$$= A + B.C = (A + B)\,(A + C)$$

To prove the identity lat take the left side of the identity,

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

We know that by the Rule 4 (A + 1 = 1), multiplying the first term by 1 + B, we get,

$$= A\;(1 + B) + B{\cdot}C$$

Applying the distributive law to the first terms within brackets,

$$= A\;1 + AB + B{\cdot}C$$

We know that by the Rule 4 (A + 1 = 1), multiplying the first term by 1 + C, we get,

$$= A\;(1 + C) + AB + B{\cdot}C$$

Applying the distributive law to the first terms within brackets,

$$= A\;1 + AC + AB + B{\cdot}C$$

Applying Rule 7 ($A{\cdot}A$ = 1) to the first term, we get,

$$= AA + AC + AB + B{\cdot}C$$

Factoring variable A out of the first and second terms and variable B from the third and fourth terms,

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

(*b*) Given : The Boolean identity

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

To prove the identity lat take the left side of the identity,

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

We know that by the Rule 4 (A + 1 = 1), multiplying the first term by 1 + A, we get,

$$= \;\overline{B}(1 + A) + A \cdot B$$

Applying the distributive law to the first terms within brackets,

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

Factoring variable A from the second and third terms, we get

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

Applying Rule 6 : $(A + \overline{A} = 1)$ to the second term we get,

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

(*c*) Given: The Boolean identity

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

To prove the identity lat take the left side of the identity,

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

Applying the distributive law to the first and second terms within brackets,

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

Applying Rule 7 : $(AA = A)$

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

Applying Rule 8 : $(A\overline{A} = 0)$ in the fourth term in the first bracket, we get,

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

Factoring out A from the first and second term in the first bracket,

$$= [A\,(1 + \overline{B}) + A \cdot B](\overline{A} + C)$$

Applying Rule 4 : $(A + 1 = 1)$ in the first term in the first bracket, we get,

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

Factoring out A from the first and second term in the first bracket,

$$= [A\,(1 + B](\overline{A} + C)$$

Applying Rule 4 : $(A + 1 = 1)$ in the first term in the first bracket, we get,

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

Applying the distributive law, we get,

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

Applying Rule 8 : $(A\overline{A} = 0)$ in the first term, we get,

$$= AC$$

Example 4-20. *Prove the following consensus laws using Boolean postulates.*

(*a*) $xy + yz + \overline{x}z = xy + \overline{x}z$

(*b*) $(x + y)(y + x)(\overline{x} + z) = (x + y)(\overline{x} + z)$

(*VTU, Jan/Feb. 2006*)

Solution. (*a*) Given : The Boolean expression $xy + yz + \overline{x}z = xy + \overline{x}z$

Multiplying the second term with $x + \overline{x}$, we get,

$$xy + yz\,(x + \overline{x}) + \overline{x}z \qquad\qquad x + \overline{x} = 1$$

$$= xy + xyz + \overline{x}yz + \overline{x}z$$

Factoring out the variable xy from the first term and $\overline{x}z$ from the third and fourth term, we get,

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

Applying Rule 4 : $(Z + 1 = 1$ and $Y + 1 = 1)$ to the first and second term,

$$= xy + \overline{x}z$$

(*b*) Given : The Boolean expression $(x + y)(y + z)(\overline{x} + z) = (x + y)(\overline{x} + z)$

Applying Rule 9 : $(\overline{\overline{A}} = A)$ in the above expression we get,

$$\overline{\overline{(x + y)(y + z)(x + x)}}$$

Applying De Morgan's Theorem 2, we get

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

Multiplying the last term with $(x + \overline{x})$, we get

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

Applying distribution law to the last term, we get,

$$= \overline{(xy + \overline{x}z + xyz + \overline{x}yz)}$$

Factoring out xy from the first and second term and $\overline{x}z$ from the third and fourth term, we get,

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

$$= \overline{(xy + \overline{x}z)}$$

Applying De Morgan's Theorem 1, we get,

$$= (x + y)(\overline{x} + z).$$

4-5. Duality Principle

One of the rule of Boolean algebra in each pair can be obtained from the other one by the following :

(*a*) By interchanging the two binary operators

(*b*) By interchanging the two identity elements

Thus, one of the commutative laws can be obtained from the other by interchanging the operators (+) and (·). The same is true of the distributive laws. So the two Boolean functions are said to be dual of each other if any one of these is obtained from the other one by as following :

(*a*) Interchanging (+) OR and (·) AND signs

(*b*) Interchanging 0 and 1

Let take the Rule 1,

$$A \cdot 0 = 0$$

We interchange (·) to (+) and 0 to 1, then we get

$$A + 1 = 1$$

This is same as Rule 4 of Boolean algebra.

Example 4-21. *Write the duals of the following Boolean expression*

(*a*) $a + b + ab = a + b$

(*b*) $a + b + \overline{a}\,\overline{b} = 1$

(*VTU Jan./Feb. 2005*)

Solution. (*a*) Given : The Boolean expression,

$$a + b + ab = a + b$$

We know that by duals theorem, interchanging of (+) and (·) signs we get,

$$a.b.\,(a + b) = a.b$$

Applying the distributive law in left term, we get

$$a.b.\,(a + b) = a.a.b + a.b.b = ab + ab = ab$$

(*b*) Given : The Boolean expression,

$$aa + b + \overline{a}\,\overline{b} = 1$$

We know that by duals theorem, interchanging of (+) and (·) signs we get,

$$(a + a).b.(\overline{a} + \overline{b}) = 1$$

Also by duals theorem, interchanging of 1 and 0 signs we get

$$(a + a).b.(\overline{a} + \overline{b}) = 0$$

Applying the distributive law in left term, we get

$$(a + a).b.(\overline{a} + \overline{b}) = ab(\overline{a} + \overline{b}) = a\overline{a}b + ab\overline{b}$$

Applying Rule 8 $(A\overline{A} = 0)$ to the first and second term, we get

$$= 0.$$

4-6. De Morgan's Theorems

De Morgan, a mathematician proposed the following two theorems that are an important part of Boolean algebra :

Theorem 1. The complement of a product of variables is equal to the sum of the complements of the variables, For two variables, the theorem 1 is given by the expression,

$$\overline{XY} = \overline{X} + \overline{Y} \qquad ...(i)$$

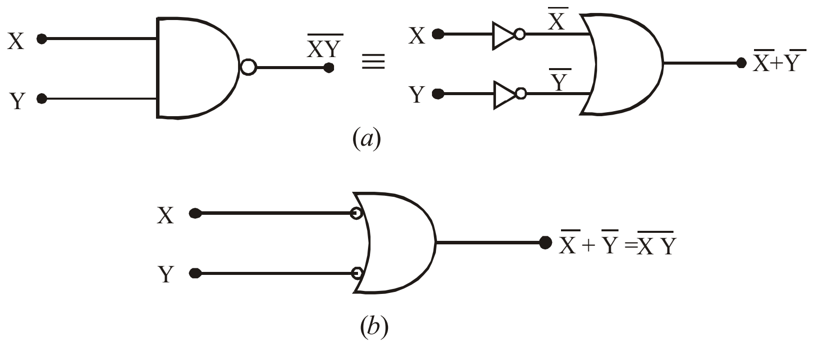
Theorem 2. The complement of a sum of variables is equal to the product of the complements of the variables. For two variables, the theorem 2 is given by the expression,

$$\overline{X + Y} = \overline{X}\,\overline{Y} \qquad ...(ii)$$

Let us now examine the two theorems in terms of logic circuits. First consider theorem 1, i.e.,

$$\overline{XY} = \overline{X} + \overline{Y}$$

The left hand side of the above equation can be viewed as the output of a NAND gate whose inputs are X and Y. On the other hand, the right hand side the equation is the result of first inverting X and Y and then putting them through an OR gate. These two representations are equivalent and are illustrated in Fig. 4-17 (a). This implies that an OR gate with INVERTERs on each of its inputs is equivalent to a NAND gate. When the OR gate with inverted inputs is used to represent the NAND function, it is usually drawn as shown in Fig. 4-17 (b), where the small circles (or bubbles) represent inversion.


X
Y
$\overline{XY}$
≡
X
Y
$\overline{X}$
$\overline{Y}$
$\overline{X}+\overline{Y}$
X
Y
$\overline{X} + \overline{Y} = \overline{X\,Y}$

(*a*)

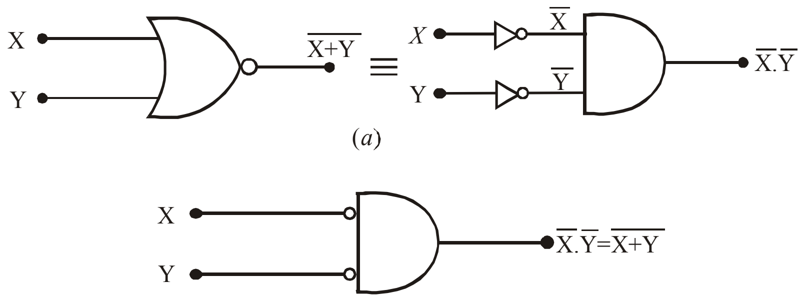
(*b*)

Fig. 4.17.

Let us now consider the theorem 2 ; i.e.,

$$\overline{X+Y}=\overline{X}.\overline{Y}$$

The left hand side of the above equation can be viewed as the output of a NOR gate whose inputs are *X* and *Y*. On the other hand, the right hand side of the equation is the result of first inverting both *X* and *Y* and then putting them through an AND gate. These two representations are equivalent and are illustrated in Fig. 4-18 (a), This implies that an AND gate with INVERTERs on each of its inputs is equivalent to a NOR gate. When the AND gate with inverted inputs is used to represent the NOR function, it is usually drawn as shown in Fig. 4-18 (b), where the small circles (or bubbles) on the inputs represent the inversion operation.


X
Y
$\overline{X+Y}$
≡
X
Y
$\overline{X}$
$\overline{Y}$
$\overline{X}.\overline{Y}$
X
Y
$\overline{X}.\overline{Y}=\overline{X+Y}$

(*a*)

(*b*)

Fig. 4-18.

It may be carefully noted that each variable in De Morgan's theorem as stated above in Equations (*ii*) and (*iii*) can also represent a combination of other variables. For example, *X* can be equal to the term *AB* + *C*, and *Y* can be equal to the term *A* + *BC*. So if you apply De Morgan's theorem for two variables as stated by :

$$\overline{XY}=\overline{X}+\overline{Y}$$

to the expression,

$$\overline{(AB+C)(A+BC)}$$

Then we will get the result,

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

Notice that the right hand side of the above expression have two terms : $\overline{AB+C}$ and $(\overline{A+BC})$. Applying De Morgan's theorem $\overline{X+Y}=\overline{X}\,\overline{Y}$ individually to each term again, we get,

$$(\overline{AB+C})+(\overline{A+BC})=(\overline{AB})\overline{C}+\overline{A}\,(\overline{BC})$$

Notice that we still have two terms on the right hand side of the above expression. This means De Morgan's theorem can again be applied. These terms are $\overline{AB}$ and $\overline{BC}$. A final application of De Morgan's theorem gives,

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

Although this result can be simplified further by the use of Boolean rules and laws yet De Morgan's theorems cannot be used any more.

4-7. Application of De Morgan's Theorems to Boolean Expressions

In order to study the procedure for the application of De Morgan's theorems and Boolean algebra, let us consider a specific Boolean expression :

$$\overline{(A+\overline{BC})+D(\overline{E+\overline{F}})}$$

Step 1. Identify the terms to which we can apply De Morgan's theorems and think of each term as a single variable, Let $\overline{A+B\overline{C}}=X$ and $D(\overline{E+\overline{F}})=Y$.

Step 2. Since $\overline{X+Y}=\overline{X}\,\overline{Y}$, therefore,

$$\overline{(\overline{A+BC})+[D(\overline{E+\overline{F}})]}=[\overline{\overline{A+B\overline{C}}}]\ \overline{D(\overline{E+\overline{F}})}$$

Step 3. Use Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bars over the left term of the right-hand side of the above expression (Note that this is not a part of De Morgan's theorem) :

$$[\overline{\overline{A+B\overline{C}}}][\overline{D(\overline{E+\overline{F}})}]=[A+B\overline{C}][\overline{D(\overline{E+\overline{F}})}]$$

Step 4. In the right term of the right-hand side of the above expression, Let $Z=\overline{E+\overline{F}}$. Then,

$$[A+B\overline{C}][\overline{D(\overline{E+\overline{F}})}]=[A+B\overline{C}][\overline{DZ}]$$

Step 5. Since $\overline{DZ}=\overline{D}\,\overline{Z}$, therefore,

$$[A+B\overline{C}][\overline{D(\overline{E+\overline{F}})}]=[A+B\overline{C}][\overline{D}+\overline{\overline{E+\overline{F}}}]$$

Step 6. Use Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bar over the $E+\overline{F}$ part of the term :

$$(A+B\overline{C})(\overline{D}+\overline{\overline{E+\overline{F}}})=(A+B\overline{C})(\overline{D}+E+\overline{F})$$

Example 4-22. *Apply De Morgan's theorems to the following Boolean expressions with* 3 *and* 4 *variables* :

(*a*) $\overline{XYZ}$ (*b*) $\overline{X+Y+Z}$

(*c*) $\overline{WXYZ}$ (*d*) $\overline{W+X+Y+Z}$

Solution. (*a*) Given : The Boolean expression $\overline{XYZ}$.

We know that the given expression by applying De Morgan's theorem 1 can be written as

$$\overline{XYZ}=\overline{X}+\overline{Y}+\overline{Z}\ \textbf{Ans.}$$

(*b*) Given, the Boolean expression $\overline{X+Y+Z}$.

We know that the given expression by applying De Morgan's theorem 1 can be written as,

$$\overline{X+Y+Z}=\overline{X}\,\overline{Y}\,\overline{Z}\ \textbf{Ans.}$$

(*c*) Given the Boolean expression $\overline{W\,XYZ}$.

We know that the given expression, by applying De Morgan's theorem 1 can be written as,

$$\overline{W\,XYZ}=\overline{W}+\overline{X}+\overline{Y}+\overline{Z}\ \textbf{Ans.}$$

(*d*) Given the Boolean expression, $W+X+Y+Z$

We know that the given expression, by applying De Morgan's theorem 2 can be written as,

$$\overline{W+X+Y+Z} = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$$ **Ans.**

Example 4-23. *Using Boolean algebra simplify the following Boolean expression.*

$$Z = \overline{AB+AC} + \overline{A}\,\overline{B}C$$

Solution. Given : The Boolean expression,

$$Z = \overline{AB+AC} + \overline{A}\,\overline{B}C$$

Applying De Morgan's theorem 2 to the first term on the right.

$$Z = \overline{(AB)}.\overline{(AC)} + \overline{A}\,\overline{B}C$$

Applying De Morgan's theorem 1 to each term on the right in the parenthesis,

$$Z = (\overline{A}+\overline{B}).(\overline{A}+\overline{C}) + \overline{A}\,\overline{B}C$$

Applying the distributive law to the two terms on the right in the parenthesis.

$$Z = \overline{A}\,\overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}C$$

Applying Rule 7 $(\overline{A}\,\overline{A} = \overline{A})$ to the first term, and applying Rule 10, $[\overline{A} + \overline{A}\,\overline{B}C = \overline{A}B\,(1+C) = \overline{A}\,\overline{B}]$ to the third and last term on the right,

$$Z = \overline{A} + \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

Applying Rule 10 $[\overline{A} + \overline{A}\,\overline{C} = \overline{A}(1+\overline{C}) = \overline{A}]$ to the first and second terms on the right,

$$Z = \overline{A} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

Applying Rule 10, $[\overline{A} + \overline{A}\,\overline{B} = \overline{A}\,(1+\overline{B}) = \overline{A}]$ to the first and second terms on the right,

$$Z = \overline{A} + \overline{B}\,\overline{C}$$ **Ans.**

Example 4-24. *Simplify the Boolean expression,*

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

to one having only single variables inverted.

Solution. Given : The expression,

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

Applying De Morgan's theorem 1 to the two terms in parenthesis :

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

Applying De Morgan's theorem 2 to the two terms in parenthesis :

$$\therefore \qquad Z = (\overline{\overline{A}}.\overline{C}) + (\overline{B}.\overline{\overline{D}})$$

Applying Rule 9 : $[\overline{\overline{A}}.\overline{C} = A\overline{C}$ and $\overline{B}\,\overline{\overline{D}} = \overline{B}D]$ to the first and second terms in parenthesis :

$$Z = (A\overline{C}) + (\overline{B}.D)$$
$$= A\overline{C} + \overline{B}D$$ **Ans.**

Example 4-25. *Simplify the following expression using De Morgan's theorems and Boolean algebra laws and rules.*

(a) $W = \overline{RST}\,(\overline{R+S+T})$

(b) $X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BC + ABC + A\overline{B}\,\overline{C} + A\overline{B}C$

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

(d) $Z = (\overline{C+D}) + \overline{A}C\overline{D} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}CD + AC\overline{D}$

Solution.

(a) Given : The expression $q = \overline{RST}\,(\overline{R+S+T})$

$$q = \overline{RST}\,(\overline{R+S+T})$$

Applying De Morgan's Theorems and to the terms on the right we get,

$$q = (\overline{R}+\overline{S}+\overline{T})(\overline{R}.\overline{S}.\overline{T})$$

Applying distributive law to the terms on the right, we get,

$$q = \overline{R}\,\overline{R}\,\overline{S}\,\overline{T} + \overline{S}\,\overline{R}\,\overline{S}\,\overline{T} + \overline{T}\,\overline{R}\,\overline{S}\,\overline{T}$$

Applying Rule 7 : *(A.A = A)* to all the three terms on the right, we get,

$$q = \overline{R}\,\overline{S}\,\overline{T} + \overline{R}\,\overline{S}\,\overline{T} + \overline{R}\,\overline{S}\,\overline{T}$$

Applying Rule 5 : *(A + A = A)* we get,

$$q = \overline{R}\,\overline{S}\,\overline{T} \textbf{ Ans.}$$

(b) Given : The expression,

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

Adding another term ABC from our own side, into the given expression we get,

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

$$(\because ABC + ABC = ABC)$$

Combining the 1st term with 4th term, 2nd with the 3rd term, and 5th term with the last term,

$$X = (\overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C}) + (\overline{A}BC + ABC) + (A\overline{B}C + ABC)$$

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

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

$$= \overline{B}\,\overline{C} + BC + AC \textbf{ Ans.}$$

(c) Given : The expression $Z = (B+\overline{C})(\overline{B}+C) + \overline{\overline{A}+B+\overline{C}}$

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

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

$$= 0 + \overline{B}\,\overline{C} + BC + 0 + \overline{\overline{A}+B+\overline{C}}$$

$$= \overline{B}\,\overline{C} + BC + \overline{\overline{A}}\,\overline{B}\,\overline{\overline{C}}$$

$$= \overline{B}\,\overline{C} + BC + A\overline{B}C$$

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

$$= \overline{B}\,\overline{C} + (B + A)\,C \textbf{ Ans.}$$

Example 4-26. *Simplify each of the following expressions using De Morgan's theorems.*

(a) $\overline{\overline{A}B\overline{C}}$ (b) $\overline{\overline{A}+\overline{B}C}$ (c) $\overline{AB\overline{CD}}$

Solution.

(a) Given : The expression $=\overline{\overline{A}B\overline{C}}$

Applying De Morgan's Theorem 1, we get,

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

Using Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bars over the first and third term of the right-side of the above expression,

$$\overline{\overline{A}B\overline{C}}=A+\overline{B}+C \quad \textbf{Ans.}$$

(b) Given : The expression $=\overline{\overline{A}+\overline{B}C}$

Applying De Morgan's Theorem 2, we get,

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

Using Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bars over A, we get,

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

Applying De Morgan's Theorem 1, we get,

$$\overline{\overline{A}+\overline{B}C}=A.\left(\overline{\overline{B}}+\overline{C}\right)$$

Again using Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bars over B, we get,

$$\overline{\overline{A}+\overline{B}C}=A.\left(B+\overline{C}\right) \quad \textbf{Ans.}$$

(c) Given : The expression $=\overline{AB\overline{CD}}$

Applying De Morgan's Theorem 1, we get,

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

Again using Rule 9: $\overline{\overline{A}}=A$ to cancel the double bars on the third term of the right-hand side, we get,

$$\overline{AB\overline{CD}}=\overline{A}+\overline{B}+CD \quad \textbf{Ans.}$$

Example 4-27. *Simplify each of the following expressions using De Morgan's theorems.*

(a) $\overline{A(\overline{B+\overline{C}})D}$ (b) $\overline{(M+\overline{N})(\overline{M}+N)}$ (c) $\overline{\overline{\overline{AB}C}D}$

Solution.

(a) Given : The expression,

$$=\overline{A\left(\overline{B+\overline{C}}\right)D}$$

Applying De Morgan's theorem 1, we get,

$$\overline{A\left(\overline{B+\overline{C}}\right)D}=\overline{A}+\left(\overline{\overline{B+\overline{C}}}\right)+\overline{D}$$

Using Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bars over the second term on the right-hand side, we get,

$$\overline{A\left(\overline{B+\overline{C}}\right)D}=\overline{A}+\left(B+\overline{C}\right)+\overline{D}$$

$$=\overline{A}+B+\overline{C}+\overline{D} \qquad \textbf{Ans.}$$

(*b*) Given : The expression,

$$=\overline{\left(M+\overline{N}\right)\left(\overline{M}+N\right)}$$

Applying De Morgan's Theorem 1, we get

$$\overline{\left(M+\overline{N}\right)\left(\overline{M}+N\right)}=\left(\overline{M+\overline{N}}\right)+\left(\overline{\overline{M}+N}\right)$$

Applying De Morgan's theorem 2 on the two terms on the right-hand side we get,

$$\overline{\left(M+\overline{N}\right)\left(\overline{M}+N\right)}=\overline{M}\,\overline{\overline{N}}+\overline{\overline{M}}\,\overline{N}$$

Using Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bars over *M* and *N*, we get

$$\overline{\left(M+\overline{N}\right)\left(\overline{M}+N\right)} = \overline{M}N+M\overline{N} \ \ \textbf{Ans.}$$

(*c*) Given : The expression,

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

Using Rule 9 : $\overline{\overline{A}}=A$ to cancel the double bar over the first term on the right-hand side, we get,

$$\overline{\overline{\overline{AB}C}D}=\left(\overline{AB}\right)C+\overline{D}$$

Applying again De Morgan's Theorem 1 to the first term on the right-hand side, we get,

$$\overline{\overline{\overline{AB}C}D}=\left(\overline{A}+\overline{B}\right)C+\overline{D}$$

$$=\overline{A}C+\overline{B}C+\overline{D} \ \ \textbf{Ans.}$$

Example 4-28. *Apply De Morgan's theorem to the each of the following Boolean expressions*:

(*a*) $\overline{(\overline{A+B})+\overline{C}}$ (*b*) $\overline{(\overline{A}+B)+CD}$ (*c*) $\overline{(A+B)\overline{C}\overline{D}+(E+\overline{F})}$

Solution.

(*a*) Given the Boolean expression : $\overline{(\overline{A+B})+\overline{C}}$

Let $\overline{A+B}=X$ and $\overline{C}=Y$. The expression $\overline{\overline{A+B}+\overline{C}}$ is of the form $\overline{X+Y}=\overline{X}\overline{Y}$ and can be rewritten as,

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

Using Rule 9 $(\overline{\overline{A}}=A)$ to cancel the double bars over the terms (*A* + *B*) and *C* on the right, we get,

$$\overline{\overline{A+B}\ +\overline{C}}=(A+B)C \ \textbf{Ans.}$$

(*b*) Given the Boolean expression : $\overline{(\overline{A}+B)+CD}$

Let $\overline{A}+B=X$ and *CD* = *Y*. The expression $\overline{(\overline{A}+B)+CD}$ is of the form $\overline{X+Y}=\overline{X}\overline{Y}$ and can be rewritten as,

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

Let $\overline{A}=Z$, then the term $\overline{\overline{A}+B}$ on the right is of the form $\overline{X+Y}=\overline{X}\overline{Y}$. Similarly the term $\overline{CD}$ on the right is of the form $\overline{XY}=\overline{X}+\overline{Y}$. Therefore the expression $(\overline{\overline{A}+B})\overline{CD}$ can be rewritten as,

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

Using Rule 9 $(\overline{\overline{A}}=A)$ to cancel the double bars on the right term $\overline{\overline{A}}\overline{B}$, we get,

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

$\therefore$ $$\overline{\overline{A}+B+CD}=A\overline{B}(\overline{C}+\overline{D})$$ **Ans.**

(*c*) Given : The expression : $\overline{(A+B)\overline{C}\overline{D}+E+\overline{F}}$

Let $(A+B)\overline{C}\overline{D}=X$ and $E+\overline{F}=Y$. The expression $\overline{(A+B)\overline{C}\overline{D}+(E+\overline{F})}$ is of the form $\overline{X+Y}=\overline{X}\,\overline{Y}$ and can be rewritten as,

$$\overline{(A+B)\overline{C}\overline{D}+(E+\overline{F})}=[\overline{(A+B)\overline{C}\overline{D}}][\overline{E+\overline{F}}]$$

Next, apply De Morgan's theorem 1 to the term $\overline{(A+B)\overline{C}\overline{D}}$ and $\overline{E+\overline{F}}$. Let $A+B=P$, $\overline{C}=Q$ and $\overline{D}=R$. Then the right term $\overline{(A+B)\overline{C}\overline{D}}$ is of the form $\overline{PQR}=\overline{P}+\overline{Q}+\overline{R}$. Similarly, let $E=M$ and $\overline{F}=N$. Then right the term $\overline{E+\overline{F}}$ is of the form $\overline{M+N}=\overline{M}\overline{N}$. Both the terms $\overline{(A+B)\overline{C}\overline{D}}$ and $\overline{E+\overline{F}}$ can be rewritten as,

$$[\overline{(A+B)\overline{C}\overline{D}}][\overline{E+\overline{F}}]=[(\overline{A+B})+\overline{\overline{C}}+\overline{\overline{D}}][\overline{E}\,\overline{\overline{F}}]$$

Using Rule 9 $(\overline{\overline{A}}=A)$ to cancel the double bars on C, D and F in the right term, we get,

$$[\overline{(A+B)\overline{C}\overline{D}}][\overline{E+\overline{F}}]=[(\overline{A+B})+C+D][\overline{E}F]$$

Next applying De Morgan's theorem again to the term $\overline{A+B}$ on the right we get,

$$[(\overline{A+B})+C+D][\overline{E}F]=(\overline{A}\overline{B}+C+D)\overline{E}F$$

$\therefore$ $$\overline{(A+B)\overline{C}\overline{D}+(E+\overline{F})}=(\overline{A}\overline{B}+C+D)\overline{E}F$$ **Ans.**

Example 4-29. *Apply De Morgan's theorem to the following expressions* :

(*a*) $\overline{(A+B+C)D}$ (*b*) $\overline{ABC+DEF}$ (*c*) $\overline{A\overline{B}+\overline{C}D+EF}$

(*Anna University, Nov./Dec. 2007*)

Solution.

(*a*) Given : The expression = $\overline{(A+B+C)D}$

Let $A+B+C=X$ and $D=Y$

Then the given expression is of the form $\overline{XY}=\overline{X}+\overline{Y}$ and can be rewritten as,

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

Next applying De Morgan's theorem 2 to the term $\overline{A+B+C}$,

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

$\therefore$ $$\overline{(A+B+C)D}=\overline{A}\,\overline{B}\overline{C}+\overline{D}$$ **Ans.**

(*b*) Given : The expression $\overline{ABC+DEF}$

Let $ABC=X$ and $DEF=Y$

Then the given expression $\overline{ABC+DEF}$ is of the form $\overline{X+Y}=\overline{X}\overline{Y}$ and, can be rewritten as,

$$\overline{ABC+DEF}=\overline{ABC}\,.\overline{DEF}$$

Applying DeMorgan's theorem 1 to each of the terms $\overline{ABC}$ and $\overline{DEF}$,

$$(\overline{ABC})(\overline{DEF})= (\overline{A}+\overline{B}+\overline{C})(\overline{D}+\overline{E}+\overline{F})$$

$$\therefore \qquad \overline{ABC+DEF} \ = (\overline{A}+\overline{B}+\overline{C})(\overline{D}+\overline{E}+\overline{F})$$

(*c*) Given : The expression, $\overline{A\overline{B}+\overline{C}D+EF}$

Let $A\overline{B}=X, \overline{C}D=Y$ and $EF=Z.$

Then the expression, $\overline{A\overline{B}+\overline{C}D+EF}$ is of the form $\overline{X+Y+Z}=\overline{X}\;\overline{Y}\;\overline{Z}$ and can be rewritten as

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

Next applying De Morgan's theorem 1 to each of the terms $\overline{A\overline{B}}, \overline{\overline{C}D}$ and $\overline{EF}$, we get,

$$(\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF})=(\overline{A}+B)(C+\overline{D})(\overline{E}+\overline{F})$$

$$\therefore \qquad \overline{A\overline{B}+\overline{C}D+EF}=(\overline{A}+B)(C+\overline{D})(\overline{E}+\overline{F}) \textbf{ Ans.} \qquad ...\left(\because \overline{\overline{B}}=B \text{ and } \overline{\overline{C}}=C\right)$$

Example 4-30. *Simplify the Boolean expression,*

$$T(x,y,z)=(x+y)\left[\overline{\overline{x}(\overline{y}+\overline{z})}\right]+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

(*Nagpur University, Mar./Apr.* 1998)

Solution. Given : The Boolean expression :

$$T(x,y,z)=(x+y)\left[\overline{\overline{x}(\overline{y}+\overline{z})}\right]+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

$$=(x+y)\left[(\overline{\overline{x}}+\overline{(\overline{y}+\overline{z})}\right]+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

$$=(x+y)\left(x+(\overline{\overline{y}}.\overline{\overline{z}})\right)+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

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

$$=x.x+xyz+xy+yyz+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

$$= x + xyz + xy + yz + \overline{x}\overline{y} + \overline{x}\overline{z}$$

$$=x(1+y+yz)+yz+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

$$=x+yz+\overline{x}\,\overline{y}+\overline{x}\,\overline{z}$$

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

$$=x+\overline{y}+yz+\overline{x}\,\overline{z}$$

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

$$=(x+\overline{z})+(\overline{y}+z)$$

$$=x+\overline{y}+(\overline{z}+z)$$

$$=x+\overline{y}+1$$

$$=x+\overline{y}$$

$$\therefore T(x,y,z)=x+\overline{y} \textbf{ Ans.}$$

Example 4-31. *Using De Morgan's theorem simplify*:

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

(*RGPV Bhopal, June 2008*)

Solution. Given : The logic expression,

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

Applying Rule 6 : $A + \overline{B}C = 1$ in the above equation, we get

$$= 1$$

Let solve with De Mogran's theorem. Apply De Morgan' theorem 2 to the second term of above expression.

$$= (A + \bar{B}C) + \left[(\bar{A}\cdot\overline{(\bar{B}\cdot C})\right]$$

Apply De Morgan's theorem 1 we can written as

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

Using Rule 9 : $\bar{\bar{A}} = A$ to cancel the double bars

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

Applying the distributive law to the last term

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

Example 4-32. *Prove the following using De morgan's theorem.*

(*a*) $AB + CD = \overline{\overline{AB}\cdot\overline{CD}}$

(*b*) $(A + B)\cdot(C + D) = \overline{\overline{A + B} + \overline{C + D}}$

(*Anna University, April/May 2008*)

Solution. (*a*) Given : The expression,

$$AB + CD = \overline{\overline{AB}\cdot\overline{CD}}$$

Let take the L.H.S.

$$= \overline{\overline{AB}\cdot\overline{CD}}$$

Applying De Morgan's theorem 1, we get,

$$= \overline{\overline{AB}} + \overline{\overline{CD}}$$

Using Rule 9 : $\bar{\bar{A}} = A$ to cancel the double bars over AB and CD, we get

$$= AB + CD$$

(*b*) Given : The expression

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

Let take the L.H.S.

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

Applying De Morgan's theorem 2, we get,

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

Using Rule 9 : $\bar{\bar{A}} + A$ to cancel the double bars over $(A + B)$ and $(C + D)$, we get

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

Example 4-33. *Find the complement of* $A\overline{B} + \overline{B}C + C\overline{D}$. (*PTU, May 2009*)

Solution. Given : The expression $X = A\overline{B} + \overline{B}C + C\overline{D}$

Applying De Morgan's theorem

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

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

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

Example 4-34. *Find the complement of*

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

Solution. Given : The function

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

Taking complement,

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

Applying De Morgan's theorem 2, we get

$$(\overline{A\overline{B}})\ (\overline{\overline{B}C})\ (C\overline{D})$$

Applying De Morgan's theorem 1, we get,

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

Using Rule 9 : $\overline{\overline{A}} = A$ to concel the double bars

$$(\overline{A} + B)\ (B + \overline{C})\ (\overline{C} + D) \quad \textbf{Ans.}$$

Example 4-35. *Complement the following Boolean expressions and write then as the sum of mix terms*

(*i*) $\overline{a} + \overline{b} + \overline{c} + \overline{d}$

(*ii*) $ab + (\overline{ab\,cd})$

(*V.T.U., Jan./Feb. 2005*)

Solution. (*i*) Given : The Boolean expression

$$\overline{a} + \overline{b} + \overline{c} + \overline{d}$$

The complement of the function

$$\overline{\overline{a} + \overline{b} + \overline{c} + \overline{d}}$$

Applying De Morgan theorem, we get

$$\overline{\overline{a}}\ \overline{\overline{b}}\ \overline{\overline{c}}\ \overline{\overline{d}}$$

Applying Rule 9 : $\overline{\overline{A}} = A$ to cancel the double bars over the expression we get,

$$= a\ b\ c\ d \ \textbf{ Ans.}$$

(*ii*) Given : The Boolean expression

$$= ab + (\overline{abcd})$$

The complement of the function

$$= \overline{ab + (\overline{abcd})}$$

Applying De. Morgan's theorem, we get

$$= \overline{ab} \cdot (\overline{\overline{abcd}})$$

Applying Rule 9 : $\overline{\overline{A}} = A$ to cancel the double bar over the expression we get,

$$= \overline{ab}\,(ab\,cd)$$

Applying De Morgan theorem, we get

$$= (\overline{a} + \overline{b})\,(ab\,cd) \quad \textbf{Ans.}$$

Example 4-36. *F* = $A + \overline{B}C$ *as sum of minterms.* *(Anna University. Nov. 2007)*

Solution. Given : The Boolean expression $A + \overline{B}C$

We know that the domain of the given expression is *A*, *B* and *C*. From the expression we find that all the two product terms are in the non-standard form. Let us convert these product terms to a standard form taking one term at a time.

The first term A has a missing variable *B* or $\overline{B}$ and C or $\overline{C}$. So multiply the first term by $(B + \overline{B})$ and $(C + \overline{C})$ as follows:

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

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

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

The second term $\overline{B}C$ he a missing variable *A* or $\overline{A}$. So multiply the second term by $(A + \overline{A})$ as follows :

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

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

There the standard form of the given expression

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

$$= ABC + A\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C \qquad (A\overline{B}C + A\overline{B}C = A\overline{B}C)$$

$$F = ABC + A\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C \quad \textbf{Ans.}$$

Example 4-37. *Convert the given expression in canonical SOP form Y = AC + AB + BC.* (*PTU, May 2009*)

Solution. Given : The Boolean expression. *AC* + *AB* + *BC*. We know that the domain of the given expression is A, B and C. From the expression. We find that all the three product terms are in the non-standard form. Let us convert these product terms to standard form taking one term at a time.

The first term *AC* has a missing variable *B* or $\overline{B}$. So multiply the first term by $B + \overline{B}$ as follows:

$$AC = AC\,(B + \overline{B}) = ACB + AC\overline{B}$$

The second term *AB* has a missing variable *C* or $\overline{C}$. So multiply the second term by $C + \overline{C}$ as follows,

$$AB = AB(C+\overline{C}) = ABC + AB\overline{C}$$

The third term BC has a missing variable A or $\overline{A}$. So multiply the third term by $A+\overline{A}$ as follows,

$$BC = BC(A+\overline{A}) = ABC + \overline{A}BC$$

Combining all the terms, thus SOP,

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

$$= ABC + A\overline{B}C + AB\overline{C} + \overline{A}BC \quad \textbf{Ans.}$$

Example 4-38. *Convert the following function to standard SOP form :*

$$f(A, B, C, D) = AB + A\overline{C} + C + AD$$

(*Nagpur University, 2008*)

Solution. Given : The Boolean expression

$$f(A, B, C, D) = AB + A\overline{C} + C + AD$$

We know that the domain of the given expression is A, B, C and D. From the expression we find that all the three product terms are in the non-standard form. Let us convert these product terms to a standard form taking one term at a time.

The first term AB has a missing variable C or $\overline{C}$ and D or $\overline{D}$. So multiply the first term by $(C+\overline{C})$ and $(D+\overline{D})$ as follows,

$$AB = AB(C+\overline{C})(D+\overline{D}) = (ABC + AB\overline{C})(D+\overline{D})$$

$$= ABCD + AB\overline{C}D + ABC\overline{D} + AB\overline{C}\,\overline{D}$$

The second term $A\overline{C}$ has a missing variable B or $\overline{B}$ and D or $\overline{D}$. So multiply the second term by $(B+\overline{B})$ and $(D+\overline{D})$ as follows,

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

$$= AB\overline{C}D + A\overline{CD} + A\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}\,\overline{D}$$

The third term C has a missing variable A or $\overline{A}$ and B or $\overline{B}$ and D or $\overline{D}$. So multiply the third term by $(A+\overline{A})$ and $(B+\overline{B})$ and $(D+\overline{D})$ as follows,

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

$$= (ABC + A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C)(D+\overline{D})$$

$$= ABCD + ABC\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}C\overline{D}$$

The fourth term AD has a missing variable B or $\overline{B}$ and C or $\overline{C}$. So multiply the fourth term by $(B+\overline{B})$ and $(C+\overline{C})$ as follows,

$$AD = AD(B+\overline{B})\;(C+\overline{C}) = (ABD + A\overline{B}D)(C+\overline{C})$$

$$= ABCD + AB\overline{C}D + A\overline{B}CD + A\overline{B}\,\overline{C}D$$

Thus the standard SOP (i.e sum-of-products) form of the given expression,

$$AB + A\overline{C} + C + AD = ABCD + AB\overline{C}D + ABC\overline{D} + AB\overline{C}\,\overline{D} + AB\overline{C}D + AB\overline{CD} + A\overline{B}\,\overline{C}D$$

$$+ A\overline{B}\,\overline{C}\,\overline{D} + ABCD + ABC\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D}$$

$$+ \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + ABCD + AB\overline{C}D + A\overline{B}CD + A\overline{B}\,\overline{C}D$$

$$= ABCD + AB\overline{C}D + ABC\overline{D} + AB\overline{C}\,\overline{D} + A\overline{C}\,\overline{B}\,\overline{D} + A\overline{B}CD$$

$$+ A\overline{B}C\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}C\overline{D} \quad \textbf{Ans.}$$

Example 4-39. *Convert the following boolean functions into standard SOP and express it in terms of minterms.*

$$Y(A, B, C) = AB + AC + B\overline{C}$$

(*GBTU., 2010-11*)

Solution. Given : The Boolean expression,

$$Y(A, B, C) = AB + AC + B\overline{C}$$

We know that the domain of the given expression is *A*, *B* and *C*. From the expression we find that all the three product terms are in the non-standard form. Let us convert this product terms to a standard form taking one term at a time.

The first term *AB* has a missing variable *C* or $\overline{C}$. So multiply the first term by ($C + \overline{C}$) as follows,

$$AB = AB\,(C + \overline{C}) = ABC + AB\overline{C}$$

The second term *AC* has a missing *B* and $\overline{B}$. So multiply the second term ($B + \overline{B}$) as follows

$$A\overline{C} = AC(B + \overline{B}) = ABC + A\overline{B}\,C$$

The third term $B\overline{C}$ has a missing variable *A* or $\overline{A}$. So multiply the third term by ($A + \overline{A}$) as follows.

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

Thus the standard *SOP* form of the given expression,

$$= ABC + A\overline{B}\,C + AB\overline{C} + \overline{A}\,B\overline{C} \quad \textbf{Ans.}$$

4-8. Boolean Analysis of Logic Circuits

Boolean algebra provides a concise way to express the operation of a logic circuit formed in the form of a (Boolean expression). Using the Boolean expression, the output of a logic circuit can be readily determined for various combinations of input values. To derive the Boolean expression of a given logic circuit, we begin at the left-most inputs and work towards the final output, writing the expression for each logic gate.

Once the Boolean expression for a given logic circuit has been determined, a truth table (i.e. a table that shows the output of the logic circuit for all possible values of the input variables) is developed. The procedure requires that we evaluate the Boolean expression for all possible combinations of values for the input variables.

This method of analysis is convenient and useful for logic circuits with fewer number of logic gates.

Example. 4-40. *Determine the Boolean expression for the logic circuit shown in Fig.* 4-19.

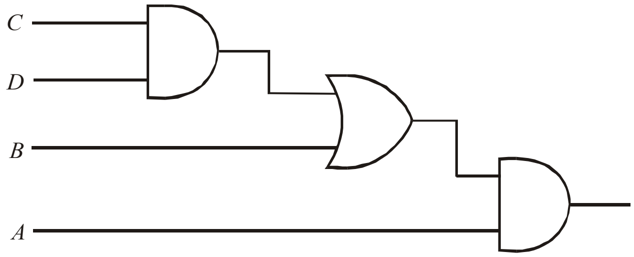


Fig. 4-19.

Analyse this circuit by evaluating the Boolean expression for all possible input combinations of values for the input variables.

Solution. Given : The logic circuit shown in Fig. 4-19.

Let us redraw the given logic circuit as shown in fig 4-20 and then determine the Boolean expression by beginning from the left most inputs and work towards the final output.

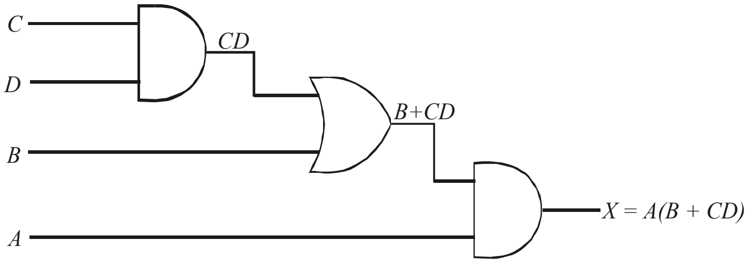


Fig. 4-20.

Thus we find that the expression for the left-most AND gate with inputs *C* and *D* is *CD*. Next the output of the left-most AND gate is one of the inputs to the OR gate and B is the other input. Therefore the expression for the OR gate is *B* + *CD*. Finally, the output of the OR gate is one of the inputs to the right-most AND gate and *A* is the other input. Therefore the expression for this AND gate is *A* (*B* + *CD*). This is the final output expression for the entire circuit. Thus the output expression for the given circuit,

$$X = A(B + CD) \textbf{ Ans.}$$

Let us now evaluate the expression *A* (*B* + *CD*). First, find the values of the variables that makes the expression equal to 1, using the laws and rules of Boolean algebra. In the present case, we find that the expression *A* (*B* + *CD*) = *1* only if *A* = 1 and *B* + *CD* = 1 because

$$A\ (B + CD) = 1.1 = 1$$

Now determine when the *B* + *CD* term equals 1. The term *B* + *CD* = 1 if either *B* = 1 or *CD* = 1 or if both *B* and *CD* equal 1 because

$$B + CD = 1 + 0 = 1$$

$$B + CD = 0 + 1 = 1$$

$$B + CD = 1 + 1 = 1$$

Next the term *CD* = 1 only if *C* = 1 and *D* = 1.

To summarize, the expression, *A* (*B* + *CD*) = 1 when *A* = 1 and *B* = 1 regardless of the values of *C* and *D* or when *A* = 1 and *C* = 1 and *D* = 1, regardless of the value of *B*. For all other possible combinations of the variables, the expression *A* (*B* + *CD*) = 0. The result is shown in the form of a truth table as shown in Fig. 4-21.

Inputs				*Output*
A	*B*	*C*	*D*	*A (B + CD)*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Fig. 4-21.

4-9. Simplification of Logic Circuits

We have already discussed in the last article that Boolean algebra is a powerful tool to analyse logic circuits, This is achieved by evaluating the Boolean expression for all possible combinations of values for the input variables.

Many times, in the application of Boolean algebra, we have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently using logic gates. Broadly speaking there are two methods to achieve this :

(1) Simplification using Boolean algebra and (2) Simplification using Karnaugh maps, Both these methods are discussed one by one in the following pages.

4-10. Simplification of Logic Circuits Using Boolean Algebra

We know that Boolean algebra can be used to simplify the Boolean expression for a given logic circuit. Such a simplification leads to the simplification of a logic circuit. This means that the logic circuit with a simplified Boolean expression contains fewer gates and therefore will be smaller and cheaper to implement than the original logic circuit.

Now we will illustrate the simplification method through examples (that follow in the subsequent pages).

Notice that the examples contain two essential steps.

1. The original Boolean expression is put into sum-of-products form by repeated application of Boolean laws, rules and theorems.
2. Once the original Boolean expression is in this form, the product terms are checked for common factors. The factory is performed whenever possible and usually it leads to elimination of one or more terms from the original Boolean expression. It may be noted that this simplification method depends on a thorough knowledge of Boolean algebra and considerable practice in its application.

Example 4-41. *Determine the Boolean expression for the output of the logic circuit shown in Fig.* 4-20.

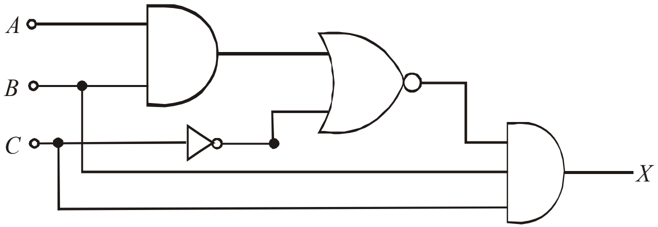


Fig. 4-22.

Solution. Given : The logic circuit shown in Fig. 4-22.

First of all, let us determine the Boolean expression for the output of the logic circuit, We know that the output of the left-most AND gate with inputs A and B is AB. Next the output of inverter is $\overline{C}$. Next the output of an OR gate with AB and $\overline{C}$ as its inputs is $\overline{(AB+\overline{C})}$. Finally, the output of right-most AND gate with $\left(\overline{AB+\overline{C}}\right)$, B and C as its inputs is,

$$X = BC\,(\overline{AB+\overline{C}}) \qquad ...(i)$$

Fig. 4-23 shows the resulting outputs at all the logic gate outputs.

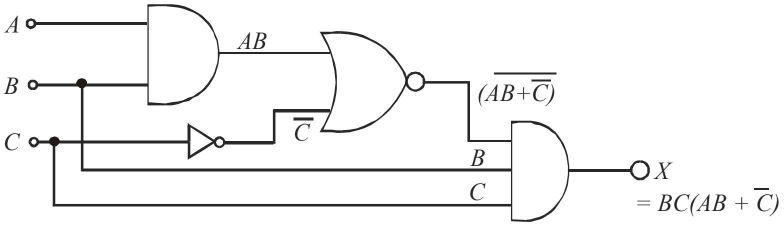


Fig. 4-23.

The Boolean expression in equation (i) can be simplified as follows by applying De Morgan's theorems. Boolean laws and rules

$$\begin{aligned}
X &= BC\,\overline{(AB+\overline{C})} \\
&= BC\,(\overline{AB}\,.\,\overline{\overline{C}}) \\
&= BC\,(\overline{AB}\,.C) && \text{—}(\because \overline{\overline{C}}=C) \\
&= BC\left[(\overline{A}+\overline{B})\right]C \\
&= BC\,(\overline{A}C+\overline{B}C) \\
&= \overline{A}BC+BC\,\overline{B}C \\
&= \overline{A}BC+0 && \text{—}(\because B\overline{B}=0) \\
&= \overline{A}BC && ...(ii)
\end{aligned}$$

The Boolean expression in equation (ii) can be implemented by an INVERTER and a three-input AND gate as shown in Fig. 4-24. Notice that the given logic circuit used four logic gates. But after simplification the logic circuit makes use of only two logic gates.

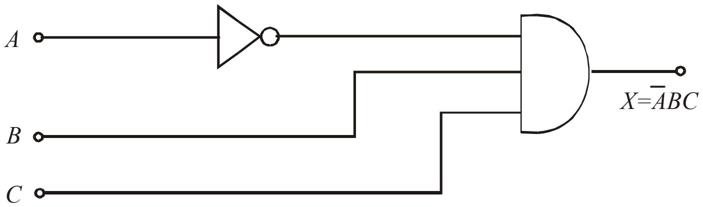
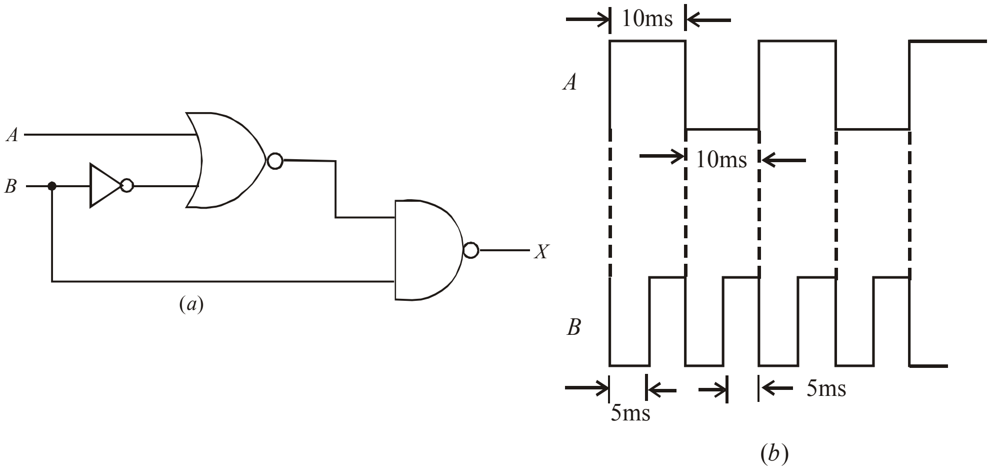


Fig. 4-24.

Example 4-42. *Fig.* 4-25 (*a*) *shows a logic circuit with the inputs A and B. The waveforms at the two inputs are also shown. in Fig.* 4-25 (*b*)



(*a*)

(*b*)

Fig. 4-25.

Determine the minimized circuit and then sketch the waveform at the output.

(*UPSC Engg. Services,* 1994)

Solution. Given : The logic circuit shown in Fig 4-25.

First of all, let us redraw the given logic circuit as shown in Fig. 4-26. Notice the input to the INVERTER gate is B and therefore its output is $\overline{B}$. Next input to NOR gate are A and $\overline{B}$. Therefore the output of this logic gate is $\overline{A+\overline{B}}$. Finally the inputs to the NAND gate are $\overline{A+\overline{B}}$ and B. Therefore the output of this logic gate is,

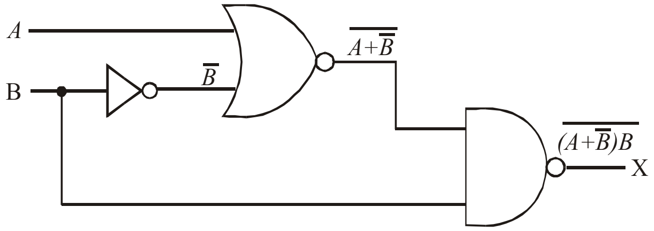

A
B
$\overline{B}$
$\overline{A+\overline{B}}$
$\overline{(\overline{A+\overline{B}})B}$
X

Fig. 4-26.

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

Applying the De Morgan's theorem '1' to the term on the right, we get,

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

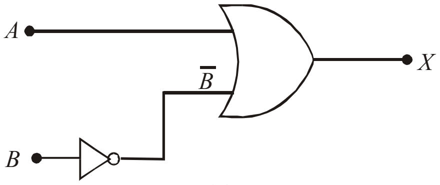
Applying Rule 9 : $(\overline{\overline{A}} = A)$ on the first term within parenthesis, we get,

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

Applying Rule 5 : (A + A = A) to the last two terms on the right, we get,

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

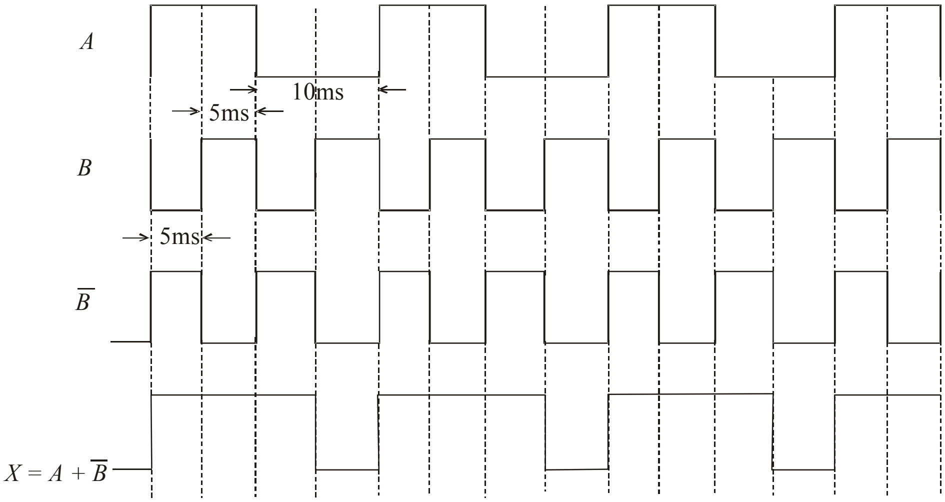
Thus the minimized circuit for the given logic circuit is as shown in Fig. 4-27 (*a*).



(*a*)

Fig. 4-27.

The resulting waveform at X is obtained by adding the waveforms at A and $\overline{B}$ and is as shown in Fig. 4-27.



(*b*)

Fig. 4-27.

Example 4-43. *Using Boolean algebra minimize the logic circuit shown in Fig.* 4-28.

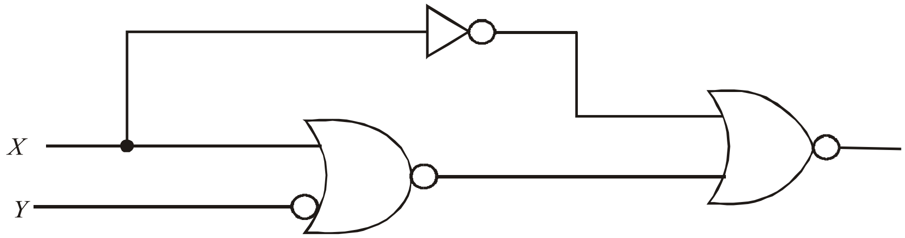


Fig. 4-28.

(*UPSC Engg. Services* 1995)

Solution. Given : The logic circuit shown in Fig 4-28.

First of all, let us redraw the given, circuit as shown in Fig 4-29. Notice that all the logic gates have been numbered for convenience. The output of INVERTER gate (1) is $\overline{X}$ and of logic gate (2) is $\overline{X+\overline{Y}}$. Finally the output of logic gate (3) is,

$$f = \overline{(\overline{X} + \overline{X+\overline{Y}})}$$

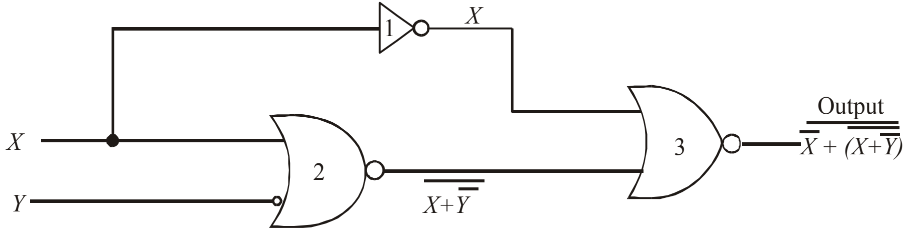

1
$\overline{X}$
X
Y
2
$\overline{X+\overline{Y}}$
3
Output
$\overline{\overline{X} + \overline{(X+\overline{Y})}}$

Fig. 4-29.

Applying De Morgan's theorem '2' to the right side, we get,

$$f = \overline{\overline{X}} . (\overline{\overline{X+\overline{Y}}})$$

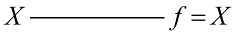


Fig. 4-29.

Applying Rule 9 : $(\overline{\overline{A}} = A)$ to the terms on the right we get,

$$f = X\,(X + \overline{Y})$$

Applying Distributive law, we get,

$$\begin{aligned} f &= X\,.\,X + X\overline{Y} \\ &= X + X\overline{Y} \\ &= X\,(1 + \overline{Y}) \\ &= X \ \textbf{Ans.} \end{aligned}$$

∴ A minimized logic circuit has no logic gate. The output is equal to the value of X itself as shown in Fig. 4-29. In other words the given logic circuit is redundant.

Example 4-44. *Determine the Boolean expression for the output, X of a logic circuit shown in Fig.* 4-30.

Minimize the Boolean expression for the output X using Boolean algebra.

(*UPSC Engg. Services,* 1997)

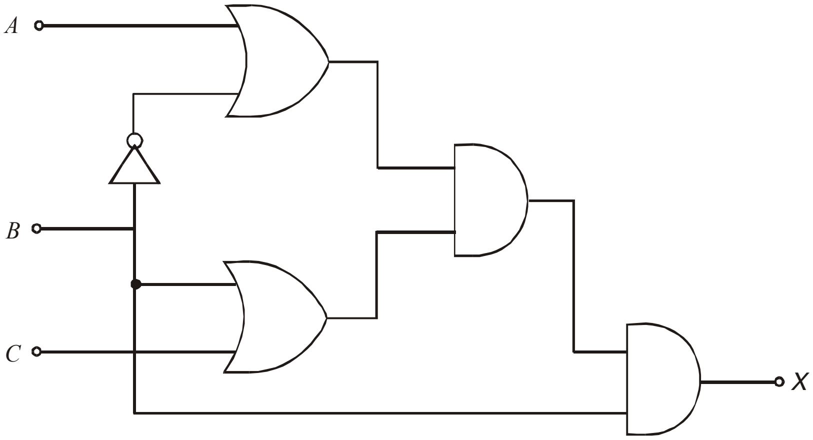


Fig. 4-30.

Solution. Given : The logic circuit shown in Fig. 4-30.

First of all, let us determine the Boolean expression for the output of the given logic circuit. The circuit has been redrawn as shown in Fig. 4-31. Notice that each logic gate has been assigned a number for convenience.

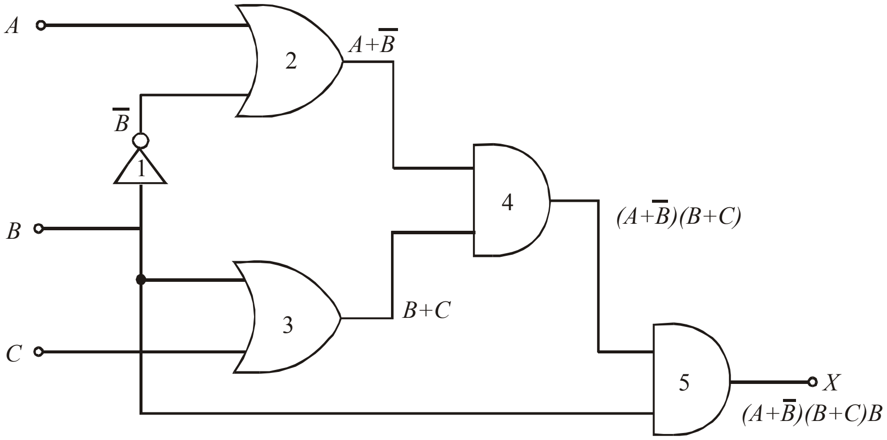


Fig. 4-31.

We know that the input to INVERTER gate (1), is B and therefore its output is $\overline{B}$. The inputs to the OR gate (2) are A and $\overline{B}$. Therefore its output is $A+\overline{B}$. The inputs to OR gate (3) are B and C. Therefore its output is $B + C$. Next the inputs to AND gate (4) are $(A+\overline{B})$ and $(B+C)$. Therefore its output is $(A+\overline{B})(B+C)$. Finally, the inputs to AND gate (5) are $(A+\overline{B})(B+C)$ and B.

Therefore its output,

$$X=(A+\overline{B})(B+C)\,B \qquad ...(i)$$

Let us now minimize this function by using laws and rules of Boolean algebra.

Applying distributing law of Boolean algebra to equation (i), we get,

$$X=(A+\overline{B})(BB+BC)$$

$$=(A+\overline{B})(B+BC) \qquad -(\because BB=B)$$

Applying Rule 10 : $(A+AB=A)$ to the term $(B+BC)$ we get,

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

Applying the distributive law again, we get,

$$X = AB + B\overline{B}$$

$$= AB \qquad -(\because B\overline{B}=0)$$

$\therefore$ The minimized Boolean expression for the given logic circuit,

$X=AB$ **Ans.**

Note. The minimized function, $X = AB$ can be implemented by using just one 2-input AND gate.

Example 4-45. *Determine the output, Y for the logic circuit shown in Fig.* 4-32.

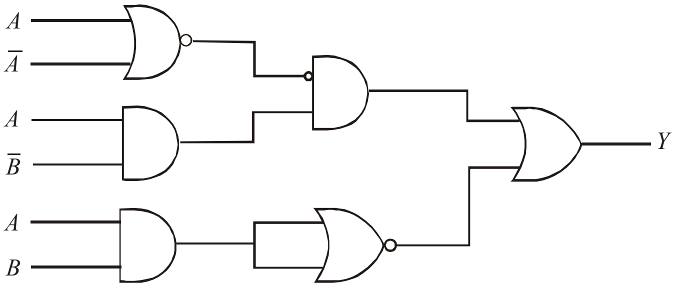


Fig. 4-32.

Simplify the Boolean expression using Boolean algebra laws, rules and theorems.

(*UPSC Engg. Services* 1998)

Solution. Given : The logic circuit shown in Fig. 4-32. First of all let us determine the Boolean expression for the output of the given logic circuit. The circuit has been redrawn as shown in Fig. 4-33 with each logic gate labelled with a number for convenience. We know that the output of NOR logic gate (1), is $\overline{A+\overline{A}}$.

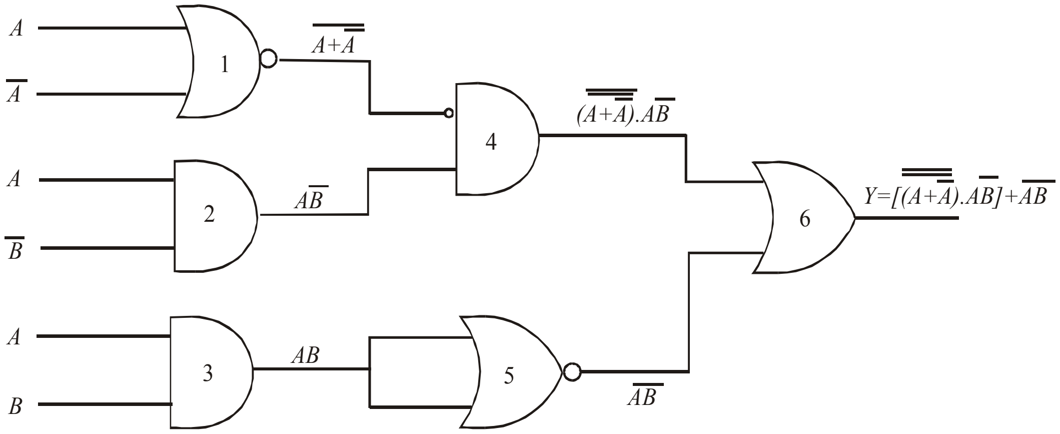

A
$\overline{A}$
1
$\overline{A+\overline{A}}$
A
$\overline{B}$
2
$A\overline{B}$
4
$\overline{\overline{(A+\overline{A})}}.A\overline{B}$
6
$Y=[\overline{\overline{(A+\overline{A})}}.A\overline{B}]+\overline{AB}$
A
B
3
AB
5
$\overline{AB}$

Fig. 4-33.

Next, the output of AND gate (2) is $A\overline{B}$ and the output of AND gate 3 is AB. Next it is the logic gate (4) with one of its input (from logic gate 1) is inverted and the other input is $A\overline{B}$. The output of this logic gate is

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

Next the NOR gate (5) is acting as an INVERTER because two of its inputs are shorted together. Therefore the output of this logic gate is $\overline{AB}$.

Finally, the output of OR gate (6) is,

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

Let us now simplify this Boolean expression using Boolean algebra.

Applying Rule 9 : $(\overline{\overline{A}}=A)$ to the term $\overline{\overline{(A+\overline{A})}}$, we get,

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

Applying Rule 6 : $(A+\overline{A}=1)$, to the first term on right within parenthesis, we get,

$$Y=\left[1.(A\overline{B})\right]+\overline{AB}$$

Applying Rule 2 : $(A.1=A)$ to the first term on the right, we get,

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

Applying De Morgan's theorem 2 to the second term on the right, we get,

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

Factoring out the variable $\overline{B}$,

$$Y=(A+1)\,\overline{B}+\overline{A}$$

Applying Rule 4 : (A + 1 = 1) to the first term on the right we get,

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

Example 4-46. *Determine the output for the logic circuit shown in Fig.* 4-34.

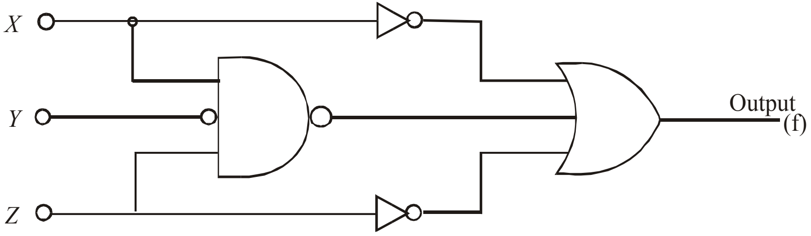


Fig. 4-34.

Minimize the output function using Boolean algebra and sketch the minimized logic circuit.
(*UPSC Engg. Services* 1996)

Solution. Given : The logic circuit shown in Fig. 4-34.

First of all, let us determine the Boolean expression for the given logic circuit. The circuit has been redrawn as shown in Fig. 4-35, with each logic gate labelled with a number for convenience. Notice the inputs to the logic gate (1), one of its inputs is *X*, the other is inverted *Y* (*i.e.*$\overline{Y}$) and the third input is *Z*. The output of this logic gate is $\overline{X\overline{Y}Z}$, Next the output of INVERTER (2) is $\overline{X}$ and that of INVERTER (3) is $\overline{Z}.$ Finally the output of OR gate (4).

$$f = \overline{X} + \overline{X\overline{Y}Z} + \overline{Z}$$

Now we will minimize this logic function using Boolean algebra.

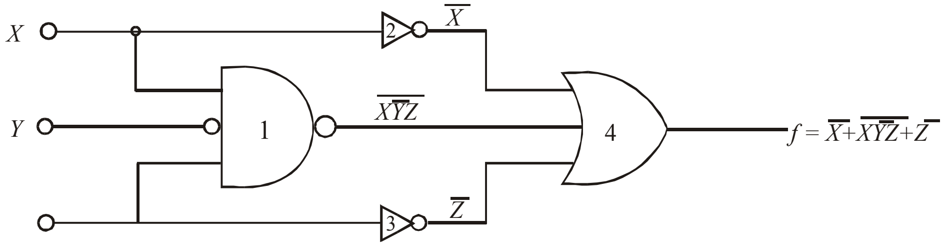

X
Y
2
$\overline{X}$
1
$\overline{X\overline{Y}Z}$
4
$f = \overline{X}+\overline{X\overline{Y}Z}+\overline{Z}$
3
$\overline{Z}$

Fig. 4-35.

We know that the output of the logic circuit,

$$f = \overline{X} + \overline{X\overline{Y}Z} + \overline{Z}$$

Applying De Morgan's theorem 1 to the second term on the right,

$$f = \overline{X} + (\overline{X} + \overline{\overline{Y}} + \overline{Z}) + \overline{Z}$$

Applying Rule 9 : $(\overline{\overline{A}} = A)$ to the second term on the right within parenthesis, we get,

$$f = \overline{X} + (\overline{X} + Y + \overline{Z}) + \overline{Z}$$

Applying Rule 5 : $(A + A = A)$ to all the terms on the right,

$$f = (\overline{X} + \overline{X}) + Y + (\overline{Z} + \overline{Z})$$

$$= \overline{X} + Y + \overline{Z}$$

Using De Morgan's theorem 1 in the reverse order, this can be rewritten as

$$f = \overline{X\overline{Y}Z}$$

This function can be implemented by a three-input NAND gate as shown in Fig. 4-36.

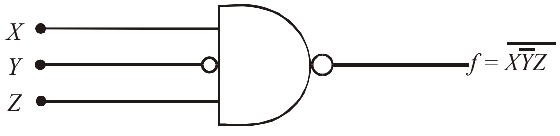


Fig. 4-36.

Note : If we compare the logic circuit of Fig. 4-36 with that of Fig. 4-34, we find that the two INVERTERs and the three input OR gate are redundant.

Example 4-47. *The circuit shown in Fig.* 4-36 *is used to implement the function Z = f (A, B) = A + B. What values should be selected for I and J.* (*UPSC Engg. Services,* 1993)

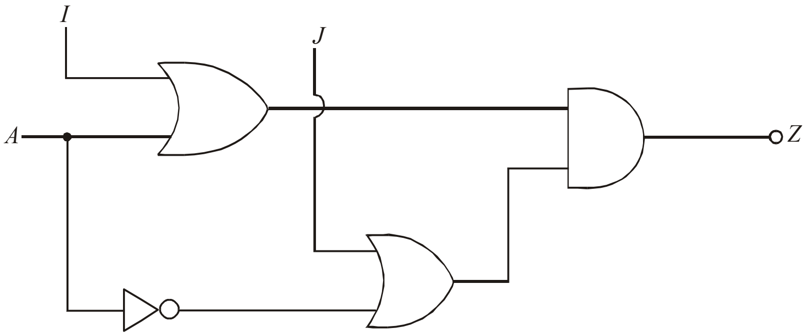


Fig. 4-37.

Solution. Given : The logic circuit of Fig. 4-37.

First of all let us, redraw the circuit as shown in Fig. 4-38. Notice that each logic gate has been assigned a number for convenience.

We know that output of gate 1 (OR gate), $= A + I$

and output of gate 2 (NOT gate), $= \overline{A}$

We also know that output of gate 3 (OR gate), $= \overline{A} + J$

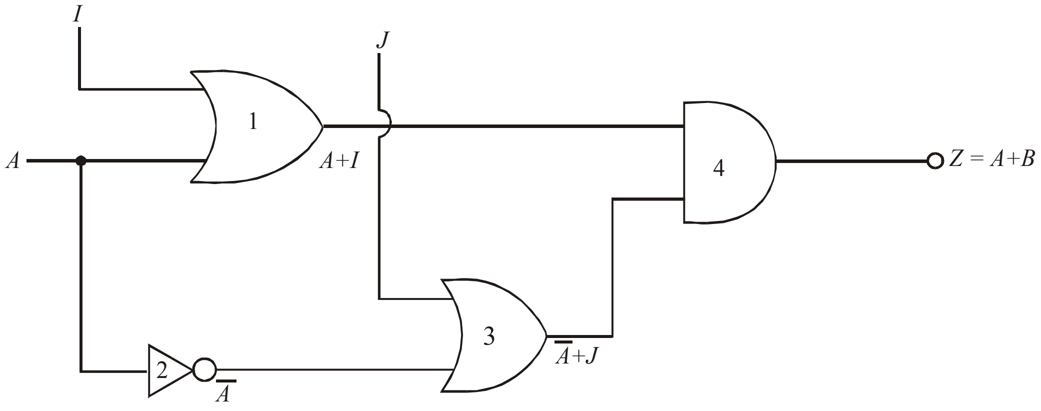


Fig. 4-38.

and output of gate 4 (AND gate) $Z = (\overline{A} + J).(A + I)$

$$= \overline{A}A + JA + \overline{A}I + JI$$

$$= 0 + JA + \overline{A}I + JI \qquad -(\because A\overline{A} = 0)$$

Since the output Z is given to be equal to $A + B$, therefore

$$A + B = JA + \overline{A}I + JI \qquad ?$$

Comparing the terms on left hand and right hand side in the above equation we find, that,

$$JA = A \qquad ...(i)$$

and $$B = \overline{A}I + JI \qquad ...(ii)$$

Fern equation (*i*), we find that if $J = 1$ then $1.A = A$. Therefore $J = 1$

Fren equation (*ii*), we find that, we find that

$$B = (\overline{A} + J)I = (\overline{A} + 1)I = 1.I = I$$

Thus $J = 1$ and $I = B$ **Ans.**

Example 4-48. *Determine the logic function realized by the circuit shown in Fig.* 4-39.

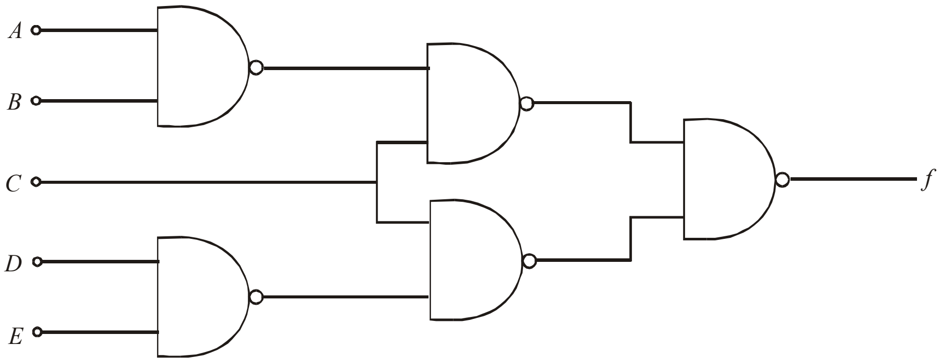


Fig. 4-39.

Minimize the output function, f using Boolean algebra.

(*UPSC Engg. Services,* 1996)

Solution. Given : The logic circuit shown in Fig. 4-39.

First of all, let us redraw the given logic circuit as shown in Fig. 4-40. Notice that all the logic gates have been assigned numbers for the convenience.

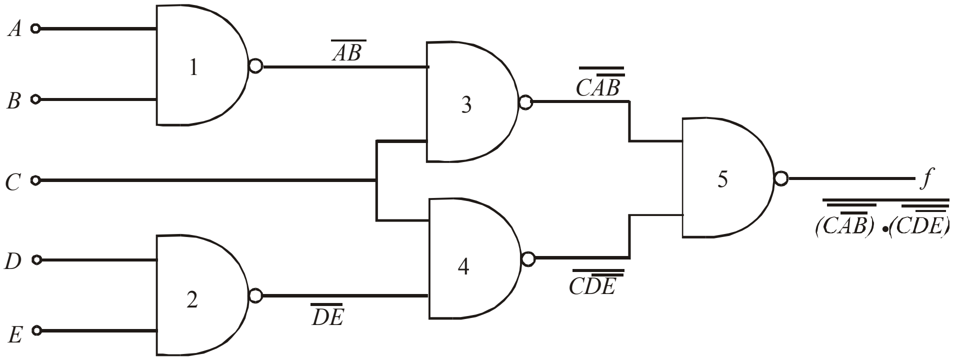

A
B
C
D
E
1
2
3
4
5
$\overline{AB}$
$\overline{C\overline{AB}}$
$\overline{C\overline{DE}}$
$\overline{DE}$
f
$\overline{(\overline{C\overline{AB}}) \cdot (\overline{C\overline{DE}})}$

Fig. 4-40.

We know that NAND gate (1) has its inputs A and B. The output of this logic gate is $\overline{AB}$. Similarly, the NAND gate 2 has D and E as its inputs. Therefore the output of this logic gate is $\overline{DE}$. Next, the NAND gate (3) has $(\overline{AB})$ and C as its inputs. The output of this logic gate is $\overline{C\overline{AB}}$. Similarly, the NAND gate (4), has $\overline{DE}$ and C as its inputs. Therefore, the output of this logic gate is $\overline{C\overline{DE}}$. Finally, the NAND gate (5) has $(\overline{C\overline{AB}})$ and $(\overline{C\overline{DE}})$ as its inputs. The output of this logic gate is,

$$f = \overline{(\overline{C\overline{AB}}).(\overline{C\overline{DE}})}$$

Let us minimize this function using Boolean algebra. Applying De Morgan's theorem '1' to the terms within parenthesis, we get,

$$f = \overline{\overline{C\overline{AB}}} + \overline{\overline{C\overline{DE}}}$$

Applying Rule 9 : $(\overline{\overline{A}} = A)$ to both the terms on the right, we get,

$$f = C\overline{AB} + C\overline{DE} \qquad ...(ii)$$

Applying De Morgan's theorem '1' again to the terms $\overline{AB}$ and $\overline{DE}$, we get,

$$\begin{aligned} f &= C(\overline{A}+\overline{B})+C(\overline{D}+\overline{E}) \\ &= C\left\{\overline{A}+\overline{B}+\overline{D}+\overline{E}\right\} \qquad ...(iii) \end{aligned}$$

Note : If we implement equation (*iii*) using NAND gates we can do so by using only 3 logic gates, as shown below :

$$f = C(\overline{A}+\overline{B}+\overline{D}+\overline{E}) = C(\overline{ABDE})$$

The circuit to implement the logic function $f = C(\overline{ABDE})$ is as shown in Fig. 4-41. But the limitation of this design is that we need one four-input NAND gate and two 2-input NAND gates. This means we need two ICs-as the four input NAND gate is not available in the same IC as that of 2-input NAND gate. So we conclude, the logic circuit shown in Fig. 4-39 could be a better choice for hardware implementation.

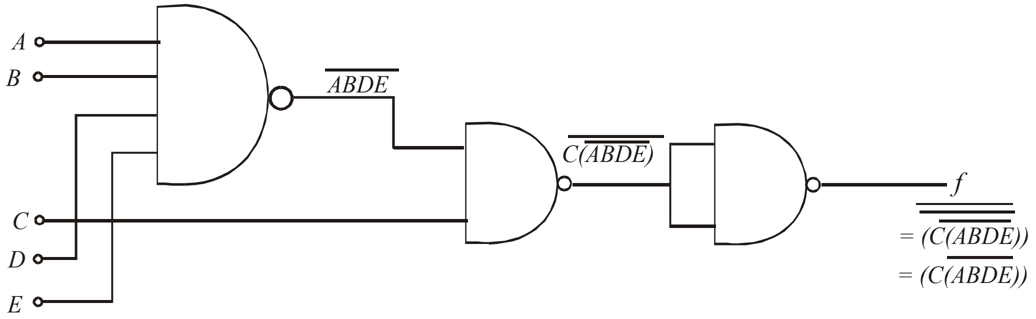

A
B
C
D
E
$\overline{ABDE}$
$\overline{C(\overline{ABDE})}$
f
$= \overline{\overline{(C(\overline{ABDE}))}}$
$= (C(\overline{ABDE}))$

Fig. 4-41.

Example 4-49. *Fig.* 4.42 *shows a combinational logic circuit. Obtain a simplified Boolean expression for this circuits at V, W, X, Y and Z in terms of input variables A, B and C.*

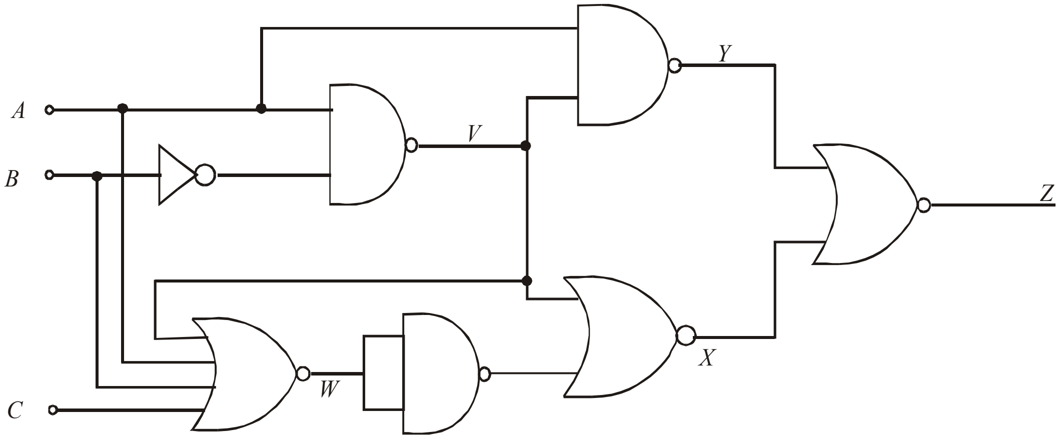


Fig. 4-42.

(*UPSC Engg. Services* 1995)

Solution. Given : The logic circuit shown in Fig. 4-42.

First of all redraw the given logic circuit as shown in Fig. 4-43. Notice that all the logic gates have been assigned numbers for convenience.

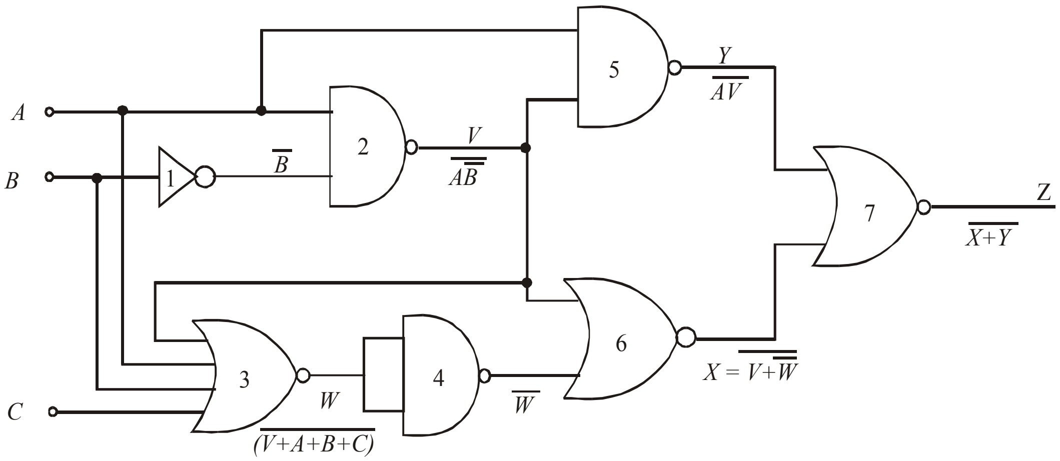


Fig. 4-43.

We know that the input to INVERTER gate (1) is B, therefore its output is $\overline{B}$. Next the inputs to NAND gate (2) are A and $\overline{B}$. Therefore its output,

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

Applying De Morgan's theorem '1' to the term on the right, we get,

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

$$= \overline{A} + B \text{ **Ans.**} \qquad ...(i)$$

Next the inputs to 4-input NOR gate (3) are V, A, B and C. Therefore its output,

$$W = \overline{V + A + B + C} \qquad ...(ii)$$

Substituting $V\,(=\overline{A} + B)$ from equation (i) into equation (ii) we get,

$$W = \overline{\left(\overline{A} + B\right) + A + B + C} = \overline{\left(\overline{A} + A\right) + \left(B + B\right) + C}$$

Applying Rule 5 : $\left(A + A = A\right)$ and Rule 6 : $\left(A + \overline{A} = 1\right)$ to be term on the right-hand side of the above equation, we get,

$$W = \overline{1 + B + C}$$

Applying Rule 4 : $\left(A + 1 = 1\right)$ to the term on the right hand side we get,

$$W = \overline{1}$$

$$= 0 \text{ **Ans.**} \qquad ...(iii)$$

Next the input to NAND gate (4) is W. Notice that it is wired as an INVERTER. Therefore its output is $\overline{W}$.

The inputs to NAND gate (5) are V and A. Therefore its output,

$$Y = \overline{AV} \qquad ...(v)$$

Substituting the value of V from equation (i) into equation (v) we get,

$$Y = \overline{AV} = \overline{A\,(\overline{A} + B)}$$

Applying the distributive law to the term on the right, we get,

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

Applying Rule 8 : $\left(A.\overline{A} = 0\right)$ to the first term on the right-hand side of the above equation, we get,

$$y = \overline{O + AB}$$

$$= \overline{AB} \qquad \textbf{Ans.} \qquad \text{...}(v)$$

The inputs to the NOR gate (6) V are $\overline{W}$, Therefore the output of this logic gate,

$$X = \overline{V + \overline{W}}$$

Substituting the values of $V\left(= \overline{A} + B\right)$ and $W\left(= 0\right)$ from equation (*i*) and (*iii*), we get,

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

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

Applying Rule : 4 $\left(A + 1 = 1\right)$ to the term on the right-hand side of the above equation,

$$X = \overline{1}$$

$$= 0 \quad \textbf{Ans.} \qquad \text{...}(vi)$$

Finally the inputs to NOR gate (7) are Y and X, therefore the output of this logic gate is

$$Z = \overline{X + Y} \qquad \text{...}(viii)$$

Substituting the value of $X = 0$ and $Y = \overline{AB}$ into equation (*viii*), we get,

$$Z = \overline{0 + \overline{AB}}$$

Applying Rule 3 : (A + 0 = A) to the term on the right, we get,

$$Z = \overline{\overline{AB}}$$

Applying Rule 9 : $(\overline{\overline{A}} = A)$ to the term on the right, we get,

$$Z = AB \ \textbf{Ans.}$$

4-11. Standard Forms of Boolean Expressions

Strictly speaking, all Boolean expressions, regardless of their form, can be converted into either of two standard forms : **sum-of-products (SOP) form** or the **product-of-sums (POS) form**. Both these standards forms of Booleans expression are discussed one by one in the following pages.

4-12. The Sum-of-Products (SOP) Form

In Boolean algebra, **a product term** (also called min-term) is the product of literals (or Boolean variables). In logic circuits, a product term is produced by AND operation with no OR operations. Some examples of product terms are :

$$AB, A\overline{B}, AB\overline{C}, A\overline{B}C\overline{D}$$

A product term is equal to 1 if only if each of the literals in the term is 1. A product term is equal to 0 when one or more of the literals are 0.

When two or more product terms are ORed (i.e. summed by Boolean addition), the resulting expression is called Sum-of-Products (SOP). Some examples of sum-of-products are :

$$AB + BCD$$

$$ABC + CDE + \overline{B}C\overline{D}$$

$$A\overline{B} + \overline{A}B\overline{C} + AC$$

It may be carefully noted that sum-of-products expression can contain a single variable term. However, in a sum-of-products expression, a single overbar cannot extend over more than one variable, although more than one variable in a term can have an overbar. For example, $\overline{A}B\overline{C}$ can be a term in sum-of-products expression but not $\overline{ABC}$.

The sum of product express as, in the form of minterm

$$= m_0 + m_1 + m_2 + \ldots m_n$$

where $m_0, m_2 \ldots$ are the minterms.

4-13. Domain of a Boolean Expression

The **domain** of a Boolean expression is the set of variables contained in the expression in either complemented or uncomplemented form. For example, the domain of the expression $A\overline{B} + \overline{A}B\overline{C}$ is the set of variables, *A*, *B* and *C*. Similarly the domain of $AB + BCD + C\overline{D} + D$ is the set of variables. *A*, *B*, *C* and *D*.

4-14. Implementation of Sum-of-Products (SOP) Expression

In order to implement a sum-of-products expression, we require ORing the outputs of two of more AND gates. A product term is produced by an AND operation and the addition of two or more product terms is produced by an OR operation. Therefore sum-of-products expression can be implemented by AND-OR logic.

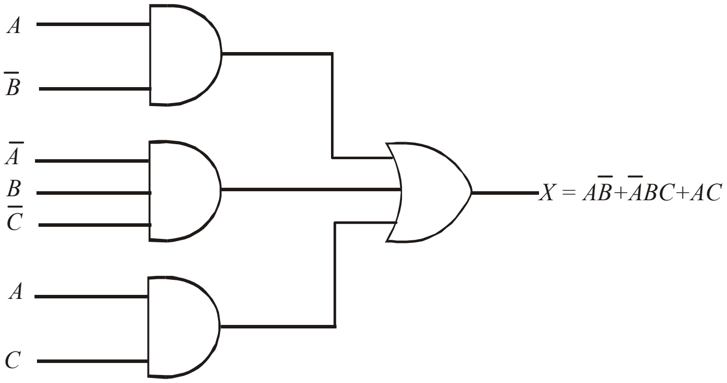


Fig. 4-44. Implementation of sum-of-products expression $A\overline{B} + \overline{A}BC + AC$ using AND-OR logic.

Fig. 4-44 shows the implementation of sum-of-products expression $A\overline{B} + \overline{A}BC + AC$. Notice that the number of AND gates is equal to the number of terms in the sum-of-products expression. The outputs of AND gates feed into the inputs of an OR gate. The output *X* of the OR gate equal the sum-of-products expression.

4-15. Conversion of a General Expression to Sum-of-Products (SOP) Form

It will be interesting to know that any logic expression can be changed into sum-of-products form by applying Boolean algebra techniques. For example, the expression *A* (*BC* + *D*) can be converted to sum-of-products form by applying the distributive law :

$$A(BC + D) = ABC + AD$$

Example. 4-50. *Convert each of the following Boolean expressions to sum-of-products form* :

(*a*) $(A + B)(C + \overline{B})$ (*b*) $(A + \overline{B}C)C$ (*c*) $(A + C)(B + AC)$

Solution. (*a*) Given, the Boolean expression, $(A + B)(C + \overline{B})$

We know that the given Boolean expression can be converted to sum-of-products form by applying the distributive law,

$$\begin{aligned}(A+B)(C+\overline{B}) &= AC+A\overline{B}+BC+B\overline{B}\\ &= AC+A\overline{B}+BC+0 \qquad \ldots(\because\ B\overline{B}=0)\\ &= AC+A\overline{B}+BC \ \ \textbf{Ans.}\end{aligned}$$

(*b*) Given, the Boolean expression $(A+\overline{B}C)C$

We know that the given Boolean expression can be converted to sum-of-products form by applying the distributive law,

$$\begin{aligned}(A+\overline{B}C)\,C &= AC+\overline{B}CC\\ &= AC+\overline{B}C \qquad \ldots(\because\ C.C=C)\end{aligned}$$

(*c*) Given, the Boolean expression, $(A + C)\,(B + AC)$

We know that the given Boolean expression can be converted to sum-of-products form by applying the distributive law,

$$\begin{aligned}(A + C)\,(B + AC) &= AB + AAC + BC + ACC\\ &= AB + AC + BC + AC \qquad \ldots(A\,.\,A = A\ ;\ C\,.\,C = C)\\ &= AB + AC + BC \ \ \textbf{Ans.} \qquad \ldots(AC + AC = AC)\end{aligned}$$

4-16. The Standard Sum-of-Products (SOP) Form

A standard sum-of-products (SOP) expression is one in which all the variables in the domain appear in each product term of the expression. For example, $AB\overline{C}D+\overline{A}\overline{B}C\overline{D}+\overline{A}B\overline{C}D$ is a standard sum-of-products form because all the four variables *A*, *B*, *C* and *D* appear in each product term in the expression.

On the other hand, a sum-of-products expression in which some of the product terms do not contain all of the variables in the domain, is called non-standard sum-of-products expression. For example, the expression $\overline{A}B\overline{C}+A\overline{B}D+AB\overline{C}D$ has a domain made up of four variables *A*, *B*, *C* and *D*. From the expression find that the first two terms (i.e. $\overline{A}B\overline{C}$ and $A\overline{B}D$) does not have a complete set of four variables in the domain. To be more precise, we find that from the first term $\overline{A}B\overline{C}$, either D or $\overline{D}$ is missing. Similarly from the second term $A\overline{B}D$, either *C* or $\overline{C}$ is missing.

Standard sum-of-product expressions are important in constructing truth tables and in Karnaugh-map simplification method. Any non-standard sum-of-products expression can be converted to the standard form using laws and rules of Boolean algebra.

4-17. Converting Product Terms to a Standard Sum-of-Products (SOP) Form

We have already discussed in the last article that any non-standard sum-of-products expression can be converted to the standard form using laws and rules of Boolean algebra. Following is the procedure for converting a non-standard sum-of-products expression to a standard form.

1. Identify the non-standard product terms in the given expression. Multiply each such product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As we know that we can multiply anything by 1 without changing its value.

2. Repeat step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to a standard form, the number of product terms is doubled for each missing variable.

Let us illustrate the procedure through the following example. Consider the conversion of Boolean expression $A\overline{B}C+\overline{A}B+AB\overline{C}D$ to a standard sum-of-products form.

We know that the domain of the given sum-of-product expression is A, B, C and D. From the expression. We find that the first two product terms i.e. $A\overline{B}C$ and $\overline{A}B$ are not in the standard form, while the third term $AB\overline{C}D$ is already a standard product term.

For convertion of a non-standard product terms to a standard form, let us take one term at a time.

The first term $A\overline{B}C$ has a missing variable D or $\overline{D}$. So multiply the first term by $D+\overline{D}$ as follows :

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

From the above expression we find that a single non-standard product term $A\overline{B}C$ has produced two standard product terms, $A\overline{B}CD$ and $A\overline{B}C\overline{D}$.

It may be noted that a variable added to its complement equals 1 $(i.e. D+\overline{D}=1)$ -*i.e.* Rule 6 of Boolean algebra (refer to table 4-1). Therefore there is no violation in the way we have converted a non-standard product term to a standard form.

The second term $\overline{A}B$ has a missing variable C or $\overline{C}$ and D or $\overline{D}$. In order to convert this non-standard product term to a standard form, we will multiply $\overline{A}B$, first by $C+\overline{C}$, and then by $D+\overline{D}$. Thus first multiplying $\overline{A}B$ by $C+\overline{C}$, we get,

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

Multiplying the two resulting terms by $D+\overline{D}$, we get,

$$\begin{aligned}\overline{A}B\overline{C}+\overline{A}B\overline{C}&=\overline{A}BC\,(D+\overline{D})+\overline{A}B\overline{C}\,(D+\overline{D})\\&=\overline{A}BCD+\overline{A}BC\overline{D}+\overline{A}B\overline{C}D+\overline{A}B\overline{C}\overline{D}\end{aligned}$$

or
$$\overline{A}B=\overline{A}BCD+\overline{A}BC\overline{D}+\overline{A}B\overline{C}D+\overline{A}B\overline{C}\overline{D}$$

Thus we find that a non-standard term $\overline{A}B$ has produced four product terms in the standard form.

The standard sum-of-products (SOP) form of the given expression,

$$A\overline{B}C+\overline{A}B+AB\overline{C}D=A\overline{B}CD+A\overline{B}C\overline{D}+\overline{A}BCD+\overline{A}BC\overline{D}+\overline{A}B\overline{C}D+\overline{A}B\overline{C}D+AB\overline{C}D$$

Example 4-51. *Convert the Boolean expression* $A\overline{B}C+\overline{B}C\overline{D}+A\overline{C}D$ *to a standard sum-of-products* (*SOP*) *form.*

Solution. Given, The Boolean expression $A\overline{B}C+\overline{B}C\overline{D}+A\overline{C}D$

We know that the domain of the given expression is A, B, C and D, From the expression we find that all the three product terms are in the non-standard form. Let us convert these product terms to a standard form taking one term at a time.

The first term $A\overline{B}C$ has a missing variable D or $\overline{D}$. So multiply the first term by $D+\overline{D}$ as follows,

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

The second term $\overline{B}C\overline{D}$ has a missing variable A or $\overline{A}$. So multiply the second term by $A+\overline{A}$ as follows,

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

The third term $A\overline{C}D$ has a missing variable B or $\overline{B}$. So multiply the third term by $B+\overline{B}$ as follows,

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

Thus the standard SOP (*i.e.* sum-of-products) form of the given expression,

$$A\overline{B}C + \overline{B}C\overline{D} + A\overline{C}D = A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}C\overline{D} + AB\overline{C}D + A\overline{B}\,\overline{C}D \text{ **Ans.**}$$

4-18. Binary Representation of a Standard Product Term

Strictly speaking, a standard product term is equal to 1 for only one combination of variable values. For example the product term $A\overline{B}C\overline{D}$ is equal to 1 when A = 1, B = 0, C = 1 and D = 0 as shown below. The product term $A\overline{B}C\overline{D}$ is 0 for all other combinations of values for the variables.

$$A\overline{B}C\overline{D} = 1.\overline{0}.1.\overline{0} = 1.1.1.1 = 1$$

It may be noted that the product term $A\overline{B}C\overline{D}$ has a binary value 1010 (i.e. decimal 10).

The student must remember that a product term is implemented with an AND gate whose output is 1 if and only if each of its inputs is 1. INVERTERS are used to produce the complements of the variables wherever required.

A sum-of-products (SOP) expression is equal to 1 if and only if one or more of the sum terms is equal to 1.

Example 4-52. *Determine the binary values for which the following sum-of-products* (*SOP*) *expression is equal to* 1,

$$A\overline{B}CD + AB\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Solution. Given, the SOP expression,

$$A\overline{B}CD + AB\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

We know that the given SOP expression is equal to 1 when any or all of the three product terms are equal to 1.

The product term $A\overline{B}CD$ is equal to 1 when A = 1, B = 0, C = 1 and D = 0, *i.e.*,

$$A\overline{B}CD = 1.\overline{0}.1\overline{0} = 1.1.1.1 = 1$$

The term $AB\overline{C}D$ is equal to 1 when A = 1, B = 1, C = 0 and D = 1, *i.e.*,

$$AB\overline{C}D = 1.1.\overline{0}.1 = 1.1.1.1 = 1$$

The term $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ is equal to 1 when A = 0, B = 0, C = 0 and D = 0, *i.e.*,

$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = \overline{0}.\overline{0}.\overline{0}.\overline{0} = 1.1.1.1 = 1$$

4-19. The Product-of-Sums (POS) Form

In Boolean algebra, a **sum term** (also called a max-term) is the sum of literals (or Boolean variables). In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of such terms are :

$$A+B,\, A+\overline{B},\, \overline{A}+B+C \text{ and } \overline{A}+B+\overline{C}+D.$$

A sum term is equal to 1 when one or more of the literals in the term are 1. A sum term is equal to 0 if and only if each of the literals is 0.

When two or more sum terms are multiplied, the resulting expression is a **product-of-sums (POS).** Some examples of product-of-sums form are :

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

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

$$(\overline{A}+B+\overline{C})(C+\overline{D}+E)(B+C+\overline{D})$$

It may be carefully noted that product-of-sums (POS) expression can contain a single variable term. However in product-of-sums (POS) expression, a single overbar cannot extend over more than one variable, although more than one variable in a term can have an overbar. For example, a product-of-sums (POS) expression can have the term $\overline{A}+B+\overline{C}$ but not $\overline{A+B+C}$.

The product of sums can be express in form of Maxterms,

$$= M_0 M_1 M_2 \ldots M_n$$

where M_0, M_1 ... are the maxterms.

4-20. Implementation of a Product-of-Sums (POS) Expression

In order to implement a product-of-sums (POS) expression, we require ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation and the product of two or more sum terms is produced by an AND operation. Therefore, a product-of-sums expression can be implemented by AND-OR logic.

Fig. 4-45 shows the implementation of product-of-sums expression, $(A+\overline{B})$ $(B+\overline{C}+D)$ $(\overline{A}+C)$. Notice that the number of OR gates is equal to the number of terms in the product-of-sums expression. The output of OR gates feed into the inputs of an AND gate. The output X of the AND gate equals the product-of-sums expression.

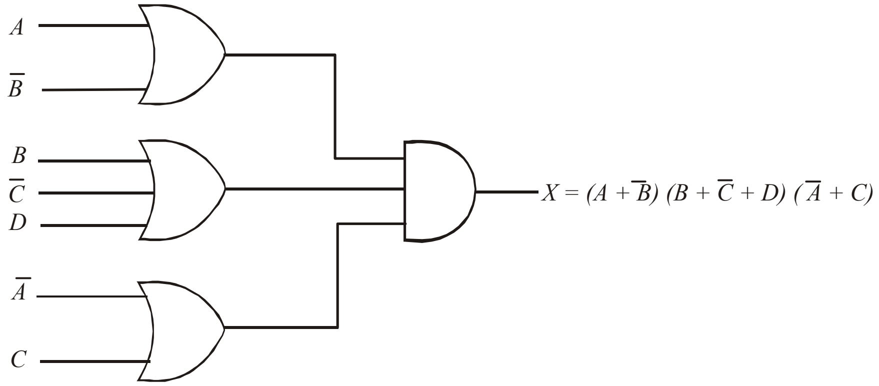

A
$\overline{B}$
B
$\overline{C}$
D
$\overline{A}$
C
$X = (A + \overline{B})\ (B + \overline{C} + D)\ (\overline{A} + C)$

Fig. 4-45. Implementation of a product-of-sums expression, $(A+\overline{B})(B+\overline{C}+D)(\overline{A}+C)$ using AND-OR logic.

4-21. The Standard Product-of-Sums (POS) Form

A standard product-of-sums (POS) expression is one in which all the variables in the domain appear in each sum term of the expression. For example, the expression,

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

is a standard product-of-sums expression.

On the other hand, a product-of-sums expression, in which some of the sum terms do not contain all of the variables in the domain, is called non-standard product-of-sums expression, For example the expression,

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

is a non-standard product-of-sums expression, because all the three sum terms do not have all the four variables.

Any non-standard product-of-sums expression can be converted to the standard form using laws and rules of Boolean algebra.

4-22. Converting a Sum Term to a Standard Product-of-Sums (POS) Form

We have already discussed in the last article that any non-standard product-of-sums expression can be converted to the standard form using laws and rules of Boolean algebra. Following is the procedure for converting a non-standard product-of-sums expression to a standard form.

1. Identify the non-standard sum terms in the given expression. To each non-standard sum term, add a product term consisting of a missing variable and its complement. This results in two sum terms. As we know we can add 0 to anything without changing its value.
2. Apply rule 12 of the Boolean algebra from Table 4-1 : $A+BC=(A+B)(A+C)$
3. Repeat step 1 until all resulting sum terms contain all variables in the domain in either complemented or uncomplemented form.

Let us illustrate the procedure by the following example, consider the conversion of the Boolean expression,

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

into a standard product-of-sums (POS) form,

We know that the domain of the given product-of-sums expression is A, B, C and D. From the expression, we find that the first two sum terms, i.e. $(\overline{A}+B+\overline{C})$ and $(B+\overline{C}+D)$ are not a standard POS form while the third term $(A+\overline{B}+\overline{C}+D)$ is already a standard term. For conversion of the non-standard sum terms to a standard form, let us take one term at a time.

The first term $\overline{A}+B+\overline{C}$ has a missing variable D or $\overline{D}$. So we add $D\overline{D}$ in this term and apply rule 12 of the Boolean algebra (From Table 4-1) as follows :

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

The second term $B+\overline{C}+D$ has a missing variable A or $\overline{A}$. So we add $A\overline{A}$ and apply rule 12 as follows :

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

Thus the standard product-of-sums form of the given Boolean expression,

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

Example 4-53. *Convert the Boolean expression* $(A+\overline{B})(B+C)$ *to a standard product-of-sums form.*

Solution. Given the Boolean expression, $(A+\overline{B})(B+C)$

We know that the domain of the given expression is A, B and C. From the expression, we find that both the sum terms *i.e.* $(A+\overline{B})$ and $(B+C)$ are in the non-standard form. Let us convert these sum terms to a standard form taking one term at a time.

The first term $(A+\overline{B})$ has a missing variable C or $\overline{C}$. So me add $C\overline{C}$ and apply rule 12 as follows.

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

The second term $(B + C)$ has a missing variable A or $\overline{A}$. So we add $A\overline{A}$ and apply rule 12 as follows.

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

Thus the standard product-of-sums form of the given Boolean expression,

$$(A+\overline{B})(B+C)=(A+\overline{B}+C)(A+\overline{B}+\overline{C})(A+B+C)(\overline{A}+B+C)$$ **Ans.**

Example 4-54. *Explain the following function in standard POS form :*

$$f(A, B, C) = AB + BC.$$

Solution. Given: the POS expression

$$f(A, B, C) = AB + BC = B\,(A + C)$$

We know that the domain of the given expression is A, B and C. From the expression, we find that both the sum terms *i.e.* B and $(A + C)$ are in the non-standard form. Let us convert these sum terms to a standard form taking one term at a time.

The first term B has a missing variable A or $\overline{A}$ and C an $\overline{C}$. So we add $A\overline{A}$ and apply rule 12 as follows,

$$B = B + AA' = (B + A)\ (B + \vec{A})$$

Both term has missing variable C or $\overline{C}$. So we add $C\overline{C}$ and apply rule 12

$$\begin{aligned} B = B + AA' + CC' &= (B + A + \overline{CC})(B + \overline{A} + C\overline{C}) \\ &= (B + A + C)(B + A + \overline{C})(B + \overline{A} + C)\ (B + \overline{A} + \overline{C}) \\ &= (A + B + C)(A + B + \overline{C})(\overline{A} + B + C)\ (\overline{A} + B + \overline{C}). \end{aligned}$$

The second term $A + C$ has a missing variable B or $\overline{B}$. So we add $B\overline{B}$ and apply rule 12 as follows.

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

Thus the standard POS form of the given Boolean expression.

$$AB + BC = (A + B + C)\ (A + B + \overline{C})\ (\overline{A} + B + C)\ (\overline{A} + B + \overline{C})\ (A + \overline{B} + C)\,.$$

4-23. Binary Representation of a Standard Sum Term

Strictly speaking, a standard sum term is equal to 0 for only one combination of variable values. For example, the sum term $A+\overline{B}+C+\overline{D}$ is 0 when $A = 0$, $B = 1$, $C = 0$ and $D = 1$ as shown below. The sum term $A+\overline{B}+C+\overline{D}$ is 1 for all other combinations of values for the variables,

$$A+\overline{B}+C+\overline{D}=0+\overline{1}+0+\overline{1}=0+0+0+0=0$$

It may be noted that the sum term has a binary value 0101 (*i.e.* decimal 5).

The student must remember that a sum term is implemented with an OR gate whose output is 0 if and only if each of its inputs is 0. INVERTERs are used to produce the complements of the variables wherever required.

A product-of-sums (POS) expression is equal to 0 if and only if one or more of the sum terms in the expression is equal to 0.

Example 4-55. *Determine the binary values of the variables for which the following standard product-of-sums (POS) expression is equal to* 0 :

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

Solution. Given, the POS expression,

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

We know that the given POS expression equals 0 when any of the three sum terms equals 0. The term $A+\overline{B}+C+D$ is equal to 0, when $A = 0$, $B = 1$, $C = 0$ and $D = 0$, *i.e.*

$$A+\overline{B}+C+D = 0+\overline{1}+0+0 = 0+0+0+0 = 0 \quad \textbf{Ans.}$$

The term $A+B+\overline{C}+D$ is equal to 0 when $A = 0$, $B = 0$, $C = 1$ and $D = 0$ *i.e.*

$$A+B+\overline{C}+D = 0+0+\overline{1}+0 = 0+0+0+0 = 0 \quad \textbf{Ans.}$$

The term $\overline{A}+\overline{B}+\overline{C}+\overline{D}$ is equal to 0 when $A = 1$, $B = 1$, $C = 1$ and $D = 1$, *i.e.*,

$$\overline{A}+\overline{B}+\overline{C}+\overline{D} = \overline{1}+\overline{1}+\overline{1}+\overline{1} = 0+0+0+0 = 0 \quad \textbf{Ans.}$$

Example 4-56. *Convert into standard SOP and standard POS form*

$$f = A\overline{B} + B\overline{C}$$ [*Gauhati University, 2003*]

Solution. Given: The expression

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

SOP:

We know that the domain of the given expression is *A*, *B* and *C*. From the expression we find that all the two product terms an in the non-standard form. Let us convert then product terms to a standard form taking one term at a time.

The first term $A\overline{B}$ has a missing variable *C* or $\overline{C}$. So multiply the first term by $C + \overline{C}$ a follows

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

The second term $B\overline{C}$ has a missing variable *A* or $\overline{A}$. So multiply the second term by $A + \overline{A}$ as follows,

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

Thus the standard SOP form of the given expression,

$$A\overline{B} + B\overline{C} = A\overline{B}C + A\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}B\overline{C} \quad \textbf{Ans.}$$

POS:

We can convert the SOP expression into POS expression, the SOP expression,

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

The evaluation of each product term in the given expression is,

101 + 100 + 110 + 010.

Since there are three variables in the domain of this expression (*i.e. A*, *B* and *C*), there are total of 2^J (= 8) possible combinations. The given SOP expression contains four of these combinations, so the

POS must contain the other four which are 000, 001, 011 and 111. Since these are binary values that make the sum term 0, therefore the equivalent POS expression is,

$$(\overline{A}+\overline{B}+\overline{C})(\overline{A}+\overline{B}+C)(\overline{A}+B+C)(A+B+C) \quad \textbf{Ans.}$$

Example 4-57. *Convert* $(A + B)$ $(A + C)$ $(B+\overline{C})$ *into standard POS form.*

[*Anna University, April/May 2000*]

Solution. Given, the Boolean expression.

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

We know that the domain of the given expression is A, B and C. From the expression. We find that the sum terms $(A + B)$, $(A + C)$ and $(B+\overline{C})$ are in the non-standard form. Let us convert these sum terms to a standard form taking one term at a time.

The first term $(A + B)$ has a missing variable C or $\overline{C}$. So we add $C\overline{C}$ and apply rule 12 a follows.

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

The second term $(A + C)$ has missing variable B or $\overline{B}$. So we add $B\overline{B}$ and apply rule 12 as follows.

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

The third term $(B+\overline{C})$ has missing variable A or $\overline{A}$.. So we add $A\overline{A}$ and apply rule 12 as follows.

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

Thus the standard product of sum of the given Boolean expression,

$$\begin{aligned}(A+B)(A+C)(B+\overline{C}) &= (A+B+C)(A+B+\overline{C})(A+B+C)(A+C+\overline{B})\\ &\qquad (A+B+\overline{C})(\overline{A}+B+\overline{C})\\ &= (A+B+C)(A+B+\overline{C})(A+C+\overline{B})(\overline{A}+B+\overline{C}) \quad \textbf{Ans.}\end{aligned}$$

Example 4-58. *Express the following function in product of sum form and find out the numerical value of maxterms:*

$$f(A, B, C, D) = A\overline{D}+BD+\overline{B}C \qquad (\textit{Nagpur University., 2004})$$

Solution. Given: The given expression.

$$f(A, B, C, D) = A\overline{D}+BD+\overline{B}C$$

We know that the domain of the given expression is A, B, C and D, from the expression we find that all the three product terms are in the non-standard form. Let convert these product terms to C standard form taking one term at a time.

The first term $A\overline{D}$ has a missing variable B or $\overline{B}$ and C or $\overline{C}$. So multiply the first term by s $(B+\overline{B})$ and $(C+\overline{C})$ as follows.

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

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

$$= \overline{A}DBC + \overline{A}DB\overline{C} + \overline{A}D\overline{B}C + \overline{A}D\overline{B}\,\overline{C}$$

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

The second term BD has a missing variable A or $\overline{A}$ and C or $\overline{C}$. So multiply the second term by $(A + \overline{A})$ and $(C + \overline{C})$ as follows,

$$BD = BD(A + \overline{A})(C + \overline{C})$$

$$= (ABD + \overline{A}BD)\,(C + \overline{C})$$

$$= ABCD + AB\overline{C}D + \overline{A}BCD + \overline{A}B\overline{C}D$$

The third term $\overline{B}C$ has a missing variable A or $\overline{A}$ and D or $\overline{D}$ as follows,

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

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

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

Thus the SOP form of the given expression,

$$\overline{A}D + BD + \overline{B}C = \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}\,\overline{C}D + ABCD$$

$$+ AB\overline{C}D + \overline{A}BCD + \overline{A}B\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D}$$

$$= \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}\,\overline{C}D + ABCD$$

$$+ AB\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}C\overline{D}$$

Since there are four variables in the domain of this expression (*i.e.* A, B, C and D) there are total of 2^4 (= 16) possible combinations. The SOP contains nine of these combinations, so the POS must contain the other seven terms, the equivalent POS expression is,

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

$$(A + \overline{B} + \overline{C} + D)(A + B + \overline{C} + \overline{D})(A + B + C + \overline{D}) \quad \textbf{Ans.}$$

Example 4-59. *Find the maxterms for the expression*

$$F = A\overline{C} + AB\overline{C} + \overline{A}BC\,. \qquad \textit{(PTU., Dec. 2009)}$$

Solution. Given: The Boolean expression,

$$F = A\overline{C} + AB\overline{C} + \overline{A}BC$$

We know that the given expression has 3 variables in its domain. A quick look at the expression indicates that, it not a standard SOP form. Let us convert it into a standard SOP form.

The first term on the right side has a missing variable B or $\overline{B}$. So we multiply this term by $B + \overline{B}$. Thus

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

$$= \underline{AB}\overline{C} + A\overline{B}\,\overline{C} + \underline{AB\overline{C}} + \overline{A}BC$$

combining the similar terms, we get,

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

The equation on the right above is a standard SOP expression. Since there are 3 variables in the domain, therefore there are eight possible combinations of binary values as shown in the left three columns of the Table 4-5.

Table 4-5.

Inputs			Output	
A	B	C	F	
0	0	0	0	→ M_0
0	0	1	0	→ M_1
0	1	0	0	→ M_2
0	1	1	1	→ m_3
1	0	0	1	→ m_4
1	0	1	0	→ M_5
1	1	0	1	→ m_6
1	1	1	0	→ M_7

The SOP contains three of these combinations, so the POS must contain the other five which are 000,001, 010, 101 and 111. Since these are binary values that make the sum term O, therefore the equivalent POS expression is,

$(\overline{A}+\overline{B}+\overline{C})(\overline{A}+\overline{B}+C)(\overline{A}+B+\overline{C})(A+\overline{B}+C)(A+B+C)$ **Ans.**

4-24. Converting Standard Sum-of-Products (SOP) to Standard Product-of-Sums (POS)

It will be interesting to know that the binary values of the product terms in a given standard sum-of-products (SOP) expression are not present in the equivalent standard prduct-of-sums (POS) expression. Also, the binary values that are not represented in the SOP expression are present in the equivalent POS expression. Therefore, to convert from standard SOP to standard POS, we take the following stepts :

1. Evaluate each product term in the SOP expression, *i.e.* determine the binary numbers which represent the product terms.
2. Determine all the binary numbers not included in the evaluation in step 1.
3. Write the equivalent sum term for each binary number from step 2 and express in POS form.

A similar procedure can be used to convert a standard POS expression to a standard SOP expression.

Example 4-60. *Convert the following sum-of-products (SOP) expression to an equivalent product-of-sums (POS) expression* :

$$\overline{A}\overline{B}\overline{C}+\overline{A}\overline{B}C+\overline{A}BC+A\overline{B}C+ABC$$

Solution. Given, the SOP expression,

$$\overline{A}\overline{B}\overline{C}+\overline{A}\overline{B}C+\overline{A}BC+A\overline{B}C+ABC$$

The evaluation of each product term in the given expression is,

$$000+001+011+101+111$$

Since there are three variables in the domain of this expression (*i.e. A*, *B* and *C*), there are total of 2^3 (= 8) possible combinations. The given SOP expression contains five of these combinations, so the POS must contain the other three which are 010, 100 and 110. Since these are binary values that make the sum term 0, therefore the equivalent POS expression is,

$$(\overline{A}+B+\overline{C})(A+\overline{B}+\overline{C})(A+B+\overline{C}) \textbf{ Ans.}$$

4-25. Boolean Expressions and Truth Tables

As a matter a fact, all standard Boolean expressions (SOP or POS) can be easily converted into truth table format. This is done by using binary values for each term in the expression. The conversion into a truth table format is required because of the fact that truth table is a common way of presenting the logical operation of a circuit. The turth table can also be used to determine a standard SOP or POS expression.

4-26. Converting Sum-of-Products (SOP) Expressions to Truth Table Format

We have already discussed in Article 4-17 that on SOP (sum-of-products) expression is equal to 1 only if at least one of the product terms is equal to 1. We know that a truth table is simply a list of the possible combinations of input variable values and the corresponding output values (*i.e.* 1 or 0). For a Boolean expression with a domain of 2 variables, there are four different combinations of those variables (because $2^2 = 4$). Similarly, for a Boolean expression with a domain of 3 variables, there are eight different combinations of those variables (because $2^3 = 8$). For a Boolean expression, with a domain of 4 variables, there are sixteen different combinations of those variables (because $2^4 = 16$) and so on.

Following is the procedure to construct a truth table for the given SOP expression.

1. List all the possible combinations of binary values of the variables in the given expression.
2. Convert the SOP expression to standard form if it is not already so.
3. Place a 1 in the output column (X) for each binary value that makes the standard SOP expression a 1 and place a 0 for all the remaining values.

In order to illustrate the procedure, consider the construction of a truth table for the standard SOP expression,

$$\overline{A}\overline{B}C + A\overline{B}\overline{C} + ABC$$

We know that there are 3 variables in the domain, so there are eight possible combinations of binary values as shown in the left three columns of the Table 4-5. The binary values that make the product terms in the expression equal to 1 are 001 for $\overline{A}\overline{B}C$, 100 for $A\overline{B}\overline{C}$, and 111 for ABC. For each of these binary values, a 1 is placed in the output column as shown in the table above. For each of the remaining binary combinations, a 0 is placed in the output column.

Table. 4-5.

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

Example 4-61. *Develop a truth table for the following Boolean expression* :

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

Solution . Given, the Boolean expression,

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

We know that the given expression has 3 variables in its domain. A quick look at the expression indicates that it is not a standard SOP (sum-of-products) form. Let us simplify the Boolean expression first and then convert it into a standard SOP form.

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

The first term on the right side has a missing variable C or $\bar{C}$. So we multiply this term by $C+\bar{C}$. The third term has two missing variables B or $\bar{B}$ and C or $\bar{C}$. So we multiply this term by $B+\bar{B}$ and $C+\bar{C}$. The fourth term again has two missing variables, A or $\bar{A}$ and B or $\bar{B}$. So we multiply this term by $A+\bar{A}$ and $C+\bar{C}$. Thus,

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

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

$$= \underline{\bar{A}BC} + \bar{A}\underset{\times}{B}\bar{C} + A\underset{?}{B}\bar{C} + \underline{\bar{A}BC} + \bar{A}\underset{\times}{B}\bar{C} + \bar{A}\underset{\times\times}{\bar{B}}C + \bar{A}\underset{o}{\bar{B}}\bar{C} + A\underset{?}{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}\underset{\times}{B}\bar{C} + \bar{A}\underset{o}{\bar{B}}\bar{C} + A\bar{B}C$$

Combining the similar terms, we get,

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

The equation on the right above is a standard SOP expression. Since there are 3 variables in the domain, therefore there are eight possible combinations of binary values as shown in the left three columns of the table 4-6. The binary values that make the product terms in the expression equal to 1 are 011 ($for\ \bar{A}BC$), 010($for\ \bar{A}BC$), 110($for\ AB\bar{C}$), 001($for\ \bar{A}\bar{B}C$), 000($for\ \bar{A}\bar{B}\bar{C}$), 100($forA\bar{B}\bar{C}$), and101(for $A\bar{B}C$). For each of these seven binary values, we place a 1 in the output column as shown in the table above. The only remaining binary combination is 111 and we place a 0 for this combination in the output column.

Table. 4-6.

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

4-27. Converting Product-of-Sums (POS) Expressions to a Truth Table Format

We have already discussed in Article 4-21 that a POS (product-of-sums) expression is equal to 0 only if at least one of the sum terms is equal to 0. Following is the procedure to construct a truth table for the given POS expression,

1. List all the possible combinations of binary values of the variables.
2. Convert the POS expression to standard form if it is not already so.
3. Place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values.

In order to illustrate the procedure, consider the construction of a truth table for a standard POS expression,

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

We know that there are 3 variables in the domain, so there are eight possible combinations of binary values as shown in the left three columns of the Table 4-7. The binary values that make the sum terms in the expression equal to 0 are $110(\text{for } \overline{A}+\overline{B}+C), 011(\text{for } A+\overline{B}+\overline{C})$ and 000 (for $A + B + C$). For each of these binary values, a 0 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 1 is placed in the output column.

Table 4-7.

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

Example 4-62. *Develop a truth table for the following standard POS (product-of-sums) expression*:

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

Solution. Given, the standard POS expression,

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

We know that there are 3 variables in the domain of the given POS expression. So there are eight possible combinations of binary values as shown in the left three columns of the Table 4-8.

Table. 4-8.

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

The binary values that make the sum terms in the expression equal to 0 are 111(for $\overline{A}+\overline{B}+\overline{C}$), 101(for $\overline{A}+B+\overline{C}$), 011(for $A+\overline{B}+\overline{C}$). For each of these binary values, a 0 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 1 is placed in the output column. **Ans.**

Example 4-63. *Transform each of the following canonical expression into its other canonical form in decimal notation.*

(i) $f(x, y, z) = \Sigma m\ (1, 3, 5)$

(ii) $f(w, x, y, z) = \lambda M\ (0, 2, 5, 6, 7, 8, 9, 11, 12)$ *(VTU., July/Aug. 2004)*

Solution. (*i*) Given : The expression

$$f(x, y, z) = \Sigma m\ (1, 3, 5)$$

There are three variables x, y and z, then the total number of term 2^3 (= 8). In the given SOP there are three terms, 1, 3 and 5. The POS form contain the remaining five term, 0, 2, 4, 6 and 7. The POS form of expression is given below :

$$f(x, y, z) = \lambda M\ (0, 2, 4, 6, 7) \quad \textbf{Ans.}$$

(*ii*) Given : The expression,

$$f(w, x, y, z) = \lambda M\ (0, 2, 5, 6, 7, 8, 9, 11, 12)$$

In the above expression there are four variables, *w*, *x*, *y* and *z*, the total number of term 2^4 (= 16). The given function is form of POS and it contain nine terms. 0, 2, 5, 6, 7, 8, 9, 11 and 12. The SOP form contain the remaining seven term, 1, 3, 4, 10, 13, 14 and 15. The SOP form of expression is given below:

$$f(w, x, y, z) = \Sigma m\ (1, 3, 4, 10, 13, 14, 15) \quad \textbf{Ans.}$$

Example 4-64. *Determine, the POS and SOP form of*

$Z = \Sigma m\ (0, 1, 3, 6, 7, 8, 9, 13, 15)$

Solution. Given : The given expression,

$$Z = \Sigma m\ (0, 1, 3, 6, 7, 8, 9, 13, 15)$$

The given function is form of min terms, so we directly find out the SOP from the expression. There are four variable *A*, *B*, *C* and *D*, thus

$$= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + AB\overline{C}\,\overline{D} + ABCD$$

Ans.

There are four variables in the domain of this expression (*i.e.* *A*, *B*, *C* and *D*), there are total of 2^4 (= 16) possible combinations. The SOP expression contains nine of these combinations, so the POS must contain the other seven term.

POS terms.

0010 + 0100 + 0101 + 1010 + 1011 + 1100 + 1110

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

$$(A + B + \overline{C} + \overline{D})(A + B + C + \overline{D}) \quad \textbf{Ans.}$$

Example 4-65. *Rewrite the following Boolean expression in the M-notation and simplify*

(*i*) $(a + b)\ (a + c)$

(*ii*) $(a + b)\ (b + c)\ (\overline{c} + a)$ *(VTU., Jan./Feb. 2005)*

Solution. (*i*) Given : The boolean expression,

$(a + b)\ (a + c)$

We know that the domain of the given expression is *a*, *b* and *c*. From the expression, we find that both the sum terms *i.e.* $(a + b)$ and $(a + c)$ are in the non-standard form. Let us convert these sum terms to a standard form taking one term at a time.

The first term $(a + b)$ has a missing variable c or $\overline{c}$. So we add $c\overline{c}$ and apply rule 12 as follows

$$\begin{aligned}(a+b) &= (a+b+c\overline{c})\\ &= (a+b+c)(a+b+\overline{c})\end{aligned}$$

The second term $(a + c)$ has a missing variable b or $\overline{b}$. So we add $b\overline{b}$ and apply rule 12 as follows.

$$\begin{aligned}(a+c) &= (a+c+b\overline{b})\\ &= (a+c+b)(a+c+\overline{b})\end{aligned}$$

Thus the standard POS (M-notation)

$$\begin{aligned}(a+b)(a+c) &= (a+b+c)(a+b+\overline{c})(a+b+c)(a+\overline{b}+c)\\ &= (a+b+c)(a+b+\overline{c})(a+\overline{b}+c)\\ &= M_0\cdot M_1\cdot M_2\\ &= \pi M(0, 1, 2) \quad \textbf{Ans.}\end{aligned}$$

(*ii*) Given : The boolean expression,

$$(a+b)(b+c)(\overline{c}+a)$$

We know that the domain of the given expression is a, b and c. From the expression we find that the sum terms *i.e.* $(a + b)$, $(b + c)$ and $(\overline{c} + a)$ are in the non-standard form. Let us convert these sum terms to a standard form taking one term at a time.

The first term $(a + b)$ has a missing variable c or $\overline{c}$. So we add $c\overline{c}$ and apply rule 12 as follows.

$$\begin{aligned}a+b &= a+b+c\overline{c}\\ &= (a+b+c)(a+b+\overline{c})\end{aligned}$$

The second term $(b + c)$ has a missing variable a or $\overline{a}$. We add $a\overline{a}$ and apply rule 12 as follows,

$$\begin{aligned}b+c &= b+c+a\overline{a}\\ &= (a+b+c)(\overline{a}+b+c)\end{aligned}$$

The third term $(\overline{c} + a)$ has a missing variable b or $\overline{b}$. So we add $b\overline{b}$ and apply rule 12 as follows

$$\begin{aligned}\overline{c}+a &= \overline{c}+a+b\overline{b}\\ &= (a+b+\overline{c})(a+\overline{b}+\overline{c})\end{aligned}$$

Thus the standard POS,

$$\begin{aligned}(a+b)(b+c)(\overline{c}+a) &= (a+b+c)(a+b+\overline{c})(a+b+c)(\overline{a}+b+c)\\ &\qquad (a+b+\overline{c})(a+\overline{b}+\overline{c})\\ &= (a+b+c)(a+b+\overline{c})(\overline{a}+b+c)(a+\overline{b}+\overline{c})\\ &= M_0\cdot M_1\cdot M_4\cdot M_3\\ &= \pi M(0, 1, 3, 4) \quad \textbf{Ans.}\end{aligned}$$

Example 4-66. *Transform each of the following canonical expressions into other canonical form in decimal notation and express is simplified form in decimal notation.*

(*i*) $f(x, y, z) = \Sigma\, m\,(0, 1, 3, 4, 6, 7)$

(*ii*) $f(w, x, y, z) = \pi\, m\,(0, 1, 2, 3, 4, 6, 12)$

(VTU. July 2006)

Solution. (*i*) Given : The expression

$$f(x, y, z) = \Sigma m\,(0, 1, 3, 4, 6, 7)$$

There are three variable x, y, z, Then the total number of term 2^3 (= 8). The given expression is SOP, it contain six term from the total number of term. So the POS form contain remaining two terms, 2 and 5. The POS form of expression is given below

$$f(x, y, z) = \pi M(2, 5) \quad \textbf{Ans.}$$

(*ii*) Given : The expression

$$f(w, x, y, z) = \pi M(0, 1, 2, 3, 4, 6, 12)$$

There are four variables w, x, y, z, then the total number of term 2^4 (= 16). The given expression is POS, it contain seven term from the total number of term. So the SOP form contain remaining nine terms, 5, 7, 8, 9, 10, 11, 13, 14 and 15. The SOP form of expression is given below

$$f(w, x, y, z) = \Sigma m(5, 7, 8, 9, 10, 11, 13, 14, 15) \quad \textbf{Ans.}$$

Example 4-67. *Expand the following into canonical form by using appropriate Boolean laws:*

(*a*) $f = A\overline{B}C + \overline{A}BC + AB + BC + \overline{A}\overline{B}\overline{C}$

(*b*) $g = (X + Y)(\overline{X} + \overline{Y} + Z)(Y + \overline{Z})(\overline{X} + Y + \overline{Z})$

(Mahatma Gandhi University, Nov. 2005)

Solution. (*a*) Given : The Boolean expression

$$f = A\overline{B}C + \overline{A}BC + AB + BC + \overline{A}\overline{B}\overline{C}$$

We know that the domain of the given expression is A, B, and C, from the expression we find that all the third and fourth product term is in the non-standard form. Let us convert these product terms to a standard form taking one term at a time.

The third term AB has a missing variable C or $\overline{C}$. So multiply the first term by $(C + \overline{C})$ as follows,

$$AB = AB(C + \overline{C}) = ABC + AB\overline{C}$$

The fourth term BC has a missing variable A or $\overline{A}$. So multiply the first term by $(A + \overline{A})$ as follows,

$$BC = BC(A + \overline{A}) + ABC + \overline{A}BC$$

Thus the standard SOP (*i.e.* sum-of-products) form of the given expression,

$$f = A\overline{B}C + \overline{A}BC + ABC + AB\overline{C} + ABC + \overline{A}BC + \overline{A}\overline{B}\overline{C}$$

Solution. $= A\overline{B}C + \overline{A}BC + ABC + AB\overline{C} + \overline{A}\overline{B}\overline{C}$ **Ans.**

(*b*) Given : The Boolean expression

$$g = (X + Y)(\overline{X} + \overline{Y} + Z)(Y + \overline{Z})(\overline{X} + Y + \overline{Z})$$

We know that the domain of the given expression is X, Y, and Z, from the expression we find that all the first and third product terms are in the non-standard form. Let us convert these terms to a standard form taking one term at a time.

The first term (X + Y) has a missing variable Z or $\overline{Z}$. So we add $Z\overline{Z}$ and apply rule 12 as follows,

$$X + Y = X + Y + Z\overline{Z} = (X + Y + Z)(X + Y + \overline{Z})$$

The third term $(Y + \overline{Z})$ has a missing vriable X or $\overline{X}$. So we add $X\overline{X}$ and apply rule 12 as follows,

$$Y + \overline{Z} = (Y + \overline{Z} + X\overline{X}) = (Y + \overline{Z} + X)(Y + \overline{Z} + \overline{X})$$

Thus the standard product-of sum form of the given Boolean expression,

$$g = (X + Y + Z)(X + Y + \overline{Z})(\overline{X} + \overline{Y} + Z)(Y + \overline{Z} + X)(Y + \overline{Z} + \overline{X})(\overline{X} + Y + \overline{Z}) \quad \textbf{Ans.}$$

Example 4-68. *Convert the following function into canonical form.*

(GBTU/MTU, 2006)

(*a*) $F = AB + B\overline{C}D + \overline{A}D$

(*b*) $F = (A + \overline{B})(\overline{C} + \overline{D})(\overline{B} + \overline{C})$

Solution. (*a*) Given : The Boolean expression

$$F = AB + B\overline{C}D + \overline{A}D$$

We know that the domain of the given expression is A, B, C and D. From the expression we find that all the three product terms are in the non-standard form. Let us convert these product terms to a standard form taking one term at a time.

The first term AB has a missing variable C or $\overline{C}$ and D or $\overline{D}$. So multiply the first term by $(C + \overline{C})$ and $(D + \overline{D})$ as follows,

$$AB = AB(C + \overline{C})(D + \overline{D}) = (ABC + AB\overline{C})(D + \overline{D})$$
$$= ABCD + AB\overline{C}D + ABC\overline{D} + AB\overline{C}\,\overline{D}$$

The second term $B\overline{C}D$ has a missing variable A or $\overline{A}$. So multiply the second term by $(A + \overline{A})$ as follows,

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

The third term $A\overline{D}$ has a missing variable B or $\overline{B}$ and C or $\overline{C}$. So multiply the third term by $(B + \overline{B})$ and $(C + \overline{C})$ as follows,

$$\overline{A}D = \overline{A}D(B + \overline{B})(C + \overline{C}) = (\overline{A}BD + \overline{A}\,\overline{B}D)(C + \overline{C})$$
$$= \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}\,\overline{C}D$$

Thus the standard SOP (i.e. sum-of-products) form of the given expression,

$$AB + B\overline{C}D + \overline{A}D = ABCD + AB\overline{C}\,\overline{D} + AB\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD$$
$$+ \overline{A}B\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}\,\overline{C}D$$
$$= ABCD + AB\overline{C}D + ABC\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}\,\overline{C}D$$

(*b*) Given : The Boolean expression

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

We know that the domain of the given expression is A, B, C and D, From the expression we find that all the terms are in the non-standard form. Let us convert these terms to a standard form taking one term at a time.

The first term $(A + \overline{B})$ has a missing variable C or $\overline{C}$ and D or $\overline{D}$. So we add $C\overline{C}$ and $D\overline{D}$ and apply rule 12 as follows,

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

The second term $(\overline{C} + \overline{D})$ has a missing variable A or $\overline{A}$ and B or $\overline{B}$. So we add $A\overline{A}$ and $B\overline{B}$ and apply rule 12 as follows,

$$\begin{aligned}\overline{C}+\overline{D} &= \overline{C}+\overline{D}+A\overline{A}+B\overline{B} = (\overline{C}+\overline{D}+A\overline{A}+B)\,(\overline{C}+\overline{D}+A\overline{A}+\overline{B})\\ &= (A+B+\overline{C}+\overline{D})(\overline{A}+B+\overline{C}+\overline{D})(A+\overline{B}+\overline{C}+\overline{D})(\overline{A}+\overline{B}+\overline{C}+\overline{D})\end{aligned}$$

The third term $(\overline{B}+\overline{C})$ has a missing variable $\overline{A}$ or $\overline{A}$ and D or $\overline{D}$. So we add $\overline{A}$ and $D\overline{D}$ and apply rule 12 as follows,

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

Thus the standard product-of sum form of the given Boolean expression,

$$\begin{aligned}(A+\overline{B})(\overline{C}+\overline{D})(\overline{B}+\overline{C}) = {}& (A+\overline{B}+C+D)(A+\overline{B}+\overline{C}+D)(A+\overline{B}+C+\overline{D})\\ & (A+\overline{B}+\overline{C}+\overline{D})(A+B+\overline{C}+\overline{D})(\overline{A}+B+\overline{C}+\overline{D})\\ & (A+\overline{B}+\overline{C}+\overline{D})(\overline{A}+\overline{B}+\overline{C}+\overline{D})(A+\overline{B}+\overline{C}+D)\\ & (\overline{A}+\overline{B}+\overline{C}+D)(A+\overline{B}+\overline{C}+\overline{D})(\overline{A}+\overline{B}+\overline{C}+\overline{D})\\ (A+\overline{B})(\overline{C}+\overline{D})(\overline{B}+\overline{C}) = {}& (A+\overline{B}+C+D)(A+\overline{B}+\overline{C}+D)(A+\overline{B}+C+\overline{D})\\ & (A+\overline{B}+\overline{C}+\overline{D})(A+B+\overline{C}+\overline{D})(\overline{A}+B+\overline{C}+\overline{D})\\ & (\overline{A}+\overline{B}+\overline{C}+\overline{D})(\overline{A}+\overline{B}+\overline{C}+D) \quad \textbf{Ans.}\end{aligned}$$

4-28. Determining Standard Expressions from a Truth Table

We have already discussed in the previous articles the procedure for converting a standard SOP or POS expression. Now we will study the method of determining standard SOP and POS expressions from a truth table.

To determine the standard SOP expression represented by a truth table, list the binary values of the input variables for which the output is 1. Convert each binary value to the corresponding product term by replacing each 1 with the corresponding variable and each 0 with the corresponding variable complement. For example, 1001 is converted to a product term as follow :

$$1001 \rightarrow A\overline{B}\overline{C}D$$

To determine a standard POS expression represented by a truth table, list the binary values for which the output is 0. Convert each binary value to the corresponding sum term by replacing each 1 with the corresponding variable complement and each 0 with the corresponding variable. For example, the binary value 1010 is converted to a sum term as follows :

$$1010 \rightarrow \overline{A}+B+\overline{C}+D$$

Example 4-69. *Fig.* 4-46 *shows a truth table for a typical digital circuit.*

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

Fig. 4-46.

Determine the standard SOP (*sums-of-products*) *expression and the equivalent standard POS* (*product-of-sums*) *expression for the given truth table.*

Solution. Given, the truth table of Fig. 4-46.

Standard SOP expression

From the given truth table we find, that there are four 1s in the output column and the corresponding binary values are, 010, 011, 110 and 111. These binary values are converted to product terms as follows :

$$010 \rightarrow \overline{A}B\overline{C}$$

$$011 \rightarrow \overline{A}BC$$

$$110 \rightarrow AB\overline{C}$$

$$111 \rightarrow ABC$$

Thus the resulting standard SOP expression for the output X is,

$$X = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + ABC \quad \textbf{Ans.}$$

Standard POS expression

From the given truth table, we find that the output is 0 for binary values 000, 001, 100 and 101. These binary values are converted to sum terms as follows :

$$000 \rightarrow A + B + C$$

$$001 \rightarrow A + B + \overline{C}$$

$$100 \rightarrow \overline{A} + B + C$$

$$101 \rightarrow \overline{A} + B + \overline{C}$$

The resulting standard POS expression for the output,

$$X = (A + B + C)(A + B + \overline{C})(\overline{A} + B + C)(\overline{A} + B + \overline{C}) \quad \textbf{Ans.}$$

4-29. The Karnaugh Map

The Karnaugh map (or simply a *K*-map) is similar to a trugh table because it presents all the possible values of input variables and the resulting output for each value. However, instead of being organised into columns and rows like a truth table, the Karnaugh map is an array of squares (or cells) in which each square represents a binary value of the input variables. The squares are arranged in a way so that simplification of given expression with two-three, four, and five variables Karnaugh maps to illustrate the principles. Karnaugh map-with five-variables is beyond the scope of this book. For higher number of variables, a Quine-McClusky method can be used. This method is also beyond the scope of this book.

The number of squares in a Karnaugh map is equal to the total number of possible input variable combination (as is the number of rows in a trugh table). For two variables, the number of square is 2^2 =4, for three variables, the number of squares is 2^3 = 8 and for four variables, the number of squares is 2^4 = 16.

4-30. The Two-variable Karnaugh Map

Fig. 4-47 (*a*) shows a two-variable Karnaugh map. As seen, it is an array of four squares. In this cass, *A* and *B* are used for two variables although any other two letters could be used. The binary values of *A* (*i.e.* 0 and 1) are indicated along the left side as $\overline{A}$ and *A* (notice the sequence) and the binary values of B are indicated across the top as $\overline{B}$ and B. The value of a given square is the value of A at the left in the same row combined with the value of B at the top in the same column. For example, a square in the upper left corner has a value of $\overline{A}\,\overline{B}$ and a square in the lower right corner has a value of AB. Fig. 4-47 (*b*) shows the standard product terms represented by each square in the karnaugh map.

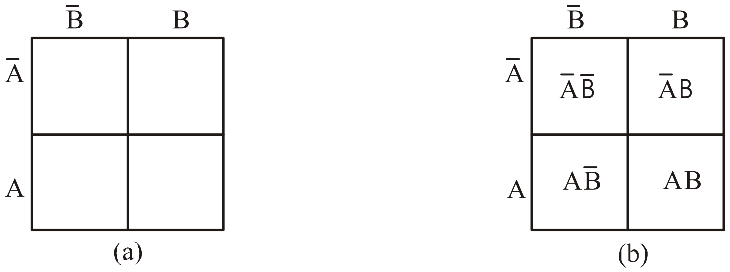


Fig. 4-47.

4-31. The Three-variable Karnaugh Map

Fig. 4-48 (*a*) shows a three-variable Karnaugh map. As seen it is an array of eight squares. In this case, A, B and C are used for the variables although any other three letters could be used. The value of A and B are along the left side (notice the sequence carefully) and the values of C are across the top.

The value of a given square is the value of A and B at the left in the same row combined with the value of C at the top in the same column. For example, a square in the upper left corner has a value of $\overline{A}\,\overline{B}\,\overline{C}$ and a square in the bottom right corner has a value of $A\,\overline{B}\,C$. Fig. 4-48 (*b*) shows the product terms that are represented by each square in the Karnaugh map.

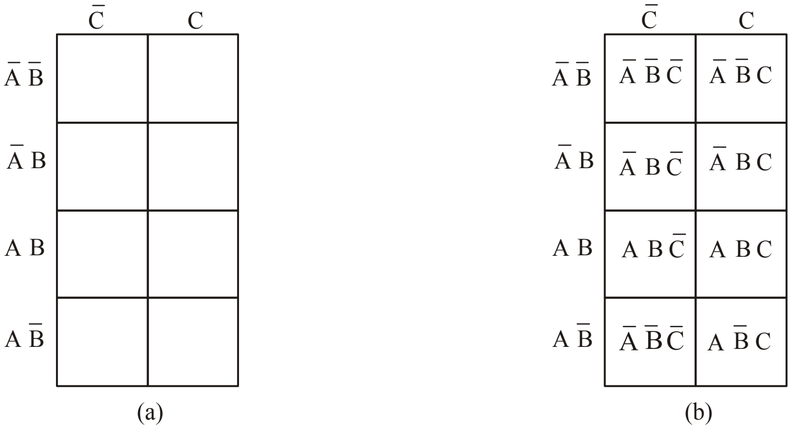

$\overline{C}$
C
$\overline{A}\,\overline{B}$
$\overline{A}\,B$
$A\,B$
$A\,\overline{B}$
(a)
$\overline{C}$
C
$\overline{A}\,\overline{B}$
$\overline{A}\,\overline{B}\,\overline{C}$
$\overline{A}\,\overline{B}\,C$
$\overline{A}\,B$
$\overline{A}\,B\,\overline{C}$
$\overline{A}\,B\,C$
$A\,B$
$A\,B\,\overline{C}$
$A\,B\,C$
$A\,\overline{B}$
$\overline{A}\,\overline{B}\,\overline{C}$
$A\,\overline{B}\,C$
(b)

Fig. 4-48.

4-32. The Four -variable Karnaugh Map

Fig. 4-49 (*a*) shows a four-variable Karnaugh map. As seen, it is an array of sixteen squares. In this case A, B, C and D are used for the variables. The values of A and B are along the left side, and the values of C and C are across the top. The sequence of the variable values may be noted carefully.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$				
$\bar{A}B$				
AB				
$A\bar{B}$				

(a)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}\bar{C}D$
$\bar{A}B$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
AB	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
$A\bar{B}$	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

(b)

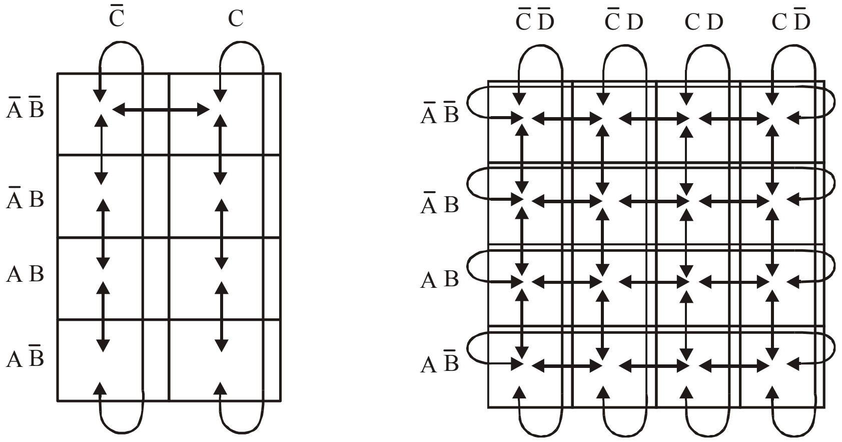
Fig. 4-49.

The value of a given square is the value of *A* and *B* at the left in the same row combined with the values of *C* and *D* at the top in the same column. For example, a square in the upper right corner has a value $\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$ and a square in the lower left corner has a value $\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$. Fig 4-49 (*b*) shows the standard product terms that are represented by each square in the four-variable Karnaugh map.

4-33. Square Adjacency in Karnaugh Map

We have already discussed a two-variable Karnaugh map, a three-variable Karnaugh map and a four-variable Karnaugh map. Now we shall discuss the concept of square adjacency in a Karnaugh map.

It will be interesting to know that the squares in a Karnaugh map are arranged in such a way that there is only a single-variable change between adjacent squares. Adjacency is defined as a single-variable change. It means the squares that differ by only one variable are adjacent in a a Karnaugh map shown in Fig. 4-50 (*b*), the $\bar{A}\,\bar{B}\,\bar{C}$ square is adjacent to $\bar{A}\,B\,\bar{C}$ square, the $\bar{A}BC$ square and the $AB\bar{C}$ square. It may be carefully noted that square with values that differ by more than one variable are not adjacent. For example, the $\bar{A}B\bar{C}$ square is not adjacent to the $\bar{A}\bar{B}C$ square, the $A\bar{B}\bar{C}$ square or the $A\bar{B}C$ square. In other words, each square is adjacent to the squares that are immediately next to it on any of its four sides. However, a square is not adjacent to the squares that diagonally touch any of its corners.



(a) (b)

Fig. 4-50.

It may also be noted that squares in the top row are adjacent to the corresponding squares in the bottom row and squares in the outerleft column are adjacent to the corresponding squares in the outer right column. This is called **"Wrap around"** adjacency because we can think of the map as wrapping around from top to bottom to form a cylinder or from left to right to form a cylinder. Fig. 4-50 (*a*) and (*b*) shows the square adjacencies with a three-variable and a four-variable Karnaugh maps respectively.

Notice the square adjacencies in a four variable Karnaugh map : Here for example, the square $\overline{A}\overline{B}\overline{C}\overline{D}$ is adjacent to $\overline{A}\overline{B}\overline{C}D$ square, $\overline{A}B\overline{C}\overline{D}$ square, $A\overline{B}\overline{C}\overline{D}$ square and $A\overline{B}C\overline{D}$ square. Similarly $\overline{A}B\overline{C}D$ square is adjacent to $\overline{A}\overline{B}\overline{C}D$ square, $\overline{A}B\overline{C}\overline{D}$ square, $\overline{A}BCD$ square and $AB\overline{C}D$ square.

4-34. Mapping a Standard SOP Expression on the Karnaugh Map

Consider a *SOP* (sum-of-products) expression. $\overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}C + A\overline{B}\overline{C}$. In order to map this expression on the Karnaugh map, we need a three variable Karnaugh-map because the given expression has three variables *A*, *B*, and *C*. Then select the first product term $\overline{A}\overline{B}\overline{C}$ and enter 1 in the corresponding square (*i.e.* the first row and the first column) as shown in Fig. 4-51.

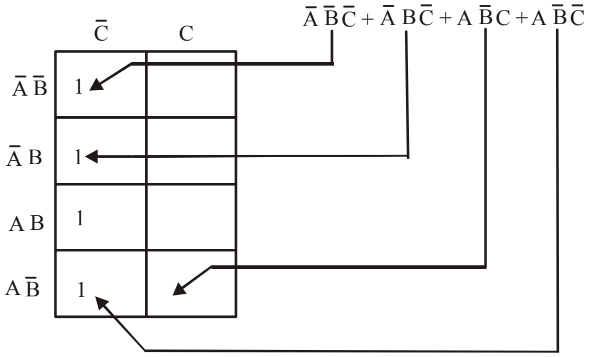

$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,C + A\,\overline{B}\,\overline{C}$
$\overline{C}$
C
$\overline{A}\,\overline{B}$
$\overline{A}\,B$
A B
$A\,\overline{B}$
1
1
1
1

Fig. 4-51.

Similarly, for the second product term, $\overline{A}B\overline{C}$ place a 1 in the second row and first column. Repeat this process for the other two product terms, *i.e.* $A\overline{B}C$ and $A\overline{B}\overline{C}$. Thw squares that do not have 1 are the squares for which the expression is 0. Usually when working with sum-of-product expressions, the 0*s* are left off the map.

Example 4-70. *Map the following SOP expression on the Karnaugh map* : $\overline{A}\overline{B}C + \overline{A}BC + AB\overline{C} + ABC.$

Solution. Sketch a three variable Karnaugh map as shown in Fig. 4-52. Select the first product term $\overline{A}\overline{B}C$ and enter 1 in the corresponding square. Similarly enter 1 for the other product terms in the given *SOP* expression. Check that number of 1*s* in the Karnaugh map is equal to the number of product terms in the given *SOP* expression.

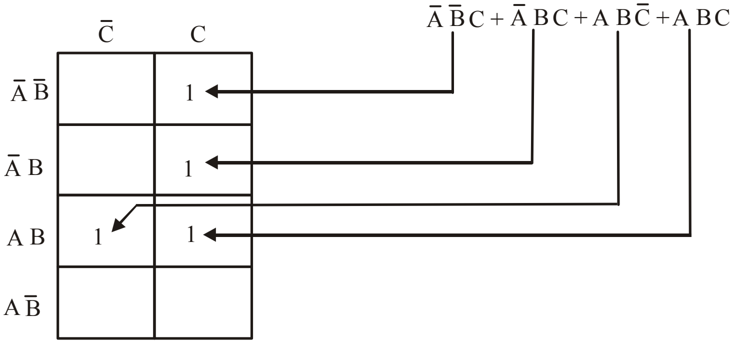
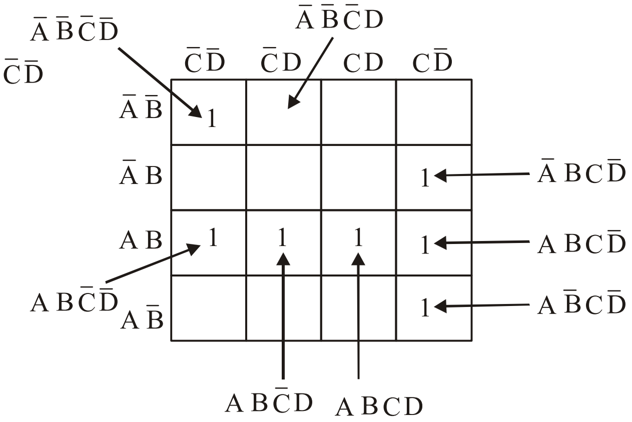

$\overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,\overline{C} + A\,B\,C$
$\overline{C}$
C
$\overline{A}\,\overline{B}$
$\overline{A}\,B$
A B
$A\,\overline{B}$
1
1
1
1

Fig. 4-52.

Example 4-71. *Map the following standard sum-of-products* (*SOP*) *expression on a Karnaugh map* :

$$\overline{A}BC\overline{D} + ABC\overline{D} + AB\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}\overline{D} + AB\overline{C}D + \overline{A}\overline{B}\overline{C}D + A\overline{B}C\overline{D} + ABCD$$

Solution. Sketch a four variable Karnaugh map as shown in Fig. 4-53. Select the first product term $\overline{A}BC\overline{D}$ from the given *SOP* expression. Check that number of 1*s* in the Karnaugh Map is equal to the number of product terms in the given *SOP* expression.



$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
$\overline{A}\,\overline{B}\,\overline{C}\,D$
$\overline{C}\,\overline{D}$
$\overline{C}\,\overline{D}$
$\overline{C}\,D$
$C\,D$
$C\,\overline{D}$
$\overline{A}\,\overline{B}$
1
$\overline{A}\,B$
1
$\overline{A}\,B\,C\,\overline{D}$
$A\,B$
1
1
1
1
$A\,B\,C\,\overline{D}$
$A\,B\,\overline{C}\,\overline{D}$
$A\,\overline{B}$
1
$A\,\overline{B}\,C\,\overline{D}$
$A\,B\,\overline{C}\,D$
$A\,B\,C\,D$

Fig. 4-53.

4-35. Mapping a Nonstandard SOP Expression on the Karanaugh Map

A nonstandard sum-of-products (*SOP*) expression is a one that has product terms with one or more missing variables. In such a case, Boolean expression must first be converted to a standard form by a procedure explained in Art. 4-17

Let us consider an example to illustrate the procedure for mapping a nonstandard *SOP* expression on the Karnaugh map. Suppose we have the *SOP* expression :

$$\overline{A} + \overline{AB} + AB\overline{C}$$

As seen, this expression is obviously not in standard form because each product term does not have three variables. The first term is missing two variables, the second term is missing one variable and the thired term is standard. In order to convert the given nonstandard *SOP* expression to a standard form, we multiply the first product term by $B+\overline{B}$ and $C+\overline{C}$, and the second term by $C+\overline{C}$. Expanding the resulting expression, we get,

$$\overline{A}\left(B+\overline{B}\right)\left(C+\overline{C}\right)+\overline{AB}\left(C+\overline{C}\right)+AB\overline{C}$$

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

Rearranging the expression for our convenience, we get

$$\overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$$

This expression can be mapped on a three-variable Karnaugh map as shown in Fig. 4-54.

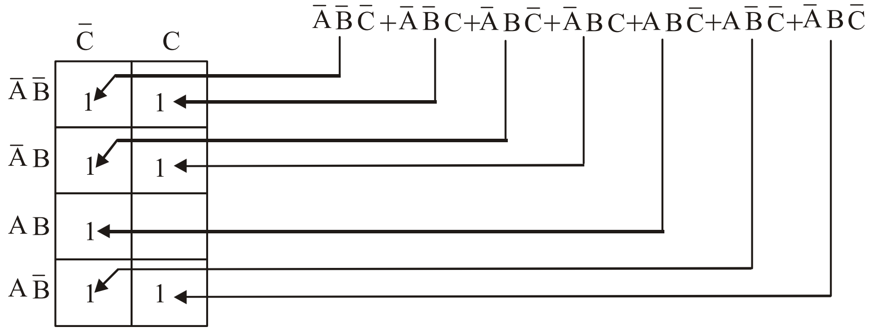

$\bar{A}\bar{B}\bar{C}+\bar{A}\bar{B}C+\bar{A}B\bar{C}+\bar{A}BC+AB\bar{C}+A\bar{B}\bar{C}+\bar{A}B\bar{C}$
$\bar{C}$
C
$\bar{A}\bar{B}$
$\bar{A}B$
AB
$A\bar{B}$
1
1
1
1
1
1
1

Fig. 4-54.

4-36. Simplification of Boolean Expression using Karnaugh Map

The process that result in an expression containing the minimum number of possible terms with the minimum number of variables is called **simplification** or **minimization.** After the *SOP* (or the Boolean) expression has been mapped on the Karnaugh map, there are three steps in the process of obtaining a minimum *SOP* expression. The three steps are : (*a*) grouping the 1*s*, (*b*) determining the product term for each group and (*c*) summing the resulting product terms.

(*a*) *Grouping the* 1*s* : We can group the 1*s* on the Karnaugh map according to the following rules by enclosing those adjacent squares containing 1*s*. The objective is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1, 2, 4, 8 or 16 squares. In the case of two-variable Karnaugh map, 4 squares is the maximum group, for three-variable map, 8 squares are the maximum group and so on.
2. Each square in the group must be adjacent to one or more squares in that same group but all squares in the same group do not have to be adjacent to each other.
3. Always include the largest possible number of 1*s* in a group in accordance with rule 1.
4. Each 1 on the Karnaugh map must be included in at least one group. The 1*s* already in a group can be included in another group as long as the overlapping groups include non-common 1*s*.

(*b*) *Determining the Product Term for each group* : Following are the rules that are applied to find the minimum product terms and the minimum sum-of-products expression :

1. *Group the squares that have* 1*s* : Each group of squares containing 1*s* creates one product term composed of all variables that occur in only one form (either uncomplemented or complemented) within the group are eliminated. These are known as contradictory variables.
2. In order to determine the minimum product term for each group, we need to look at the standard methodology for three-variable and four-variable Karnaugh map respectively.
   (*i*) *For a three-variable K-map* : (1) for 1-square group we get a three-variable product term, (2) for a 2-square group, we get a two-variable product term, (3) for a 4-square product term, we get a one-variable product term.
   (*ii*) *For a four-variable K-map* : (1) For a 1-square group, we get a four-variable product term, (2) for a 2-square group, we get a three-variable product term, (3) for a 4-square group, we get a two-variable product term, and (4) for a 8-square group, we get a one-variable product term.

(*c*) *Summing the resulting product terms* : When all the minimum product terms are derived from the Karnaugh map, these are summed to form the minimum sum-of-products expression.

Note. In some cases, there may be more than one way to group the 1*s* to form the product terms. Whatever be the way, the minimal expression must have the same number of product terms and each product term, the same number of Boolean variables.

The examples given below will help you to understand and apply the simplification of the *SOP* expression using Karnaugh map.

4-37. The Five-variable Karnaugh Map

Fig. shows a five-variable Karnaugh map (K-map). This map is not as simple to use as map for four or fewer variables. A five-variable map needs 32 squares ($2^5 = 32$) and a six variable map needs 64 squares ($2^6 = 64$).

A five-variable map is shown in Fig. 4-55. If consists of 2 four-variable maps with variables *A*, *B*, *C*, *D* and *E*. Variable *A* distinguishes between the two maps. One four variable map represents the 16 squares in which *A* = 0 and the other four variable map represents the squares in which *A* = 1. Minterms 0 through 15 are indicated in K-map with *A* = 0 and min terms 16 through 31 in K-map with *A* = 1. Each four variable map retians the previously defined adjacexey when taken separately. In addition, each square in the *A* = 0 map is adjacent to the corresponding square in the *A* = 1 map. For example, minterms 4 is adjacent to minterms 20 and minterm 15 to 31. Consider the two half maps as being one on top of the other one as shows in Fig. 4-56. Any squares that fall over the other are considered adjacent.

Four variable map in which A = 0.

A = 0

D

BC \ DE	00	01	11	10
00	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}\,\bar{E}$	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}\,E$	$\bar{A}\,\bar{B}\,\bar{C}\,D\,E$	$\bar{A}\,\bar{B}\,\bar{C}\,D\,\bar{E}$
01	$\bar{A}\,\bar{B}\,C\,\bar{D}\,\bar{E}$	$\bar{A}\,\bar{B}\,C\,\bar{D}\,E$	$\bar{A}\,\bar{B}\,C\,D\,E$	$\bar{A}\,\bar{B}\,C\,D\,\bar{E}$
11	$\bar{A}\,B\,C\,\bar{D}\,\bar{E}$	$\bar{A}\,B\,C\,\bar{D}\,E$	$\bar{A}\,B\,C\,D\,E$	$\bar{A}\,B\,C\,D\,\bar{E}$
10	$\bar{A}\,B\,\bar{C}\,\bar{D}\,\bar{E}$	$\bar{A}\,B\,\bar{C}\,\bar{D}\,E$	$\bar{A}\,B\,\bar{C}\,D\,E$	$\bar{A}\,B\,\bar{C}\,D\,\bar{E}$

B C

E

Four variable map in which A = 1.

A = 1

D

BC \ DE	00	01	11	10
00	$A\,\bar{B}\,\bar{C}\,\bar{D}\,\bar{E}$	$A\,\bar{B}\,\bar{C}\,\bar{D}\,E$	$A\,\bar{B}\,\bar{C}\,D\,E$	$A\,\bar{B}\,\bar{C}\,D\,\bar{E}$
01	$A\,\bar{B}\,C\,\bar{D}\,\bar{E}$	$A\,\bar{B}\,C\,\bar{D}\,E$	$A\,\bar{B}\,C\,D\,E$	$A\,\bar{B}\,C\,D\,\bar{E}$
11	$A\,B\,C\,\bar{D}\,\bar{E}$	$A\,B\,C\,\bar{D}\,E$	$A\,B\,C\,D\,E$	$A\,B\,C\,D\,\bar{E}$
10	$A\,B\,\bar{C}\,\bar{D}\,\bar{E}$	$A\,B\,\bar{C}\,\bar{D}\,E$	$A\,B\,\bar{C}\,D\,E$	$A\,B\,\bar{C}\,D\,\bar{E}$

B C

E

Fig. 4-55.

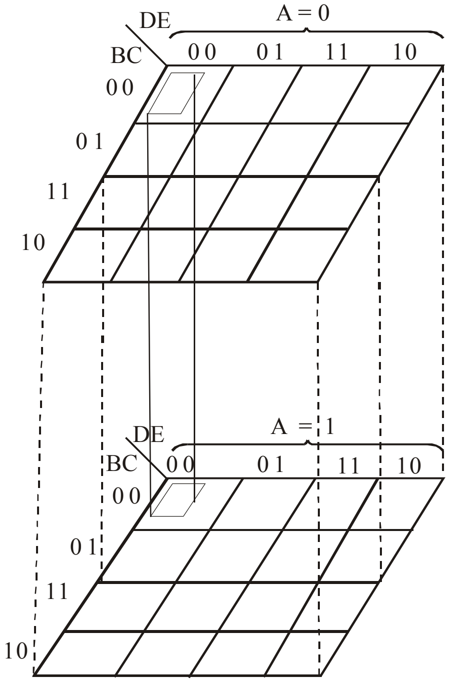


Fig. 4-56.

Example 4-72. *Simplify the following Boolean expression using the Karnaugh mapping technique :*

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

Solution. The first step is to map the given Boolean expression on the Karnaugh map. Notice that there are three variables *A*, *B* and *C* in the Boolean expression, therefore we need a three-variable Karnaugh map.

The Boolean expression to be mapped is,

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

Note that the given Boolean expression is a nonstandard *SOP* expression because the first product term $\overline{A}B$ has the variable *C* missing in it. This can be converted into a standard *SOP* form by modifying the expression as below.

$$\begin{aligned} X &= \overline{A}B.1 + \overline{A}\,\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C} \\ &= \overline{A}B\left(C+\overline{C}\right) + \overline{A}\,\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C} && \ldots\left(C+\overline{C}=1\right) \\ &= \overline{A}BC + \overline{A}B\overline{C} + \overline{A}\,\overline{B}C + AB\overline{C} + A\overline{B}\,\overline{C} && \ldots(i) \end{aligned}$$

4-38. The Six-variable Karnaugh Map

Fig. 4-57 shown a six-variable Karnaugh map. This map is much more complex. A six- variable map needs 64 squares ($2^6 = 64$). If consist of 4- four variable maps with variables *A*, *B*, *C*, *D*, *E* and *F*. Each map represents four combinations of *A* and *B*. The adjacent squares can be grouped together. The first column with 4th column, 2nd column with 7th column, 3rd column with 6th column, 1st row with 4th row, 2nd row with 7th row, and 3rd row with 6th row can be combined together to get the minterms.

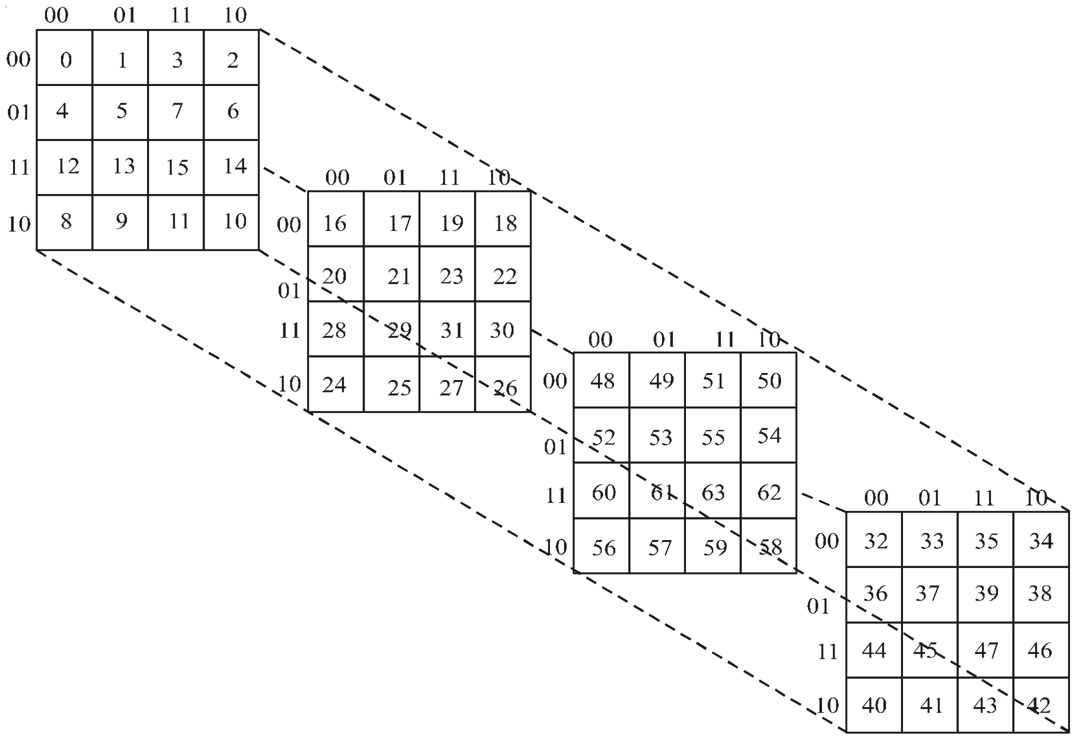


Fig. 4.57. *Six-variable Karnaugh Map.*

Equation (*i*) can be mapped on the Karnaugh map shown in Fig. 4-50. In order to simplify the given expression, the 1*s* can be grouped together as shown by the loop around the 1*s*. The four 1*s* in the first column are grouped together and the term we get is $\bar{C}$. This is because of the fact that the squres within this group contain both *A* and $\bar{A}$ and *B* and $\bar{B}$, so these variables are eliminated. Similarly, the two 1*s* in the second row are grouped together and the term we get is $\bar{A}B$. This is because of the fact that squares in this group contain both *C* and $\bar{C}$ which is eliminated. Summing up the two-product terms, the simplified expression is,

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

Example 4-73. *Simplify the following SOP expresssion using the Karnaugh mapping procedure* :

$$X = \bar{A}B\bar{C}D + A\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}D + AB\bar{C}D + AB\bar{C}\bar{D} + ABCD$$

Solution. First of all, notice that the given *SOP* expression is already in the standard from *i.e.* all the product terms in the given expression have all four variables *A*, *B*, *C* and *D*.

Next sketch a four-variable Karnaugh map. Select the first product term $(\bar{A}B\bar{C}D)$ from the given expression and enter 1 in Fig. 4-59. Similarly enter 1 for the other product terms in the given *SOP* expression to complete the mapping. In order to simplfy the given *SOP* expresssion, the 1*s* can be grouped together as shown be the loop around the 1*s*. The four 1*s* in the second column are grouped together and the product term we get $\bar{C}D$. This is because of the fact that squares within this group contain both *A* and $\bar{A}$ and *B* and $\bar{B}$, so these variables are eliminated.

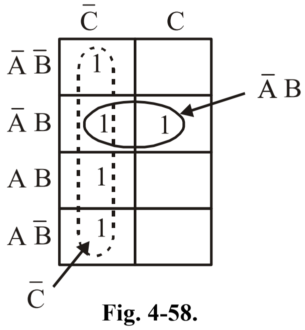


Fig. 4-58.

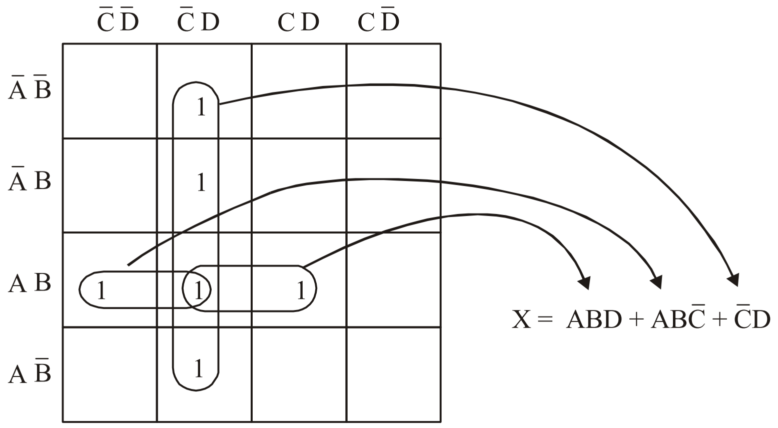


Fig. 4-59.

The two 1*s* in the first and second columns can be grouped together. This group contains both *D* and $\overline{D}$, so this variable is eliminated and the resulting product term is $AB\overline{C}$. Similarly the two 1*s* in the second and third columns can be grouped together. This group contains both *C* and $\overline{C}$, so this variable is eliminated and the resulting product term is *ABD*.

The resulting minimal or simplified *SOP* expression is obtained by summing up the three product terms $\overline{C}D, AB\overline{C}$ and ABD as shown below :

$$X = ABD + AB\overline{C} + \overline{C}D.$$

Example 4-74. *Simplify the following SOP expression using the Karnaugh mapping tehnique.*

$$X = B\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}BCD + ABCD$$

Solution. First of all, notice that the given *SOP* expression is in the nonstandard *SOP* form because the first product term $\left(B\overline{C}\overline{D}\right)$ has a variable *A* and $\overline{A}$ missing in it. Let us convert the given *SOP* expression into a standard *SOP* expression into a standard *SOP* form as shown below :

$$X = 1.B\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}BCD\;ABCD$$

$$= \left(\overline{A} + A\right)BCD + ABCD + ABCD + ABCD + ABCD \qquad ...\left(\because \overline{A} + A = 1\right)$$

$$= \overline{A}B\overline{C}\overline{D} + AB\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}BCD + ABCD$$

This expression can be mapped on to the four-variable Karnaugh by entering 1 for each product term in the corresponding square as shown in Fig. 4-60.

In order to simplify the *SOP* expression, the 1*s* can be grouped together as shown by the loop around the 1*s*. *The four* 1*s* looped together form the first and second columns, contain both *A* and $\overline{A}$ and *D* and $\overline{D}$, so these variables are eliminated and the resulting product term is $B\overline{C}$. Similarly, the four 1*s* looped together form the second and the third column contain both *A* and $\overline{A}$ and C and $\overline{C}$ so these variables are eliminated and the resulting product term is *BD*. The resulting simplified *SOP* expression is the sum of the product term $B\overline{C}$ and *BD*, *i.e.,*

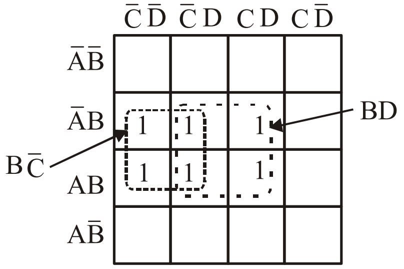


Fig. 4-60.

$$X = B\overline{C} + BD$$

Example 4-75. *Fig.* 4-61 *shows a Karnaugh map of a sum-of-products* (*SOP*) *function. Determine the simplified SOP function.* *(UPSC Civil Services 2000)*

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1		1
$\bar{A}B$		1		
AB			1	
$A\bar{B}$	1	1		1

Fig. 4-61.

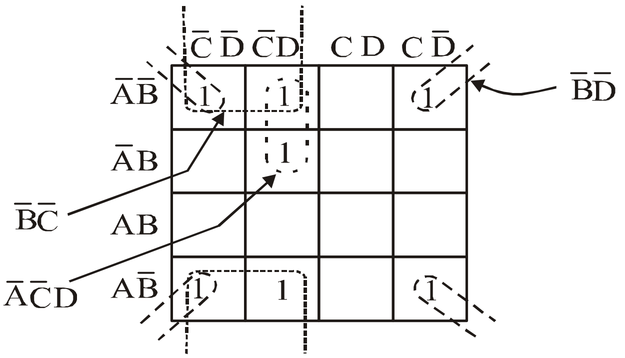


Fig. 4-62.

Solution. The grouping of 1*s* is as shown in Fig. 4-62. Notice the "wrap around" four-square group that includes the 1s on four corners of the Karnaugh map. This group produces a product term $\overline{B}\overline{D}$. This is determined by observing that the group contains both *A* and $\overline{A}$ and *C* and $\overline{C}$, so these variables are eliminated.

Another group of four with "wrap-around" adjacency is formed form the top and the bottom rows of the Karnaugh map. This group overlaps with the previous group and produces a product term $\overline{B}\overline{C}$. This is determined by observing that this group contains both *A* and $\overline{A}$ and *D* and $\overline{D}$, so these variables are eliminated.

The remaining 1 is absorbed in a overlapping group of two squares. This group produces a three-variable term $\overline{A}\overline{C}D$. This is determined by observing that this group contains both *B* and $\overline{B}$, so this variable is eliminated. This resulting simplified *SOP* function is the sum of the product terms $\overline{B}\overline{D}+\overline{B}\overline{C}$ and $\overline{A}\overline{C}D$, *i.e.*

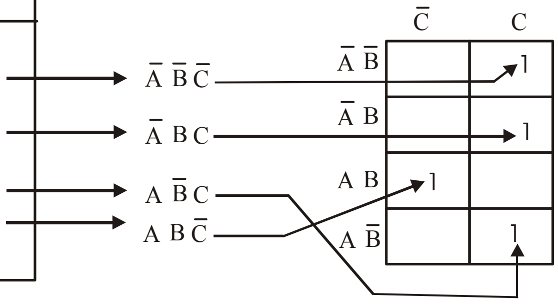
$$X=\overline{B}\overline{D}+\overline{B}\overline{C}+\overline{A}\overline{C}D$$

4-39. Mapping Directly on Karnaugh Map from a Truth Table

It is possible to map directly on Karnaugh map from a truth table. Recall that a truth table. Recall that a truth table gives the output of a Boolean expression for all possible input variable combinations. Let us illustrate direct mapping through an example of a Boolean expression and its trugh table representation.

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

(a)



(b)

Fig. 4-63.

Let $X = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}$. Then its truth table can be indicated as shown in Fig. 4-63 (*a*) and its Karnaugh mapping is shown in Fig. 4.55 (*b*). Notice in the truth table that the output *X* is 1 for four different input variable combinations.

It is evident from the Fig. 4.55 (*a*) and (*b*) that truth table and Karnaugh map are simply different ways to represent a logic fuction.

Example 4.76. *Reduce the following function using k map and realize using NAND gates only.*

$$f(ABC) = AB\overline{C} + \overline{A}\overline{B}C + ABC + A\overline{B}C \qquad \text{(Anna University, May/June 2006)}$$

Solution: The given Booleans function indicates that its output is 1 corresponding to the term indicated within the expression $AB\overline{C}$, $\overline{A}\overline{B}C$, ABC and $A\overline{B}C$. Next sketch a 3 variable kaenaugh map. Select the first product term ($AB\overline{C}$) from the given expression and enter 1 in Fig. 4-64. Similarly enter 1 for the other product terms in the given expression.

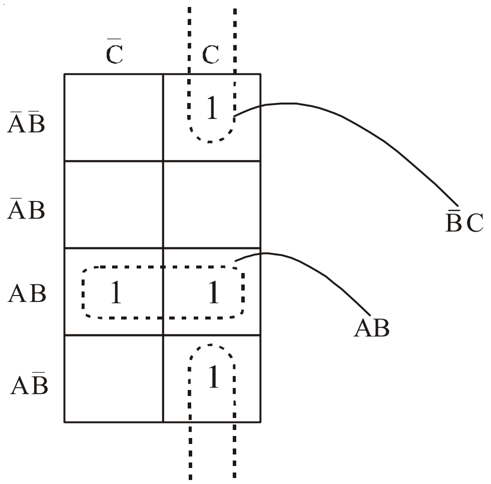


Fig. 4-64

In order to simplify the Boolean expression respresented on the karnaugh map, group the 1's as shown in fig. the group of two squares in the first coloum produces $\overline{B}C$. Another group of two squares in the third row produces term AB.

$$f(A, B, C) = \text{AB} + \overline{B}C$$

Let us first of all, implement this circuit using the basic logic gates i.e AND OR and INVERTER gates as shown in Fig. 4.65.

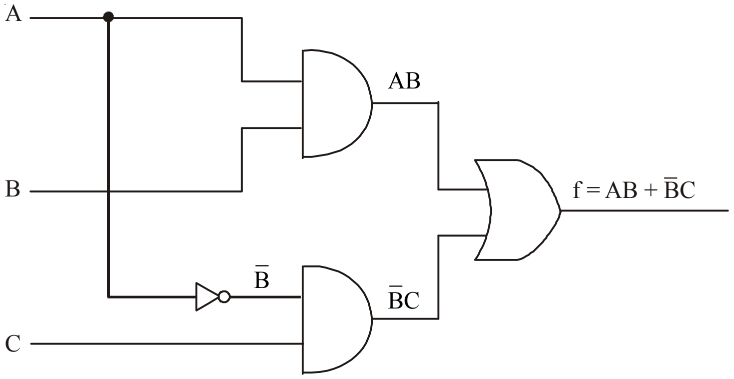


Fig. 4-65

Now we can replace each login gates by it NAND gate equivalen. Notice that INVERTER replaced by a single NAND gate each AND is replaced by two NAND gates which the or gate is replaced by tow NAND gates. The resulting circuit is shown in Fig. 4.66.

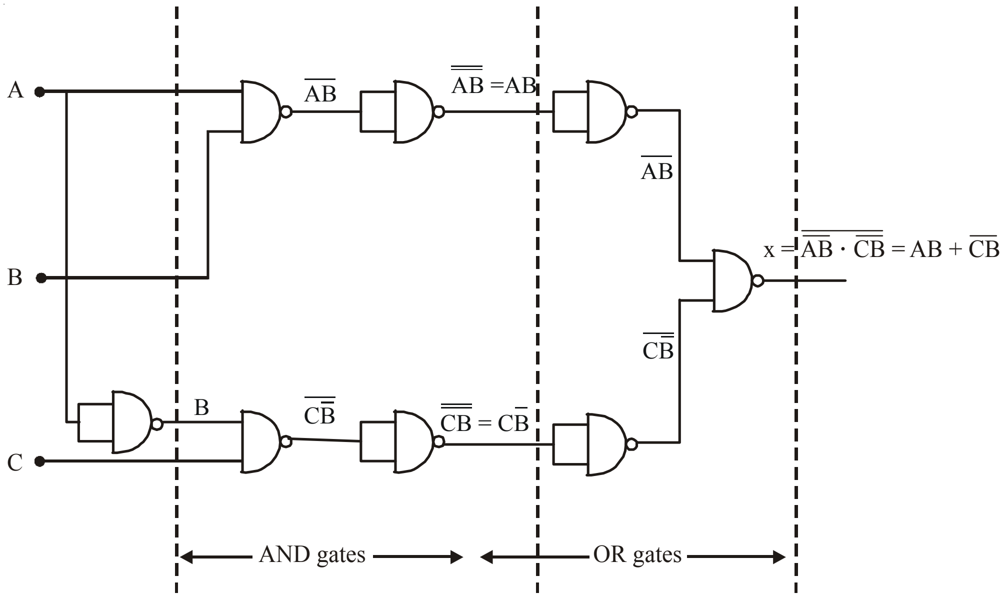

A
B
C
$\overline{AB}$
$\overline{\overline{AB}}$ =AB
$\overline{AB}$
x = $\overline{\overline{AB} \cdot \overline{C\bar{B}}}$ = AB + C$\bar{B}$
$\overline{C\bar{B}}$
B
$\overline{C\bar{B}}$
$\overline{\overline{C\bar{B}}}$ = C$\bar{B}$
AND gates
OR gates

Fig. 4-66

Removing the two INVERTER on the same line we can simplify the circuit shown in Fig. 4.67.

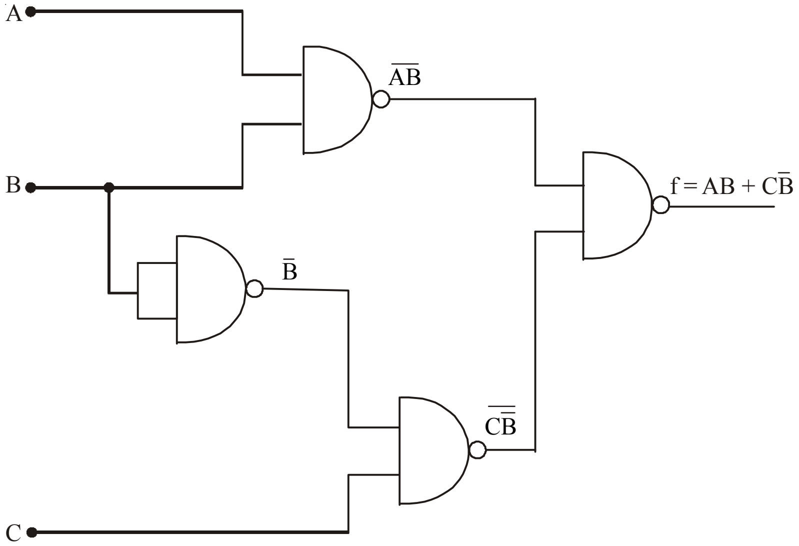


Fig. 4-67

Example 4.77. *Show that*

$f(A, B, C, D) = \Sigma m$ (0, 3, 5, 6, 9, 10, 12, 15) equals to $A \odot B \odot C \odot D$ using K-map

(*Nagpur University, 2007*)

Solution. Given: The function

$$f = (A, B, C, D) = \Sigma\, m\, (0, 3, 5, 6, 9, 10, 12, 15)$$

The Boolean function indicate that the output is 1 correspondingly to the term indicated with in the expression *i.e.* 0, 3, 5, 6, 9, 10, 12, and 15. We can map then value directly to the four variable karnaugh map shown in fig. In order to simplify the Boolean expression represented on the karnaugh map group the 1_s as shown in Fig. 4-68.

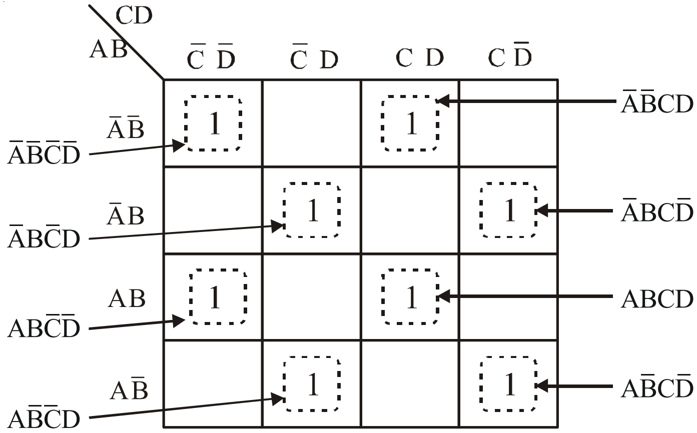


Fig. 4-68

The resulting simplified expression,

$$f(A,B,C,D) = \bar{A}\bar{B}\bar{C}\bar{D}+\bar{A}\bar{B}CD+\bar{A}B\bar{C}D+\bar{A}BC\bar{D} + AB\bar{C}\bar{D}+ABCD+A\bar{B}\bar{C}D+A\bar{B}C\bar{D}$$

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

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

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

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

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

$$= (A+B)\bullet(C+D)$$

$$= A+B+C+D$$ **Ans.**

Example 4-78. *A four variable functions is given as f (A, B, C, D) = Σm (0, 2, 3, 4, 5, 7, 9, 13, 15). Use a k-map to minimize the function.*

(*PTU, Dec. 2008*)

Solution. The given Boolean function indicates that its output is 1corresponding to the term indicated within the expression i.e. 0, 2, 3, 4, 5, 7, 9, 13, and 15. This is shown in Fig. 4-69. We can map these values directly on to the four-variable karnaugh mas as shown in Fig 4.70. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in Fig 4.70.

Decimal Number	*Inputs*				*Outputs*	
	A	*B*	*C*	*D*	*F*	
0	0	0	0	0	1	$\overline{A}\overline{B}\overline{C}\overline{D}$
1	0	0	0	1	0	
2	0	0	1	0	1	$\overline{A}\overline{B}C\overline{D}$
3	0	0	1	1	1	$\overline{A}\overline{B}CD$
4	0	1	0	0	1	$\overline{A}B\overline{C}\overline{D}$
5	0	1	0	1	1	$\overline{A}B\overline{C}D$
6	0	1	1	0	0	
7	0	1	1	1	1	$\overline{A}BCD$
8	1	0	0	0	0	
9	1	0	0	1	1	$A\overline{B}\overline{C}D$
10	1	0	1	0	0	
11	1	0	1	1	0	
12	1	1	0	0	0	
13	1	1	0	1	1	$AB\overline{C}D$
14	1	1	1	0	0	
15	1	1	1	1	1	$ABCD$

Fig. 4-69

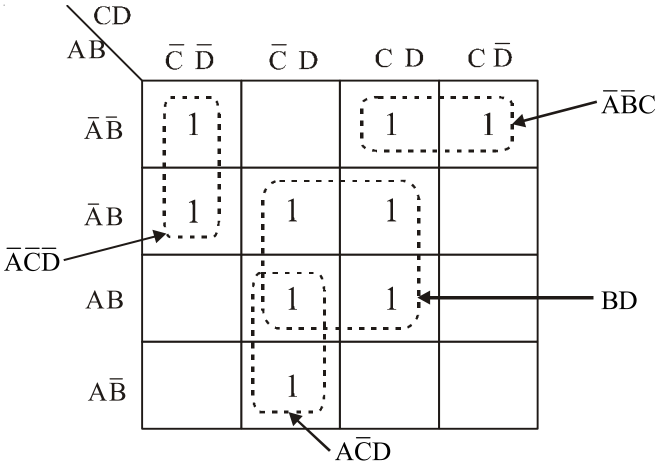


Fig. 4-70

The group of four sequares in the seuan and third column produces a product term BC. This is determined by observing that the group contains A and $\overline{A}$ and D and $\overline{D}$ so these variable are dominated and the resulting product term is BD. Another group of two squares in first column produces term $\overline{A}\overline{C}\overline{D}$. The variable B is diminated became the group contains both B and $\overline{B}$. Similarly the group of two squares in column second produces the term $A\overline{C}D$. The variable B is eliminated because the group contains both B and $\overline{B}$.

Another group of two squares in first row produces the term $\overline{A}\overline{B}C$. The variable D is eliminated because the group contains D and $\overline{D}$. Thus the resulting simplified expression.

$$f(A, B, C, D) = \overline{A}\overline{C}\overline{D} + \overline{A}\overline{B}C + BD + A\overline{C}D \text{ **Ans.**}$$

Example 4.79. *Minimize the four variable logic function using k-map f (A, B, C, D) = Σm (0, 1, 3, 5, 7, 8, 9, 11, 14).* *(Anna University, April/May 2008)*

Solution. The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e 0, 1, 3, 5, 7, 8, 9, 11, and 14. This is shown in Fig. 4-71. We can map these values directly on to the four-variable karnough map as shown in Fig. 4-72. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in Fig. 4-72.

Decimal Number	*Inputs*				*Outputs*	
	A	*B*	*C*	*D*	*F*	
0	0	0	0	0	1 →	$\bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	1 →	$\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	0	
3	0	0	1	1	1 →	$\bar{A}\bar{B}CD$
4	0	1	0	0	0	
5	0	1	0	1	1 →	$\bar{A}B\bar{C}D$
6	0	1	1	0	0	
7	0	1	1	1	1 →	$\bar{A}BCD$
8	1	0	0	0	1 →	$A\bar{B}\bar{C}\bar{D}$
9	1	0	0	1	1 →	$A\bar{B}\bar{C}D$
10	1	0	1	0	0	
11	1	0	1	1	1 →	$A\bar{B}CD$
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	1 →	$ABC\bar{D}$
15	1	1	1	1	0	

Fig. 4-71

The group of four 1s looped together forn the second and third columns contain both *B* and $\bar{B}$ and *C* and $\bar{C}$, so these variable are eliminated and resulting product term is $\bar{A}D$.

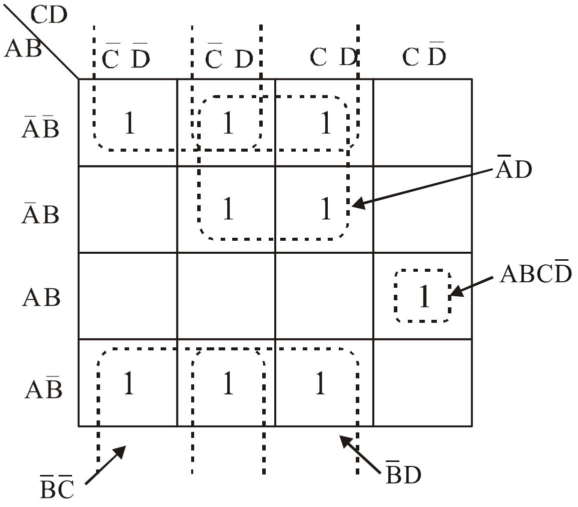


Fig. 4-72

Similarly another group with "wrap-around" adjacency is formed form the top and the bottom rows of the karnaugh map: This group overlaps with the previous group and produces a product term $\overline{B}D$. This is determined by observing that this group contain both A and $\overline{A}$ and C and $\overline{C}$ so these variable are eliminated. Similarly from the other group it produce a product of term $\overline{B}\overline{C}$.

Another group of one square in the fourth column and third row produces the term $ABC\overline{D}$. Thus the resulting simplified expression,

$$f(A,B,C,D) = \overline{A}D + \overline{B}C + \overline{B}\overline{C} + ABC\overline{D}$$ **Ans.**

Example 4-80. *Using K-map obtain the minimal sum of product and minimal product of sums of the function*

f (A, B, C, D) = Σm (1, 2, 3, 5, 6, 7, 8, 13) *(VTU., Jan./ Feb. 2004)*

Solution. Given : The function

$f(A, B, C, D) = \Sigma m\ (1, 2, 3, 5, 6, 7, 8, 13).$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e 1, 2, 3, 5, 6, 7, 8, and 13. As shown in Fig. 4-73. The function is form of SOP, we can convert the SOP in form of POS. The POS is 0 corresponding to the term 0, 4, 9, 10, 11, 12, 14, 15.

$f(A, B, C, D) = \pi\, M\ (0, 4, 9, 10, 11, 12, 14, 15)$

The table for the function is shown in Fig. 4-73.

Decimal		*Inputs*			*Outputs*
Number	***A***	***B***	***C***	***D***	***F***
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

Fig. 4-73

To simplify the SOP, we can map SOP values directly on to the four variable karnaugh map as shown in Fig. 4-74. In order to simplify the Boolean expression represented on the karnaugh map group the 1s as shown in the Fig. 4-74.

The minimal sum of products

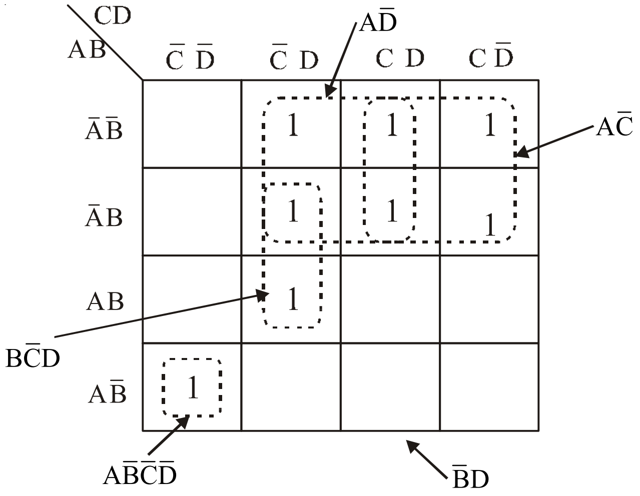


Fig. 4-74

$$f(A, B, C, D) = A\overline{B}\overline{C}\overline{D} + \overline{A}D + \overline{A}C + B\overline{C}D \textbf{ Ans.}$$

To simplify the POS, we can map POS values directly on the four variable karnaugh map as shown in Fig. 4-75. This expression can be mapped on the four-variable karnaugh by entering O for each turn in the corresponding square.

The POS function

$$f(A, B, C, D) = \pi M(0, 4, 9, 10, 11, 12, 14, 15)$$

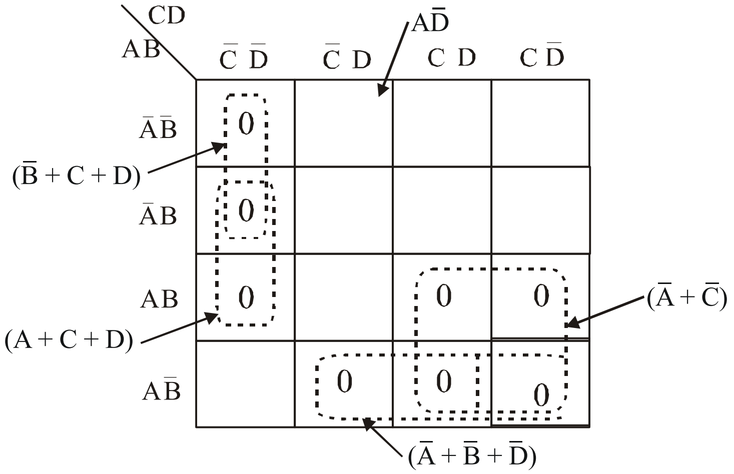


Fig. 4-75

The minimal product of sum.

$$f(A, B, C, D) = (A + C + D)(\overline{B} + C + D)(\overline{A} + B + D)(\overline{A} + C) \textbf{ Ans.}$$

Exmaple 4-80. $y = \Sigma m$ *(0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15)*

Minimize the expression using K-map. *(Gauhati University, 2007)*

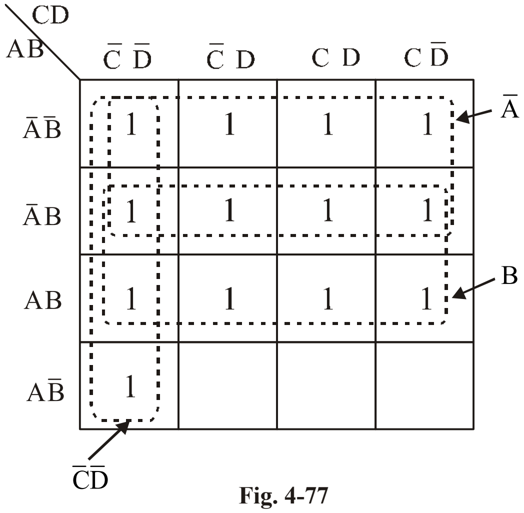
Solution. Given: The expression

$$y = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15)$$

The given expression indicate that its output is 1 corresponding to the term indicated within the expression i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15. This is shown in Fig. 4-76. We can map these value directly on to the four variable karnaugh map as shown in Fig. 4-77,

Decimal Number	*Inputs*				*Outputs*
	A	*B*	*C*	*D*	*F*
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Fig. 4-76.



AB \ CD	$\overline{C}\,\overline{D}$	$\overline{C}\,D$	$C\,D$	$C\,\overline{D}$
$\overline{A}\,\overline{B}$	1	1	1	1
$\overline{A}B$	1	1	1	1
AB	1	1	1	1
$A\overline{B}$	1			

Fig. 4-77

To simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in Fig. 4-77. The group of eight squares in the first and second row produces a term $\overline{A}$. Similar the group of eight squares in the second and third produces a term B. The group of four square in the first column produce a term $\overline{C}\overline{D}$. Thus the simplified expression,

$$f(A, B, C, D) = \overline{A} + B + \overline{C}\,\overline{D} \text{ } \textbf{Ans.}$$

Example 4-82. *Simplify the following expression using k-map and realize using AND-OR realization*

$f(A,B,C,D) = \Sigma m\ (1, 5, 7, 9, 11, 13, 15)$ *(Nagpur University, 2004)*

Solution: Given. The expression

$$f(A,B,C,D) = \Sigma m(1, 5, 7, 9, 11, 13, 15)$$

The given Boolean function indicates that it output is 1 corresponding to the term indicated within the expression i.e. 1, 5, 7, 9, 11, 13, and 15. This is shown in Fig. 4-78. We can map these values directly on the four-variable karnaugh map as shown in Fig. 4-79. In order to simplify the Boolean expression represented on the karnaugh map. group the 1s as shown in the Fig. 4-79.

Decimal	*Inputs*				*Outputs*
Number	*A*	*B*	*C*	*D*	*F*
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Fig. 4.78

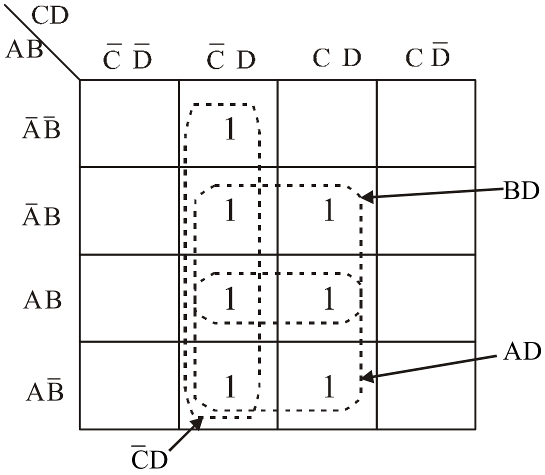


Fig. 4-79

The resulting simplified logic function is the sum of the three product terms $\overline{C}D$, BD and AD, i.e

$$f(A,B,C,D) = \overline{C}D + AD + BD \text{ Ans.}$$

The function is implement using the logic gates. i.e AND and OR as shown in Fig. 4-80. The term $\overline{C}D$, AD and BD is obtained by using AND gate and final output function is obtained by OR gate.

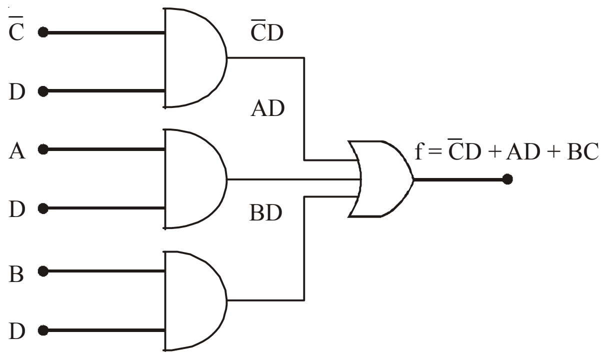


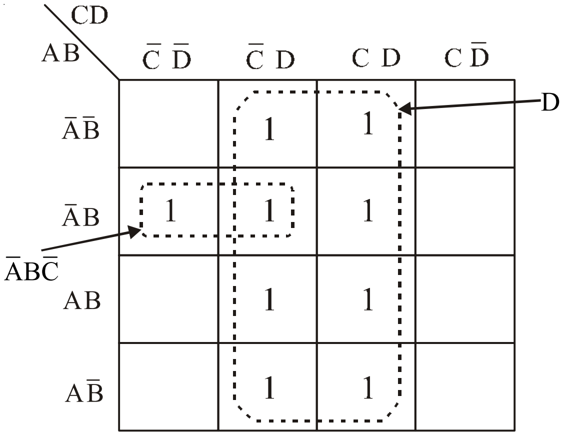
Fig. 4-80

Example 4-83. *Using k-map realize the following expression using minimum number of gates*

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

(*Gujrat Technological University., Dec. 2009*)

Solution. The given SOP expression is the standard SOP form. This expression can be mapped on to the four variable karnaugh by entering 1 for each product term in the corresponding square as shown in Fig. 4-81.



AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$		1	1	
$\bar{A}B$	1	1	1	
AB		1	1	
$A\bar{B}$		1	1	

Fig. 4-81

In order to simplify the SOP expression the 1s can be grouped together as shown is by the loop around the 1s. The eight 1s looped together form the second and third columns, contain both A and $\bar{A}$ and B and $\bar{B}$ and C and $\bar{C}$ so these variables are eliminated and the resulting product term is D. Similarly the two 1s looped together form the second and third column contain both D and $\bar{D}$ so these variable are eliminated and resulting product terms is $\bar{A}B\bar{C}$. The resulting simplified SOP expression is the sum of product term D and $\bar{A}B\bar{C}$ *i.e.*

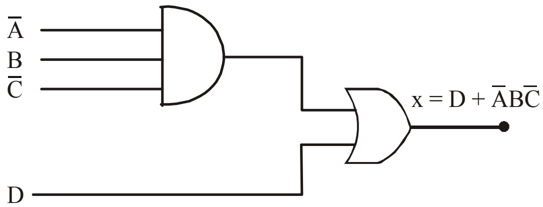


Fig. 4-82

$$X = D + \bar{A}B\bar{C} \ \textbf{Ans.}$$

This can be implemented using logic gate as shown in Fig. 4-82.

Example 4-84. *Using k-map determine the minimal sum of product expression and realize the simplified expression using only NAND gates.*

$f(w, x, y, z) = \pi m\ (0, 2, 3, 7, 8, 9, 10)$ (*VTU., July/Aug. 2004*)

Solution. Given: The expression

$$f(w, x, y, z) = \pi m\,(0, 2, 3, 7, 8, 9, 10)$$

The function is in the form of product of sum. There are four variables w, x, y and z there for the total possible combinations an $2^4 = 16$. The POS contain seven terms so the SOP contains the remaining term.

SOP term : 1, 4, 5, 6, 11, 12, 13, 14, 15.

$$f(\text{w, x, y, z}) = \Sigma\text{m}\,(1, 4, 5, 6, 11, 12, 13, 14, 15)$$

The function indicate that is output is 1 corresponding to the term indicated with in the SOP expression i.e. 1, 4, 5, 6, 11, 12, 13, 14 and 15. This is shown in Fig. 4-83. We can map these value directly on to the four-variable karnaugh map as shown in Fig. 4-84. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig 4-84. The group of two square in the second column produces a product term $\overline{wy}x$. The group of two squares on the third column produce the term *wyz*.

The group of the four square in the third row produce the term *wx*. The four square in the column first and fourth produce the term $x\overline{z}$.

Decimal Number	***Inputs*** *w*	*x*	*y*	*z*	***Outputs F***	
0	0	0	0	0	0	
1	0	0	0	1	1 →	$\overline{w}\overline{x}\,\overline{y}z$
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	1 →	$\overline{w}x\overline{y}\overline{z}$
5	0	1	0	1	1 →	$\overline{w}x\overline{y}z$
6	0	1	1	0	1 →	$\overline{w}xy\overline{z}$
7	0	1	1	1	0	
8	1	0	0	0	0	
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	1 →	$w\overline{x}yz$
12	1	1	0	0	1 →	$wx\overline{y}\overline{z}$
13	1	1	0	1	1 →	$wx\overline{y}z$
14	1	1	1	0	1 →	$wxy\overline{z}$
15	1	1	1	1	1 →	$wxyz$

Fig. 4.83

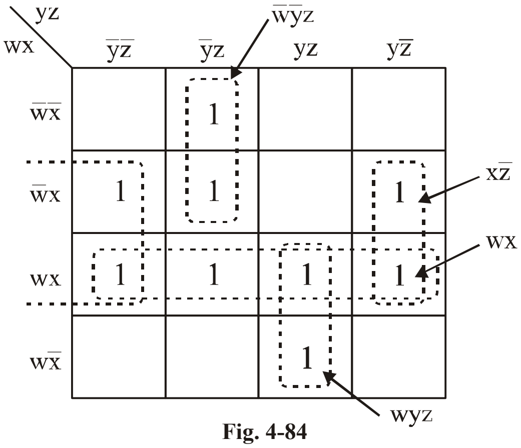


Fig. 4-84

The resulting simplified expression,

$$f(w, x, y, z) = wx + x\overline{z} + wyz + \overline{w}\overline{y}z$$

This can be implemented using NAND gate as show in Fig. 4-85.

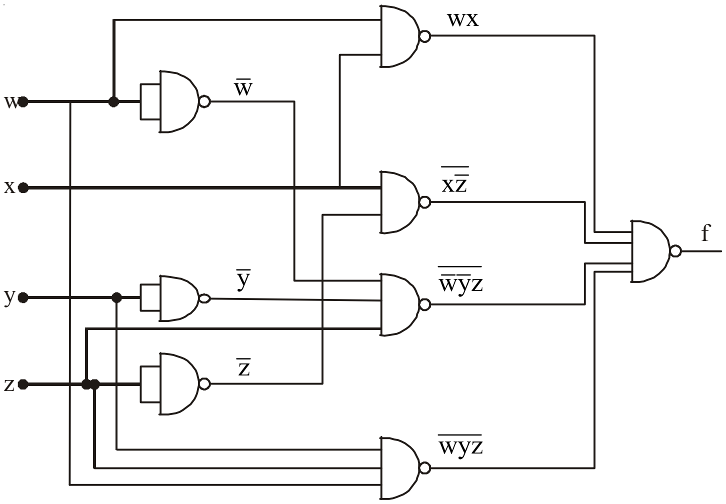

wx
w
w
x
$\overline{x\overline{z}}$
f
$\overline{\overline{w}\overline{y}z}$
y
y
z
z
$\overline{wyz}$

Fig. 4-85

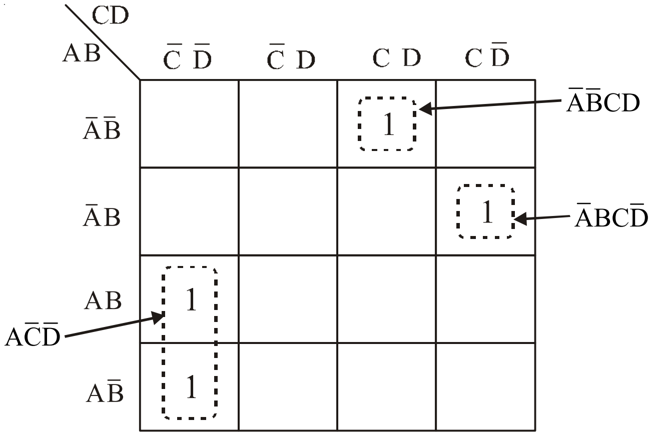
Example 4-85. $Y = \overline{A}\overline{B}CD + \overline{A}BC\overline{D} + A\overline{B}\overline{C}\overline{D} + AB\overline{C}\overline{D}$ *using k-map simplify the function and implement the minimal function using (i) NAND gate only (ii) NOR gates only.*

(Mahatma Gandhi University. May 2005)

Solution. Given: The function

$$Y = \overline{A}\overline{B}CD + \overline{A}BC\overline{D} + A\overline{B}\overline{C}\overline{D} + AB\overline{C}\overline{D}$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e $\overline{A}\overline{B}CD$, $\overline{A}BC\overline{D}$, $A\overline{B}\overline{C}\overline{D}$ and $AB\overline{C}\overline{D}$ .We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-86. In order to simplify the Boolean expression represented on the karnaugh map. group the 1s as shown in the Fig. 4-86.



AB \ CD	$\overline{C}\,\overline{D}$	$\overline{C}\,D$	$C\,D$	$C\,\overline{D}$
$\overline{A}\overline{B}$			1	
$\overline{A}B$				1
AB	1			
$A\overline{B}$	1			

$\overline{A}\overline{B}CD$
$\overline{A}BC\overline{D}$
$A\overline{C}\overline{D}$

Fig. 4-86

The group of two squares in the first column produces term $A\overline{C}\overline{D}$. This is determined by observing that the group contains both B and $\overline{B}$, so this variable is eliminated. Another group of one square in the third column produces term $\overline{A}\overline{B}CD$. similarly another group of one square in the fourth column produce term $\overline{A}BC\overline{D}$. Thus the resulting simplified expression

$$Y = A\overline{C}\overline{D} + \overline{A}\overline{B}CD + \overline{A}BC\overline{D}$$

This can be implemented using NAND gate as shown in Fig 4-87.

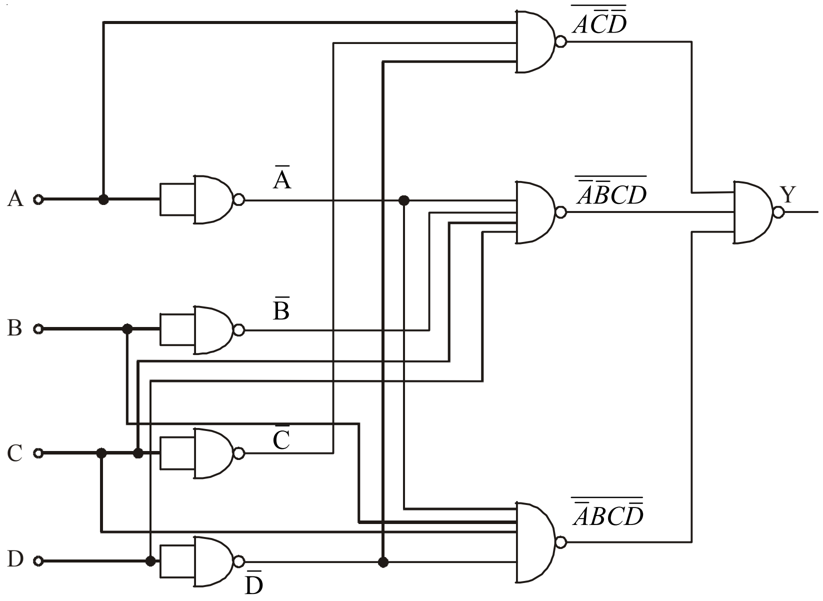

$\overline{A\overline{C}\overline{D}}$
A
$\overline{A}$
$\overline{\overline{A}\overline{B}CD}$
Y
B
$\overline{B}$
C
$\overline{C}$
$\overline{\overline{A}BC\overline{D}}$
D
$\overline{D}$

Fig. 4-87

This can be implemented using NOR gate as shown in Fig 4-88.

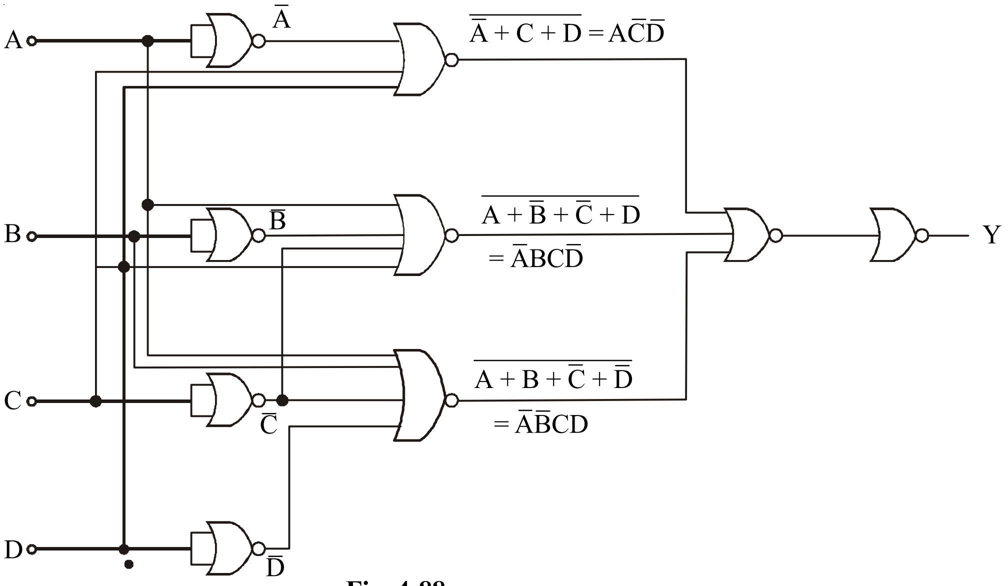

A
$\overline{A}$
$\overline{\overline{A}+C+D}=A\overline{C}\overline{D}$
B
$\overline{B}$
$\overline{A+\overline{B}+\overline{C}+D}$
$=\overline{A}BC\overline{D}$
Y
C
$\overline{C}$
$\overline{A+B+\overline{C}+\overline{D}}$
$=\overline{A}\overline{B}CD$
D
$\overline{D}$

Fig. 4-88

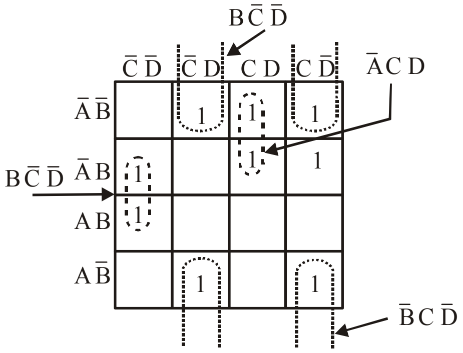
Example 4-86. *Implement the following Boolean expression using minimum number of 3-input NAND gates.*

$$f(A,B,C,D) = \Sigma(1,2,3,4,7,9,10,12) \qquad \textit{(UPSC Engg. Services 1991)}$$

Solution. The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression *i.e.,* 1, 2, 3, 4, 7, 9, 10 and 12. This is shown in Fig. 4-89 (*a*). We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 4-89 (*b*). In order to simplify the Boolean expression represented on the Karnaugh map, group the 1*s* as shown in the Fig. 4-89 (*b*). The group of two squares in the first coloum produces a product term $B\bar{C}\bar{D}$. This is determined by observing that the group contains both *A* and $\bar{A}$, so this variable is eliminated. Another group of two squares in the second column produces term $\bar{B}CD$. The variable *A* is eliminated because the group contains both *A* and $\bar{A}$.

Decimal Number	***Inputs***				***Outputs***	
	A	***B***	***C***	***D***	***F***	
0	0	0	0	0	0	
1	0	0	0	1	1	→ $\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	1	→ $\bar{A}\bar{B}C\bar{D}$
3	0	0	1	1	1	→ $\bar{A}\bar{B}CD$
4	0	1	0	0	1	→ $\bar{A}B\bar{C}\bar{D}$
5	0	1	0	1	0	
6	0	1	1	0	0	
7	0	1	1	1	1	→ $\bar{A}BCD$
8	1	0	0	0	0	
9	1	0	0	1	1	→ $A\bar{B}\bar{C}D$
10	1	0	1	0	1	→ $A\bar{B}C\bar{D}$
11	1	0	1	1	0	
12	1	1	0	0	1	→ $AB\bar{C}\bar{D}$
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	0	

(a)



(b)

Fig. 4-89.

Another group of two squares in the third column produces the term $\bar{A}CD$. The variable *B* is eliminated because the group contains both *B* and $\bar{B}$. Still another group of two squares in the fourth column produces the term $\bar{B}C\bar{D}$. The variable *A* is eliminated because the group contains both *A* and *A*. Thus the resulting simplified expression,

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

This can be implemented using 3-input NAND gate as shown as shown in Fig. 4-90.

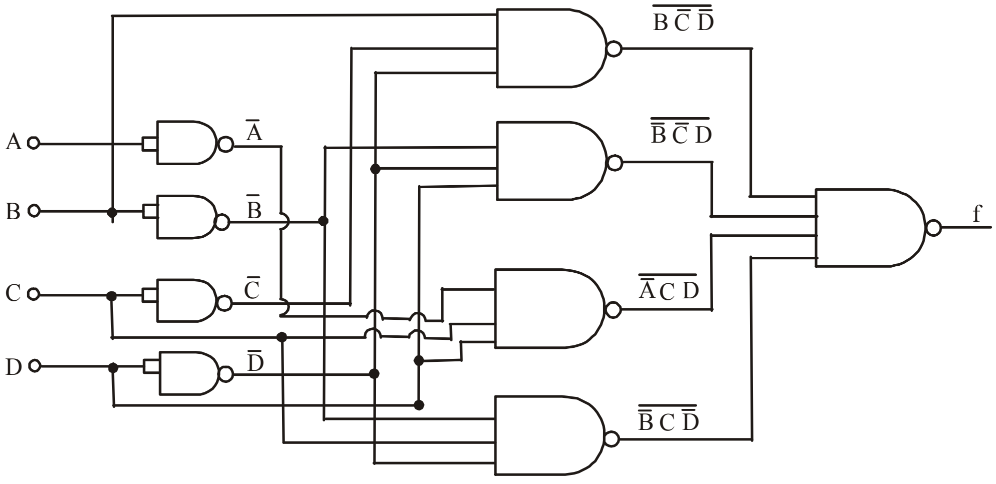

$\overline{B\bar{C}\bar{D}}$
$\overline{\bar{B}\bar{C}D}$
$\overline{\bar{A}CD}$
$\overline{\bar{B}C\bar{D}}$
A
B
C
D
$\bar{A}$
$\bar{B}$
$\bar{C}$
$\bar{D}$
f

Fig. 4-90.

4-40 "Don't Care" conditions

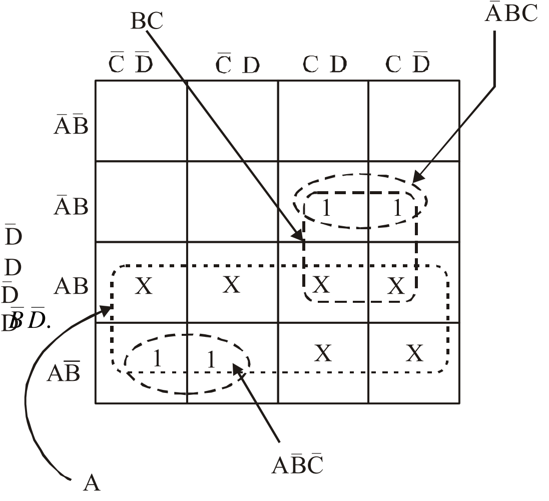
In digital systems design sometimes a situation arises in which some input variable conditions are not allowed. For example in a *BCD* (binary coded decimal) code, there are six invalid combinations : 1010, 1011, 1100, 1101, 1110 and 1111. Since these unallowed states will never occur in an application application involving the *BCD* code, they can be treated as "terms either 1 or 0 may be assigned to the output.

Now, we shall discuss as how the "don't care" terms can be used to advantage on the Karnaugh map for simplifying the Logic equations.

Consider for example, a combinational circuit which products a '1' output corresponding to a *BCD* input equal and greater than 6. The output is 0 corresponding to a *BCD* input less than 6. The truth table for this situation is as shown in Fig. 4-91 (*a*).

Inputs				*Outputs*	
A	*B*	*C*	*D*	*X*	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	1	→ $\bar{A}BC\bar{D}$
0	1	1	1	1	→ $\bar{A}BCD$
1	0	0	0	1	→ $A\bar{B}\bar{C}\bar{D}$
1	0	0	1	1	→ $A\bar{B}\bar{C}D$
1	0	1	0	x	
1	0	1	1	x	
1	1	0	0	x	
1	1	0	1	x	
1	1	1	0	x	
1	1	1	1	x	

(a)



(b)

Fig. 4-91.

We know that we can place 1*s* directly from the trugh table on the Karnaugh map. Similarly we can place *X* for the "don't care" enteries directly on the Karnaugh map as shown in Fig. 4-91 (*b*). When grouping the 1*s*, *Xs* can be treated as 1*s* make a larger grouping or as 0*s,* if they cannot be used to advantage Recall the larger the group of 1*s* the simpler the resulting term will be. Taking advantage of the "don't care" and using them as 1*s*, the resulting expression for the output is *A* + *BC.* However, if the "don't cares" are not used as 1*s*, the resulting expression is $A\bar{B}\bar{C}+\bar{A}BC$. Thus we can see the advantage of using "dont care" terms to get simplest logic expression.

Example 4-87. *Simplify using k-map*

(*i*) $F(W, X, Y, Z) = \Sigma m(0, 1, 2, 5, 8, 14) + d(4, 10, 13)$

(*ii*) $F(A, B, C, D, E) = \Sigma m(1, 2, 3, 8, 9, 10, 15, 22, 27) + d(0, 19)$

(*Nagpur University. 2008*)

Solution. (*i*) Given. The Boolean expression

$$F(W, X, Y, Z) = \Sigma m(0, 1, 2, 5, 8, 14) + d(4, 10, 13)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e. 0, 1, 2, 5, 8 and 14. We know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping as shown in Fig. 4-92.

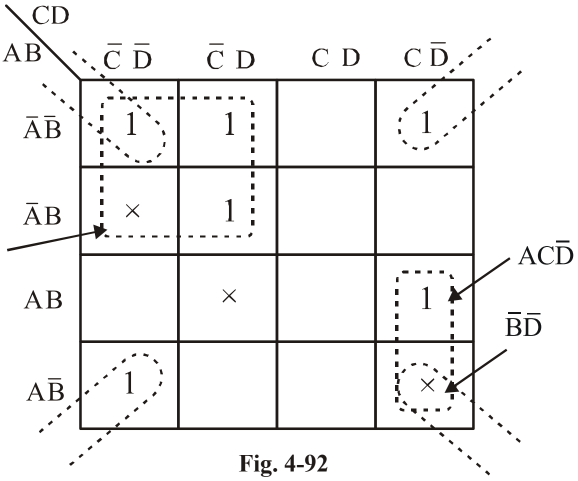

CD
AB
$\overline{C}\,\overline{D}$
$\overline{C}\,D$
$C\,D$
$C\,\overline{D}$
$\overline{A}\overline{B}$
$\overline{A}B$
AB
$A\overline{B}$
1
1
1
×
1
×
1
1
×
$\overline{A}\overline{C}$
$AC\overline{D}$
$\overline{B}\overline{D}$

Fig. 4-92

The wrap-around four square group that includes the 1s and Xs on four corners of the karnaugh map. This group produces a product term $\overline{B}\overline{D}$. This is determined by observing that the group contains both A and $\overline{A}$ and C and $\overline{C}$, so these variables are eliminated.

Another group of four containg 1s and Xs is formed near the top left corner of karnaugh map. This group produces a product term $\overline{A}\overline{C}$. This is determined by observing that the group contains both B and $\overline{B}$, D and $\overline{D}$ so these variable are eliminated.

Another group of two containing 1s and Xs is formed in fourth column. This group produces a product term $AC\overline{D}$. This is determined by observing that the group contains both C and $\overline{C}$ so these variables are eliminated.

The resulting simplified logic function is the sum of three product term.

$$f(W,X,Y,Z) = \overline{B}\overline{D}+\overline{A}\overline{C}+AC\overline{D} \textbf{ Ans.}$$

(*ii*) Given: The Boolean expression,

$$F(A, B, C, D, E) = \Sigma m(1,2,3,8,9,10,15,22,27) + d(0,19)$$

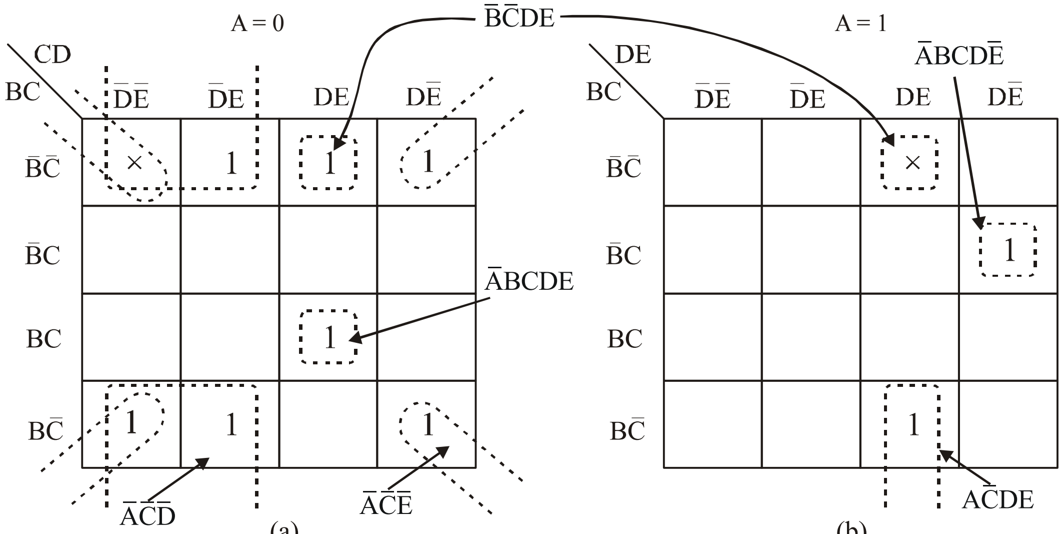
Solution. The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e 1, 2, 3, 8, 9, 10, 15, 22, and 27. The output is X corresponding to the term 0 and 19. We know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping.

The fire variable map is shown in Fig. 4-93. It consist of two four variable map with variables A,B,C,D and E Variable. A distinguishes between the two maps, fig 4-94 (a) represent the 16 square where A = 0 and other four variable map represents the squares where A = 1. Minitemrs 0 through 1s belong with A = 0 and minterms 16 through 31 with $A = 1$.

The wrap around four square group that includes the 1s on four corner of the karnaugh map. This group produce a term $\overline{A}\overline{C}\overline{E}$. This is determine by observing that the group contains both B and $\overline{B}$ and D and $\overline{D}$, so these variable are eliminated.

Another group of four squares in the first and fourth row produces the term $\overline{A}\overline{C}\overline{D}$. This is determine by observing that the group contains both B and $\overline{B}$ and E and $\overline{E}$, so these variables are eliminated. Similarly the another group in one square third column, third row produces the term $\overline{A}BCDE$.

Another group of one square in Fig. 4-93. (b) produces the term $A\overline{B}CD\overline{E}$. Similarly another group of two square in is map in third column produce the term $A\overline{C}DE$. This is determine by observing that the group contain B and $\overline{B}$, so these variable are eliminated.



BC \ DE	D̄Ē	D̄E	DE	DĒ
B̄C̄	×	1	1	1
B̄C				
BC			1	
BC̄	1	1		1

(a)

BC \ DE	D̄Ē	D̄E	DE	DĒ
B̄C̄			×	
B̄C				1
BC				
BC̄			1	

(b)

Fig. 4-93

One square in Fig. 4-93. (b) in first column and one square in Fig 4-93 (b) in first column overlop each other, so they make one group. It produce the term $\overline{B}\overline{C}DE$. This is determine by observing that the group contains both A and $\overline{A}$, so these variables are eliminated. Thus the resulting simplified expression,

$$F(A, B, C, D, E) = \overline{A}\overline{C}E + \overline{A}\overline{C}\overline{D} + \overline{A}BCDE + \overline{B}\overline{C}DE + A\overline{B}CD\overline{E} + A\overline{C}DE \quad \textbf{Ans.}$$

Example 4-88. *Get the minimized sum-of products expression for*

$$f(A, B, C, D) = \Sigma m\ (0, 1, 5, 6, 7, 8, 9) + d\Sigma m(10, 11, 12, 13, 14, 15)$$

Use karnaugh map to simplification (*VTU., Jan./Feb. 2005*)

Solution. The given Boolean function indicates that it output is 1 corresponding to the term indicated within the expression i.e, 0, 1, 5, 6, 7, 8, and 9. The output is X corresponding to the term 10, 11, 12, 13, 14, and 15. This is shown in Fig. 4-94. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-95.

Decimal Number	*Inputs*				*Outputs*
	A	*B*	*C*	*D*	*F*
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

Fig. 4-94

We know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping. The group of four squares containing 1s and Xs in the second column. This group produces a term $\overline{C}D$. The other group of four squares in the third and fourth column produce the term BC. Similarly the other group of four squares in first and second column produce the term $\overline{B}\,\overline{C}$. Thus the resulting simplified expression

$$f(A, B, C, D) = \overline{B}\,\overline{C} + \overline{C}D + BC \ \textbf{Ans.}$$

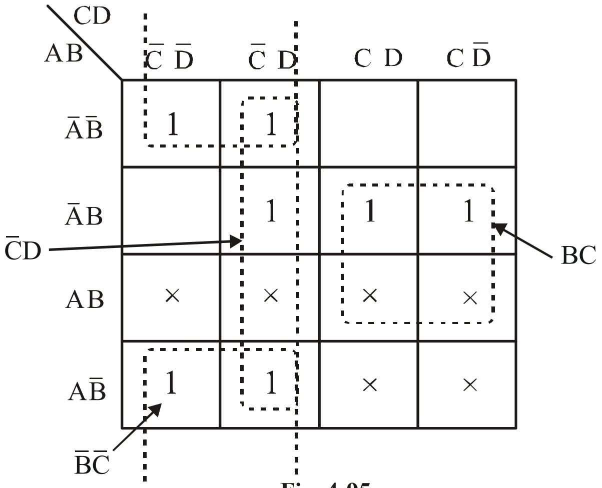


Fig. 4-95

Example 4-89. *Simplify the function using karnaugh map and implement using minimum number of logic gates.*

$F = \Sigma m\ (2, 9, 10, 12, 13) + D(15, 14)$ *(PTU., Dec. 2009)*

Solution. Given: The Boolean expression

$F = \Sigma m\ (2, 9, 10, 12, 13) + D(15, 14)$

Decimal Number	***Inputs*** *A*	*B*	*C*	*D*	***Outputs*** *F*	
0	0	0	0	0	0	
1	0	0	0	1	0	
2	0	0	1	0	1	→ $\overline{A}\overline{B}C\overline{D}$
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	0	
7	0	1	1	1	0	
8	1	0	0	0	0	
9	1	0	0	1	1	→ $A\overline{B}\overline{C}D$
10	1	0	1	0	1	→ $A\overline{B}C\overline{D}$
11	1	0	1	1	0	
12	1	1	0	0	1	→ $AB\overline{C}\overline{D}$
13	1	1	0	1	1	→ $AB\overline{C}D$
14	1	1	1	0	X	→ $ABC\overline{D}$
15	1	1	1	1	X	→ ABCD

Fig. 4.96

The given Boolean function indicates that its output is 1 corresponding to the term indicates within the expression *i.e.* 2, 9, 10, 12, and 13. The don't care condition is apply. We place X in the term 15 and 14. This shown in Fig. 4-96. We can map these values directly on the four-variable karnaugh map as shown in Fig. 4-97.

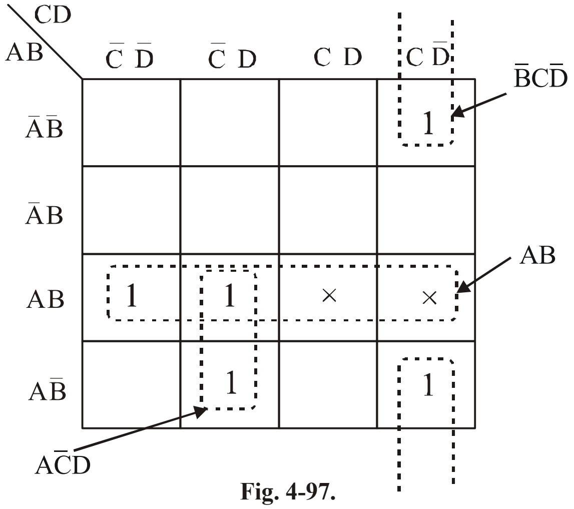


Fig. 4-97.

Taking advantage of the Xs and using then as 1s, the grouping of 1s and Xs and using is as shown in Fig. 4-98. The group of four square in the third row includes the 1's and Xs produce term, AB. This group contain both C and $\bar{C}$ and D and $\bar{D}$ so these variable are eliminated and resulting produce term AB. Similarly the two square in second and fourth colums produce the term $A\bar{C}D$ and $BC\bar{D}$ respectively. Thus the resulting simplified expression.

$$F(A, B, C, D) = AB + A\bar{C}D + BC\bar{D} \textbf{ Ans.}$$

This can be implemented using logic gate is shown in Fig. 4-98.

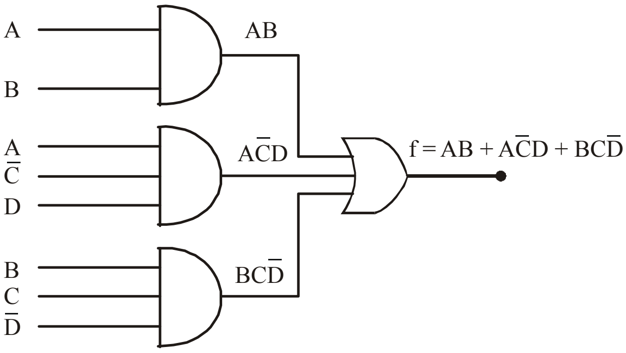


Fig. 4-98.

Example 4-90. *Simplify the following function using k-map and realize using NOR gates only.*

$F(A, B, C, D) = \pi M(1, 3, 5, 8, 14) + d(0, 9)$ (*Nagpur University, 2008*)

Solution. Given: The function

$$F(A, B, C, D) = \pi M(1, 3, 5, 8, 14) + d(0, 9)$$

The Boolean function indicates that its output is 0 corresponding to the term indicated within the expression i.e. 1, 3, 5, 8, and 14. In don't care condition the output is X corresponding to the term 0 and 9. This is shown in Fig. 4-99. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-100. We know that when grouping the 0s, Xs can be treated as 1s to make a larger grouping.

Decimal Number	*Inputs*				*Outputs*	
	A	*B*	*C*	*D*	*F*	
0	0	0	0	0	X	→ $\bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	0	→ $\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	1	
3	0	0	1	1	0	→ $\bar{A}\bar{B}CD$
4	0	1	0	0	1	
5	0	1	0	1	0	→ $\bar{A}B\bar{C}D$
6	0	1	1	0	1	
7	0	1	1	1	1	
8	1	0	0	0	0	→ $A\bar{B}\bar{C}\bar{D}$
9	1	0	0	1	X	
10	1	0	1	0	1	
11	1	0	1	1	1	
12	1	1	0	0	1	
13	1	1	0	1	1	
14	1	1	1	0	0	→ $ABC\bar{D}$
15	1	1	1	1	1	

Fig. 4-99.

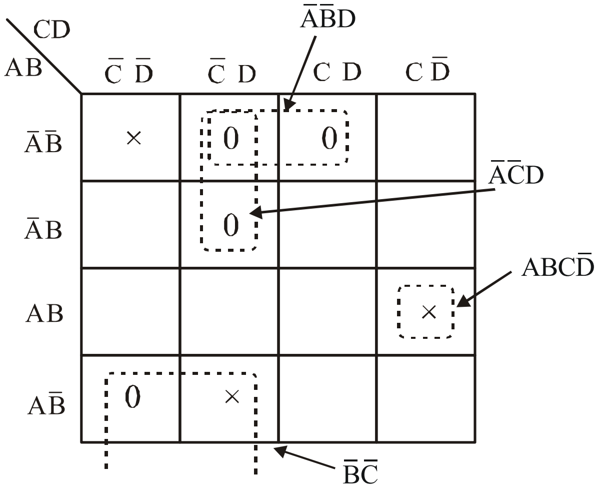


Fig. 4-100

The group of four squares in the first and last row produces a product term $\bar{B}\bar{C}$. This is determine by observing that the group contains both A and $\bar{A}$ and D and $\bar{D}$, so this variables is eliminated.

Another group of two squares in the first row produces term $\bar{A}\bar{B}C$. This is determined by observing that the group contains both D and $\bar{D}$, so these variable are eliminated. Similarly another group of two squares in second column produces term $\bar{A}\bar{C}D$. This is determined by observing that the group contains B and $\bar{B}$, so these variable are eliminated.

Another group of one square in the fourth column produce a term $ABC\overline{D}$. The logic function is the sum of four terms: $\overline{B}\,\overline{C}, \overline{A}\,\overline{B}C, \overline{A}\,\overline{C}D$ and $ABC\overline{D}$

$$F\,(A, B, C, D) = \overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}\,\overline{C}D + ABC\overline{D}$$

Applying DeMorgan's theorem

$$F\,(A, B, C, D) = \overline{(\overline{\overline{B}\,\overline{C}} + \overline{\overline{A}\,\overline{B}C} + \overline{\overline{A}\,\overline{C}D} + \overline{ABC\overline{D}})}$$

$$F\,(A, B, C, D) = (B + C)\,(A + B + \overline{C})(A + C + \overline{D})(\overline{A} + \overline{B} + \overline{C} + \overline{D})\ \textbf{Ans.}$$

This can be implemented using NOR gate as shown in Fig. 4-101.

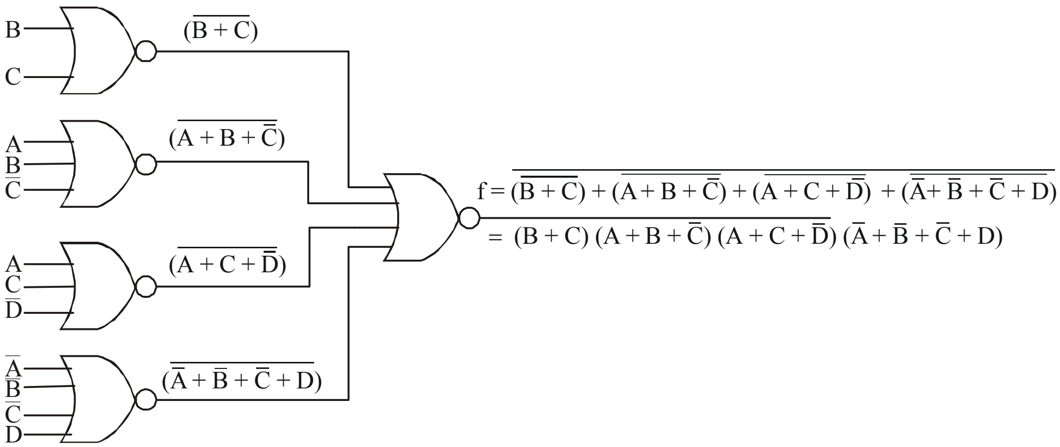

B
C
$\overline{(B + C)}$
A
B
$\overline{C}$
$\overline{(A + B + \overline{C})}$
A
C
$\overline{D}$
$\overline{(A + C + \overline{D})}$
$\overline{A}$
$\overline{B}$
$\overline{C}$
D
$\overline{(\overline{A} + \overline{B} + \overline{C} + D)}$
$f = \overline{\overline{(B + C)} + \overline{(A + B + \overline{C})}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

Fig. 4-101

Example 4-91. *Simplify the Boolean function in*

$$f(A,B,C) = \Sigma(2,3,6,7)$$

(*i*) *SOP*

(*ii*) *POS* (*GBTU/MTU, 2009-10*)

Solution.

(*i*) *SOP*

Given: the Boolean function

$$f(A,B,C) = \Sigma(2,3,6,7)$$

The given function indicates that output is 1 corresponding to the term 2, 3,6, and 7 as shown in Fig. 4-102. We can map these values directly on to the three-variable karnaugh map as shown in Fig. 4-102. In order to simplify the SOP Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-103.

Decimal Number	*Inputs*			*Outputs*
	A	*B*	*C*	*F*
0	0	0	0	0
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	1
7	1	1	1	1

Fig. 4-102

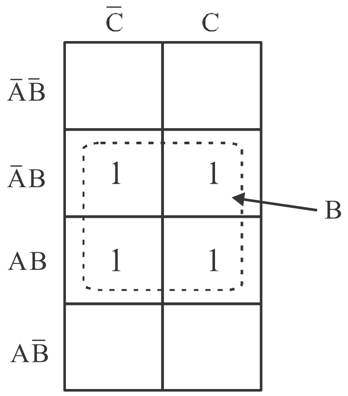


Fig. 4-103

The group of four squares in the middle of the karnaugh map. This group produces a term B. This is determined by observing that this group contains both A and $\overline{A}$ and C and $\overline{C}$ so these variables are eliminated. Thus the resulting simplified expression.

$f(A, B, C) = B$ **Ans.**

(*ii*) POS

Given: the Boolean function

$f(A, B, C) = \Sigma\,(2, 3, 6, 7)$

The given function indicates that output is 0 corresponding to the term 0, 1, 4 and 5 as shown in Fig. 4-104. We can map these values directly on to the three-variable karnaugh map as shown in Fig. 4-105. In order to simplify the POS Boolean expression represented on the karnaugh map, group the 0s as shown in the Fig. 4-105.

Decimal Number	***Inputs*** *A*	*B*	*C*	***Outputs*** *F*
0	0	0	0	0
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	1
7	1	1	1	1

Fig. 4-104.

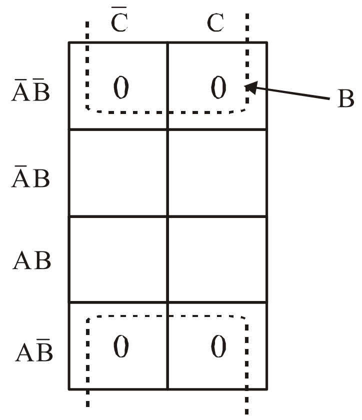


Fig. 4-105

The group of four squares with " wrap-around" adjacency is formed form the top and the bottom rows of the karnaugh map. This group produces a term B. This is determined by observing that this group contains both A and $\overline{A}$ and C and $\overline{C}$ so these variables are eliminated. Thus the resulting simplified expression.

$f(A,B,C) = B$ **Ans.**

Example 4-92. *Simplify using k-map*

(i) $f(A, B, C, D) = \Sigma m\,(0, 1, 4, 5, 9, 11, 14, 15) + d(10, 13)$

(ii) $f(A, B, C, D) = \Pi M\,(4, 6, 10, 12, 13, 15)$ *(Nagpur University, 2008)*

Solution.

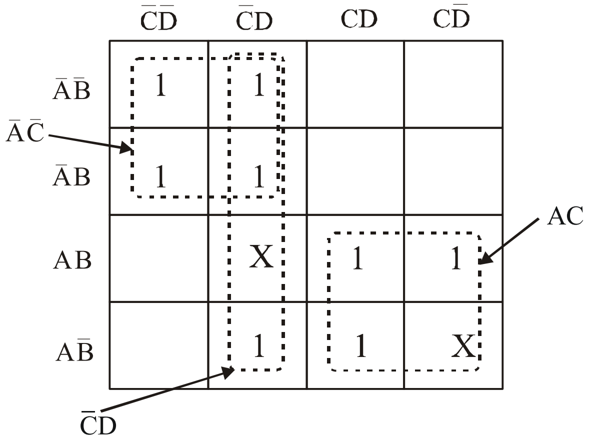
(*i*) Given: The function

$$f(A, B, C, D) = \Sigma m\,(0, 1, 4, 5, 9, 11, 14, 15) + d(10, 13)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 1, 4, 5, 9, 11, 14, and 15. In don't care condition the output is X corresponding to the term 10 and 13. This is shown in Fig. 4-106. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-107. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s and Xs as shown in the Fig. 4-107.

Decimal Number	***Inputs***				***Outputs***	
	A	***B***	***C***	***D***	***F***	
0	0	0	0	0	1	→ $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
1	0	0	0	1	1	→ $\overline{A}\,\overline{B}\,\overline{C}D$
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	1	→ $\overline{A}B\overline{C}\,\overline{D}$
5	0	1	0	1	1	→ $\overline{A}B\overline{C}D$
6	0	1	1	0	0	
7	0	1	1	1	0	
8	1	0	0	0	0	
9	1	0	0	1	1	→ $A\overline{B}\,\overline{C}D$
10	1	0	1	0	X	→ $A\overline{B}C\overline{D}$
11	1	0	1	1	1	→ $A\overline{B}CD$
12	1	1	0	0	0	
13	1	1	0	1	X	→ $AB\overline{C}D$
14	1	1	1	0	1	→ $ABC\overline{D}$
15	1	1	1	1	1	→ $ABCD$

Fig. 4-106



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	1		
$\overline{A}B$	1	1		
AB		X	1	1
$A\overline{B}$		1	1	X

Fig. 4-107

We know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping. The group of four squres in the top left corner of karnaugh map produces a product term $\overline{A}\overline{C}$. This is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$ so these variable are eliminated.

Another group of four squares in the second column produces term $\overline{C}D$. This is determined by observing. that the group contains both A and $\overline{A}$ and B and $\overline{B}$ so these variables are eliminated. Similarly another group of four squares in the bottom right corner produces term. This is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$ so these variable are eliminated. Thus the resulting simplified expression.

$$f(A, B, C, D) = \overline{A}\overline{C} + \overline{C}D + AC$$

(*ii*) Given: The function

$$f(A, B, C, D) = \Pi M\,(4, 6, 10, 12, 13, 15)$$

The given Boolean function indicates that its output is 0 corresponding to the term indicated within the expression i.e., 4, 6, 10, 12, 13, and 15. This is shown in Fig. 4-108. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-109. In order to simplify the Boolean expression represented on the karnaugh map, group the 0s as shown in the Fig. 4-109.

Decimal Number	***Inputs***				***Outputs***
	A	***B***	***C***	***D***	***F***
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Fig. 4-108

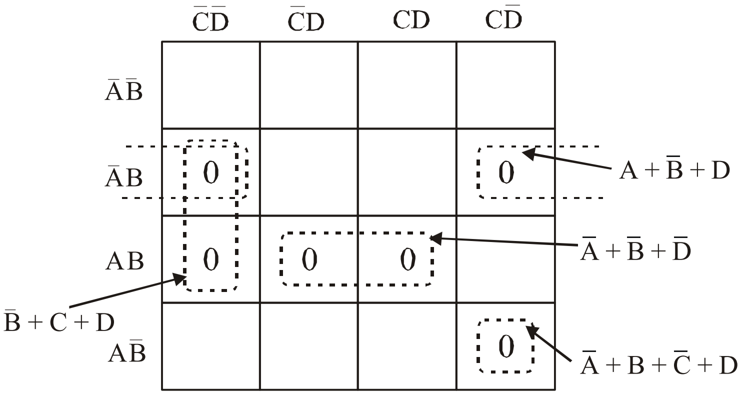

$\overline{C}\overline{D}$
$\overline{C}D$
CD
$C\overline{D}$
$\overline{A}\overline{B}$
$\overline{A}B$
AB
$A\overline{B}$
0
0
0
0
0
0
$A + \overline{B} + D$
$\overline{A} + \overline{B} + \overline{D}$
$\overline{B} + C + D$
$\overline{A} + B + \overline{C} + D$

Fig. 4-109

The group of two squares in the first column of karnaugh map produces a product term $B\bar{C}\bar{D}$. This is determined by observing that the group contains both A and $\bar{A}$, so this variable is eliminated.

Another group of two squares in the second row produces term $\bar{A}B\bar{D}$. This is determined by observing that the group contains both C and $\bar{C}$, so this variable is eliminated. Similarly another group of two squares in the third row karnaugh map produces term BD. This is determined by observing that the group contains both C and $\bar{C}$ so this variable is eliminated.

Another group of one square in the fourth column produces term $A\bar{B}C\bar{D}$. Thus the resulting simplified expression.

$$f(A, B, C, D) = (\bar{B}+C+D)(A+\bar{B}+D)(\bar{A}+\bar{B}+\bar{D})(\bar{A}+B+\bar{C}+D) \textbf{ Ans.}$$

Example 4-93. *Using k-map method reduce the following function and implements the simplified function using NOR gates only*

$$f(A, B, C, D) = \Sigma m\ (0, 1, 5, 6, 8, 10, 12, 15) + d\ (3, 7, 9)$$

(*Mahatma Gandhi University, Jan 2007*)

Solution. Given: The function

$$f(A, B, C, D) = \Sigma m\ (0, 1, 5, 6, 8, 10, 12, 15) + d\ (3, 7, 9)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 1, 5, 6, 8, 10, 12, and 15. In don't care condition the output is X corresponding to the term 3, 7, and 9. This is shown in Fig. 4-110. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-111. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s and Xs as shown in the Fig.4-111.

Decimal Number	***Inputs***				***Outputs***	
	A	***B***	***C***	***D***	***F***	
0	0	0	0	0	1	$\longrightarrow \bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	1	$\longrightarrow \bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	0	
3	0	0	1	1	X	$\longrightarrow \bar{A}\bar{B}CD$
4	0	1	0	0	0	
5	0	1	0	1	1	$\longrightarrow \bar{A}B\bar{C}D$
6	0	1	1	0	1	$\longrightarrow \bar{A}BC\bar{D}$
7	0	1	1	1	X	$\longrightarrow \bar{A}BCD$
8	1	0	0	0	1	$\longrightarrow A\bar{B}\bar{C}\bar{D}$
9	1	0	0	1	X	$\longrightarrow A\bar{B}\bar{C}D$
10	1	0	1	0	1	$\longrightarrow A\bar{B}C\bar{D}$
11	1	0	1	1	0	
12	1	1	0	0	1	$\longrightarrow AB\bar{C}\bar{D}$
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	1	$\longrightarrow ABCD$

Fig. 4-110

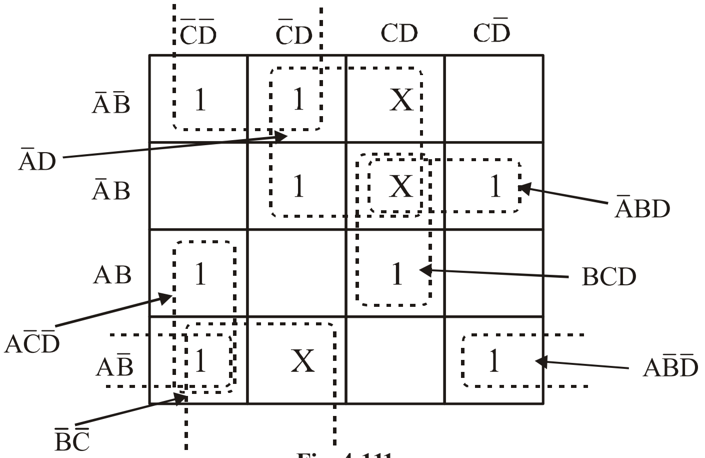

$\overline{C}\overline{D}$
$\overline{C}D$
CD
$C\overline{D}$
$\overline{A}\overline{B}$
$\overline{A}B$
AB
$A\overline{B}$
1
1
X
1
X
1
1
1
1
X
1
$\overline{A}D$
$\overline{A}BD$
BCD
$A\overline{C}\overline{D}$
$A\overline{B}\overline{D}$
$\overline{B}\overline{C}$

Fig. 4-111

We know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping. The group of two squares in the first column of karnaugh map produces a product term $A\overline{C}\overline{D}$. This is determined by observing that the group contains both B and $\overline{B}$,so this variable is eliminated.

The group of two squares in the third column of Karnaugh map produces a product term BCD. This is determined by observing that the group contains both A and $\overline{A}$, so this variable is eliminated.

The group of two squares in the second row of Karnaugh map produces a product term $\overline{A}BC$ This is determined by observing that the group contains both D and $\overline{D}$, so this variable is eliminated.

The group of two squares in the fourth row of Karnaugh map produces a product term $A\overline{B}\overline{D}$. This is determined by observing that the group contains both D and $\overline{D}$, so this variable is eliminated.

Another group of four squares in the top of Karmaugh map produces term $\overline{A}D$ This is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$ so these variables are eliminated. Similarly another group of four squares produces term $\overline{B}\overline{C}$. This is determined by observing that the group contains both A and $\overline{A}$ and D and $\overline{D}$ so these variables are eliminated. Thus the resulting simplified expression,

$$f(A, B, C, D) = A\overline{C}\overline{D} + BCD + \overline{A}BC + A\overline{B}\overline{D} + \overline{A}D + \overline{B}\overline{C}.\ \textbf{Ans.}$$

Example 4-94. *Minimise that $Y = A + \overline{A}B + AC$ using k map*

(*Mahatma Gandhi University, Nov 2005*)

Solution. Given: The Boolean expression

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

We know that the domain of the given expression is A, B and C, from the expression we find that all the three product terms is in the non-standard form. To map the value let us first convert these product terms to a standard form taking one term at a time.

The first term A has a missing variable B or $\overline{B}$ and C or $\overline{C}$. So multiply the first term by (B+ $\overline{B}$) and (C + $\overline{C}$) as follows,

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

The second term $\overline{A}$ B has a missing variable C or $\overline{\text{C}}$. So multiply the second term by (C + $\overline{\text{C}}$) as follows,

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

The third term AC has a missing variable B or $\overline{\text{B}}$. So multiply the third term by (B + $\overline{\text{B}}$) as follows,

$$AC = AC(B+\overline{B}) = ABC + A\overline{B}C$$

The standard SOP,

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

The three variable standard SOP function indicates that its output is 1 corresponding to the term indicated within the expression i.e., ABC, $AB\overline{C}$, $A\overline{B}C$, $A\overline{B}\overline{C}$, $\overline{A}BC$ and $\overline{A}B\overline{C}$ .This is shown in Fig.4-112. We can map these values directly on to the three-variable karnaugh map as shown in Fig. 4-113. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-113.

Decimal Number	***Inputs***			***Outputs***	
	A	***B***	***C***		
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	1	→ $\overline{A}B\overline{C}$
3	0	1	1	1	→ ABC
4	1	0	0	1	→ $A\overline{B}\overline{C}$
5	1	0	1	1	→ $A\overline{B}C$
6	1	1	0	1	→ $AB\overline{C}$
7	1	1	1	1	→ ABC

Fig. 4-112

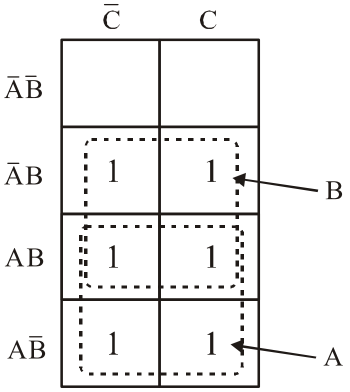


Fig. 4-113

A group of four squares in the second and third row column produces term. This is determined by observing that the group contains both A and $\overline{A}$ and C and $\overline{C}$ so these variables are eliminated. Similarly another group of four squares in the third and fourth row produces term A. This is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$ so these variables are eliminated. Thus the resulting simplified expression,

$$Y = A + B \textbf{ Ans.}$$

Example 4-95. *Using k-map, simplify f = Σm (0, 1, 5, 7, 10, 14) +d (2, 4).*

(*Mahatma Gandhi University, Nov 2005*)

Solution. Given: The function

$$f = \Sigma m\,(0, 1, 5, 7, 10, 14) + \mathrm{d}\,(2, 4).$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 1, 5, 7, 10, and 14. The output is X corresponding to the term 2 and 4. This is shown in Fig.4-114. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-113. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-115.

Decimal Number	***Inputs***				***Outputs***	
	A	***B***	***C***	***D***	***F***	
0	0	0	0	0	1	→ $\overline{A}\overline{B}\overline{C}\overline{D}$
1	0	0	0	1	1	→ $\overline{A}\overline{B}\overline{C}D$
2	0	0	1	0	X	→ $\overline{A}\overline{B}C\overline{D}$
3	0	0	1	1	0	
4	0	1	0	0	X	→ $\overline{A}B\overline{C}\overline{D}$
5	0	1	0	1	1	→ $\overline{A}B\overline{C}D$
6	0	1	1	0	0	
7	0	1	1	1	1	→ $\overline{A}BCD$
8	1	0	0	0	0	
9	1	0	0	1	0	
10	1	0	1	0	1	→ $\overline{A}B\overline{C}D$
11	1	0	1	1	0	
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	1	→ $\overline{A}\overline{B}\overline{C}D$
15	1	1	1	1	0	

Fig. 4-114

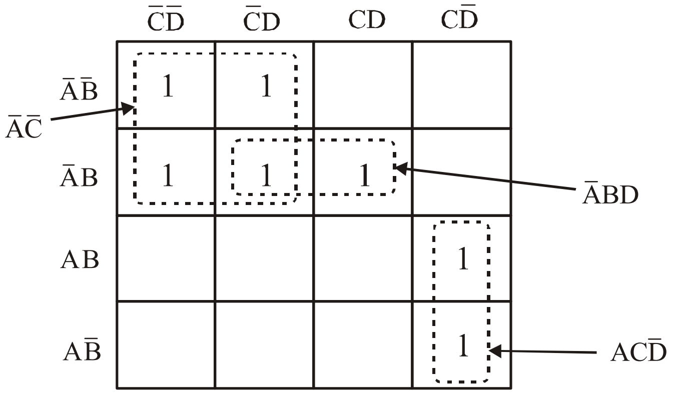


Fig. 4-115

We know that when grouping the 1s,Xs can be treated as 1s to make a larger grouping. The group of four square in the top left corner of karnaugh map produces a product term $\overline{A}\overline{C}$ This is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$ so these variables are eliminated.

Another group of two square in the second row produces term $\overline{A}BD$. This is determined by observing that the group contains both C and $\overline{C}$, so this variable is eliminated. Similarly another group of two squares in the fourth column produces term $AC\overline{D}$. This is determined by observing that the group contains both B and $\overline{B}$ so this variable is eliminated. Thus the resulting simplified expression.

$$f(A,\ B,\ C,\ D) = \overline{A}\overline{C} + \overline{A}BD + AC\overline{D}$$

Example 4-96. *Show that A ⊙ B ⊙ C ⊙ D = Σm (0, 3, 5, 6, 9, 10, 12, 15)*

(*VTU, Jan/Feb 2006*)

Solution. Given: The expression A ⊙ B ⊙ C ⊙ D = Σm(0, 3, 5, 6, 9, 10, 12, 15)

We know that ($A \odot B = AB + \overline{AB}$), We get

$$= (AB + \overline{AB}) \odot (CD + \overline{CD})$$

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

$$= ABCD + \overline{AB}CD + AB\overline{CD} + \overline{ABCD} + A\overline{BC}D + A\overline{B}C\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D}$$

$$= \Sigma m (0, 3, 5, 6, 9, 10, 12, 15)$$

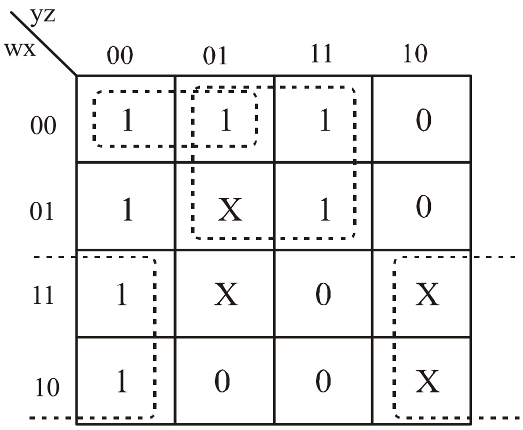
Example 4-97. *Using karnaugh maps, determine the minimal sums and minimal products for*

$$f(w, x, y, z) = \Sigma m\ (0,\ 1,\ 3,\ 7,\ 8,\ 12) + d_c\ (5,\ 10,\ 13,\ 14)$$

Is your answer unique? (*VTU, Jan/Feb 2006*)

Solution. The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 1, 3, 7, 8, and 12. The output is X corresponding to the term indicated within the expression i.e., 5, 10, 13, and 14. The output is 0 corresponding to the remaining terms.

To find the Minimal Sum of product we know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. Recall, the larger the group of 1s, the simpler the resulting term will be. Taking advantage of the Xs and using them as 1s, the grouping of 1s and Xs and using is as shown in Fig. 4-116.



wx \ yz	00	01	11	10
00	1	1	1	0
01	1	X	1	0
11	1	X	0	X
10	1	0	0	X

Fig. 4-116

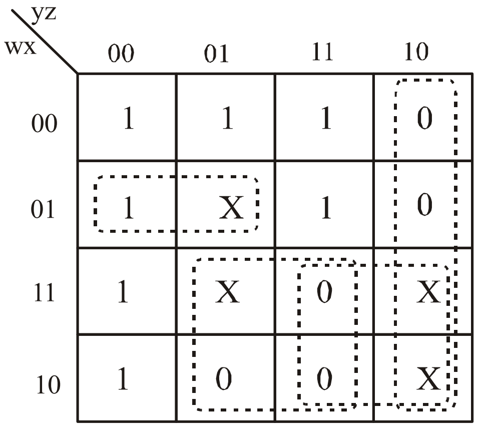
A group of four square containing 1s and Xs around the second and third column of karnaugh map is formed. This group produces a product term $\overline{w}z$. This is determined by observing that the group contains both x and $\overline{x}$ and y and $\overline{y}$ so these variables are eliminated.

Another group of four containing 1s and Xs is formed near the first and fourth column of karnaugh map. This group produces a product term $w\overline{z}$. This is determined by observing that the group contains both x and $\overline{x}$ and y and $\overline{y}$ so these variable are eliminated. Similarly the group of two containing 1s and Xs in the first row of karnaugh map. This group produces a product term $\overline{wxy}$. This is determined by observing that the group contains both z and $\overline{z}$ so these variables are eliminated.

The resulting simplified logic function is the sum of the three product terms, the Minimal Sum of Product (SOP),

$$f(w, x, y, z) = \overline{wxy} + w\overline{z} + \overline{w}z \textbf{ Ans.}$$

For the Minimal Product of Sum (POS) we know that when grouping the 0s, XS, can be treated as 0s to make a larger grouping. The group of four containing 0s and Xs in the third column of karnaugh map. This group produces a product term $y\overline{z}$. This is determined by observing that the group contains x and $\overline{x}$ and w and $\overline{w}$ so these variable are eliminated. Similarly the group of two containing 0s and Xs in the second row of karnaugh map. This group produces a product term $\overline{w}x\overline{y}$. This is determined by observing that the group contains both z and $\overline{z}$ so these variables are eliminated as shown in Fig. 4-117.



wx \ yz	00	01	11	10
00	1	1	1	0
01	1	X	1	0
11	1	X	0	X
10	1	0	0	X

Fig. 4-117

Another group of four containing 0s and Xs in the second and third column of Kamaugh map. This group produces a product term wz. This is determined by observing that the group contians both x and x and y and y so these variables are eliminated. Similarly the group of four containing 0s and Xs in the second and third column of Karnaugh map. This group produces a product term wy. This is determined by observing that the group contians both z and $\overline{z}$ and x and $\overline{x}$ so these variables are eliminated.

POS form

$$f = (\bar{y} + z)(\bar{w} + \bar{y})(\bar{w} + \bar{z})(w + \bar{x} + y) \text{ Ans.}$$

The answer is not unique; SOP is not same as POS solution.

Example 4-98. *Using karnaugh maps, determine the minimal sums and minimal products for*

$$f(A, B, C, D) = \Sigma m\,(0, 3, 5, 6, 9, 10, 12, 15)$$

equal to $A \odot B \odot C \odot D$ using k-map. (*Nagpur University, 2007*)

Solution. The given Boolean function indicates that its output is 1corresponding to the term indicated within the expression i.e., 0, 3, 5, 6, 9, 10, 12, and 15. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-118. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-118.

The group of two squares in the first column produces a product term. This determined by observing that the group contains both A and so this variable is eliminated. Another group of two squares in the second column produces term The variable A is eliminated because the group contains both A and

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1		1	
$\bar{A}B$		1		1
AB	1		1	
$A\bar{B}$		1		1

Fig. 4-118

A group of four squares containing 1s and Xs around the second and third column of karnaugh map is formed. This group produces a product term $\bar{w}z$. This is determined by observing that the group contains both x and $\bar{x}$ and y and $\bar{y}$ so these variables are eliminated.

Another group of four containing 1s and Xs is formed near the first and fourth column of karnaugh map. This group produces a product term $w\bar{z}$. This is determined by observing that the group contains both x and $\bar{x}$ and y and $\bar{y}$ so these variables are eliminated. Similarly the group of two containing 1s and Xs in the first row of karnaugh map. This group produces a product term $\overline{wxy}$. This is determined by observing that the group contains both z and $\bar{z}$ so these variables are eliminated.

The resulting simplified logic function is the sum of the three product terms, the Minimal Sum of Product (SOP),

$$f(w, x, y, z) = \overline{wxy} + w\bar{z} + \bar{w}z \text{ Ans.}$$

Example 4-99. *Minimize the given Boolean. Using K-map*

F(A, B, C, D) = Σm (3, 4, 5, 7, 9, 13, 15) (*GBTU/MTU, 2006*)

Solution. Given: The function

$$F(A, B, C, D) = \Sigma m\,(3, 4, 5, 7, 9, 13, 15)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 3, 4, 5, 7, 9, 13, and 15. This is shown in Fig. 4-119. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-120. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-120.

Decimal Number	***Inputs***				***Outputs***
	A	***B***	***C***	***D***	***F***
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Fig. 4-119.

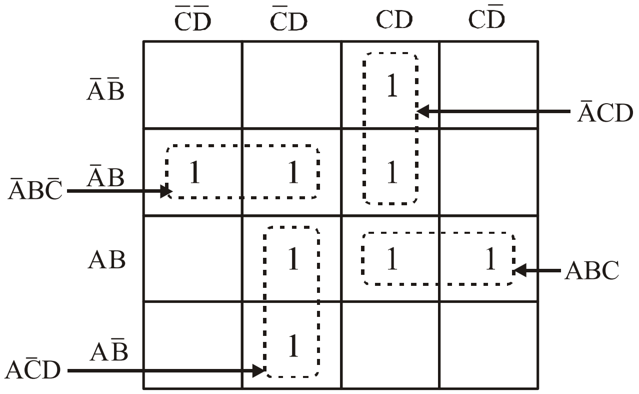


Fig. 4-120.

The group of two squares in the second column of karnaugh map produces a product term $A\overline{C}D$. This is determined by observing that the group contains both B and $\overline{B}$, so this variable is eliminated. Another group of two squares in the third column of karnaugh map produces a product term $A\overline{C}D$. This is determined by observing that the group contains both *B* and $\overline{B}$, so this variable is eliminated.

Another group of two squares in the second row of karnaugh map produces a product term $\overline{A}B\overline{C}$. This is determined by observing that the group contains both *D* and $\overline{D}$, so this variable is eliminated. Similarly another group of two squares in the third row of karnaugh map produces a product term *A B C*. This is determined by observing that the group contains both *D* and $\overline{D}$, so this variable is eliminated. Thus the resulting simplified expression,

$$F(A,\ B,\ C,\ D) = A\overline{C}D + \overline{A}CD + \overline{A}B\overline{C} + ABC \textbf{ Ans.}$$

Example 4-100. *Minimize using K-map $F(A,B,C,D) = \sum m\,(1,3,5,7,9,11,13,15)$*

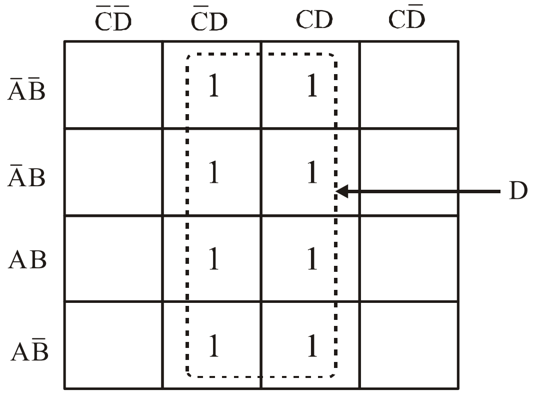
Solution. Given: The function

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

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 1, 3, 5, 7, 9, 11, 13, and 15. This is shown in Fig.4-121. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-122. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-122.

Decimal	*Inputs*				*Outputs*
Number	*A*	*B*	*C*	*D*	*F*
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Fig. 4-121.



	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$		1	1	
$\overline{A}B$		1	1	
AB		1	1	
$A\overline{B}$		1	1	

Fig. 4-122.

The group of four squares in the first and second column of karnaugh map produces a product term *D* This is determined by observing that the group contains both *A* and $\overline{A}$ and *B* and $\overline{B}$ and *C* and $\overline{C}$, so these variable are eliminated. Thus the resulting simplified expression,

F(A, B, C, D) = D **Ans.**

Example 4-101. *Simplify the function expression using k-map and implement the output using fundamental gates.*

$$F(A, B, C, D) = \Sigma m(1, 3, 4, 6, 8, 9, 11, 13, 15) + d\,(0, 2, 14) \quad \textit{(GBTU/MTU, 2007)}$$

Solution. Given: The function

$$F(A, B, C, D) = \Sigma m\,(1, 3, 4, 6, 8, 9, 11, 13, 15) + d\,(0, 2, 14)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 1, 3, 4, 6, 8, 9, 11, 13, and 15. In don't care condition the output is X corresponding to the term 0, 2, and 14. This is shown in Fig. 4-123. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-124. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s and Xs as shown in the Fig. 4-124.

Decimal Number	***Inputs***				***Outputs***
	A	***B***	***C***	***D***	***F***
0	0	0	0	0	X
1	0	0	0	1	1
2	0	0	1	0	X
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	X
15	1	1	1	1	1

Fig. 4-123.

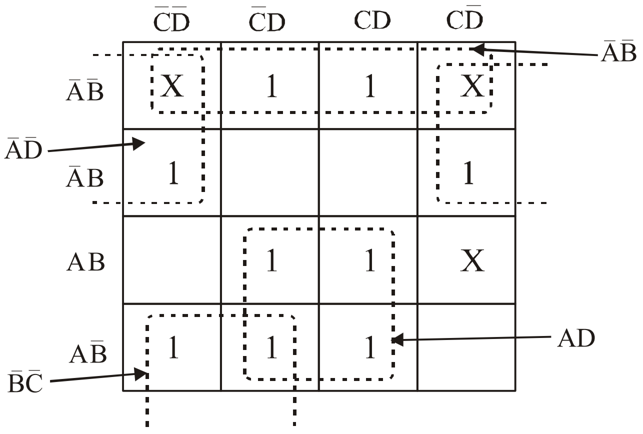


Fig. 4-124.

We know that when grouping the 1s, Xs can be treated as 1s to make a larger grouping. The group of four squares in the first row of karnaugh map produces a product term $\overline{A}\overline{B}$. This is determined by observing that the group contains both C and $\overline{C}$ and D and $\overline{D}$, so these variables are eliminated.

Another group of four with "wrap-around" adjacency is formed form the top and the bottom rows of the karnaugh map. This group overlaps with the previous group and produces a product term $\overline{B}\overline{C}$. This is determined by observing that this group contains both A and $\overline{A}$ and D and $\overline{D}$ so these variables are eliminated.

Similarly another group of four with "wrap-around" adjacency is formed from the left and the right column of the karnaugh map. This group overlaps with the previous group and produces a product term $\overline{A}\overline{D}$. This is determined by observing that this group contains both B and $\overline{B}$ and C and $\overline{C}$ so these variables are eliminated.

Another group of four squares in the lower of karnaugh map produces a product term AD this is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$, so these variables are eliminated. Thus the resulting simplified expression.

$$F(A, B, C, D) = \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{D} + AD \text{ Ans.}$$

Example 4-102. *Simplify the following expression using k-map*

$$F(A, B, C, D) = \Pi M\ (0, 1, 3, 6, 7, 8, 9, 11, 13, 14, 15) \qquad \textit{(GBTU/MTU, 2007)}$$

Solution. Given: The function

$$F(A, B, C, D) = \Pi M\ (0, 1, 3, 6, 7, 8, 9, 11, 13, 14, 15)$$

The given Boolean function indicates that its output is 0 corresponding to the term indicated within the expression i.e., 0, 1, 3, 6, 7, 8, 9, 11, 13, 14, and 15. This is shown in Fig. 4-125. We can map these values directly on to the four-variable karnaugh map as shown in Fig. 4-126. In order to simplify the Boolean expression represented on the karnaugh map, group the 0s as shown in the Fig. 4-126.

Decimal Number	***Inputs***				***Outputs***
	A	***B***	***C***	***D***	***F***
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Fig. 4-125.

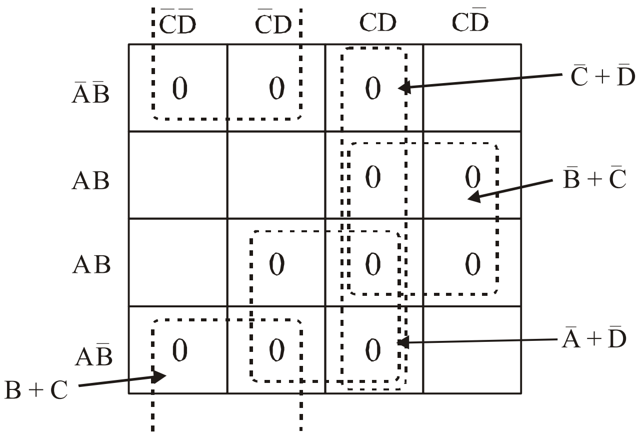

$\overline{C}\overline{D}$
$\overline{C}D$
CD
$C\overline{D}$
$\overline{A}\overline{B}$
AB
AB
$A\overline{B}$
0
0
0
0
0
0
0
0
0
0
0
$\overline{C}+\overline{D}$
$\overline{B}+\overline{C}$
$\overline{A}+\overline{D}$
$B+C$

Fig. 4-126.

The group of four squares in the third column of karnaugh map produces a term $\overline{C}+\overline{D}$. This is determined by observing that the group contains both *A* and $\overline{A}$ and *B* and $\overline{B}$, so these variables are eliminated.

Another group of four squares in the third and fourth column of karnaugh map produces a term $\overline{B}+\overline{C}$ This is determined by observing that the group contains both *A* and $\overline{A}$ and *D* and $\overline{D}$, so these variables are eliminated.

Similarly another group of four squares in the second and third column of karnaugh map produces a term $\overline{A}+\overline{D}$ This is determined by observing that the group contains both *B* and $\overline{B}$ and *C* and $\overline{C}$, so these variables are eliminated.

Another group of four squares with "wrap-around" adjacency is formed form the top and the bottom rows of the karnaugh map. This group overlaps with the previous group and produces a term B + C. This is determined by observing that this group contains both *A* and $\overline{A}$ and *D* and $\overline{D}$ so these variables are eliminated. Thus the resulting simplified expression,

$$F(A, B, C, D) = (\overline{C}+\overline{D})(\overline{B}+\overline{C})(\overline{A}+\overline{D})(B + C) \textbf{ Ans.}$$

Example 4-103. *Simplify the following expression and implement it with two levels NAND gated circuit*

$$Y = BC + BC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} \qquad (GBTU/MTU, 2009\text{-}10)$$

Solution. Given: The Boolean expression

$$Y = BD + BC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D}$$

We know that the domain of the given expression is A, B, C, and D. From the expression we find that the first and second product terms are in the non-standard form. To simplify this expression with karnaugh map we have to convert it into standard form. Let us convert these product terms to a standard form taking one term at a time.

The first term BD has a missing variable *A* or $\overline{A}$ and *C* or $\overline{C}$. So multiply the first term by $(A+\overline{A})$ and $(C+\overline{C})$ as follow,

$$BD = BD\,(A+\overline{A})\,(C+\overline{C}) = (ABD + \overline{A}\,BD)\,(C+\overline{C})$$
$$= ABCD + AB\overline{C}\,D + \overline{A}\,BCD + \overline{A}\,BC\overline{D}$$

The second term $BC\overline{D}$ has a missing variable A or $\overline{A}$. So multiply the second term by $(A + \overline{A})$ as follow,

$$BC\overline{D}\,(A + \overline{A}) = ABC\overline{D} + \overline{A}\,BC\overline{D}$$

Thus the standard SOP (i.e., sum-of-product) form of the given expression,

$$Y = ABCD + AB\overline{C}\,D + \overline{A}\,BCD + \overline{A}\,B\overline{C}\,D + ABC\overline{D} + \overline{A}\,BC\overline{D} + A\overline{B}\;\overline{C}\;\overline{D}$$

We can map these SOP values directly on to the four-variable karnaugh map as shown in Fig. 4-127. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s as shown in the Fig. 4-127.

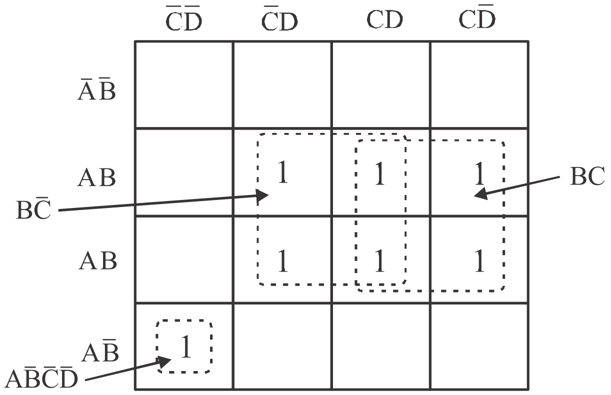


Fig. 4-127

The group of four squares in the second and third column of karnaugh map produces a term $B\overline{C}$. This is determined by observing that the group contains both A and $\overline{A}$ and D and $\overline{D}$, so these variable are eliminated. Another group of four squares in the third and fourth column of karnaugh map produces a term BC. This is determined by observing that the group contains both A and $\overline{A}$ and D and $\overline{D}$, so these variables are eliminated.

Another group of one squares in the first column of karnaugh map produces a term $A\overline{B}\overline{C}\overline{D}$. Thus the resulting simplified expression,

$$F(A, B, C, D) = B\overline{C} + BC + A\overline{B}\overline{C}\overline{D} \quad \textbf{Ans.}$$

Example 4-104. *Simplify the Boolean function with don't care condition using karnaugh map*

F(A, B, C, D) = Σ(1, 3, 4, 13, 15) +d(2, 5, 6, 7) *(GBTU/MTU, 2009-10)*

Solution. Given: The function

F(A, B, C, D) = Σ (1, 3, 4, 13, 15) +d(2, 5, 6, 7)

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 1, 3, 4, 13, and 15. In don't care condition the output is X corresponding to the term 2, 5, 6, and 7. This is shown in Fig. 4-128. We can map these values directly on to the four-variable karnaugh map as shown in fig. 4-129. In order to simplify the Boolean expression represented on the karnaugh map, group the 1s and Xs as shown in the Fig. 4-129.

Decimal Number	*Inputs* *A*	*B*	*C*	*D*	*Outputs F*
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	X
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	X
6	0	1	1	0	X
7	0	1	1	1	X
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Fig. 4-128

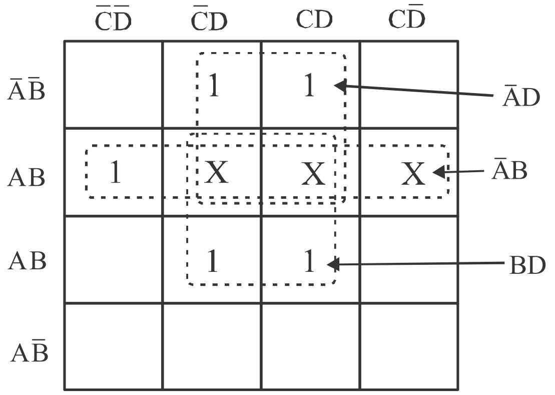


Fig. 4-129

The group of four squares in the second row of karnaugh map produces a term $\overline{A}\ B$. This is determined by observing that the group contains both *C* and $\overline{C}$ and *D* and $\overline{D}$, so these variables are eliminated.

Another group of four squares in the first and second row of karnaugh map produces a term $\overline{A}\ D$. This is determined by observing that the group contains both *B* and $\overline{B}$ and *C* and $\overline{C}$, so these variables are eliminated.

Similarly another group of four squares in the second and third row of karnaugh map produces a term *BD*. This is detemined by abserving that the group contains both *B* and $\overline{B}$ and *D* and $\overline{D}$, so these variables are eliminated. Thus the resulting simplified expression.

$F(A, B, C, D) = \overline{A}\ B + \overline{A}\ D + BD$ **Ans.**

Example 4-105. *Simplify the following Boolean function using K-map F(A, B, C, D) =* Σ *(1, 3, 7, 11, 15) with don't care conditions*

d (A, B, C, D) = Σ *(0, 2, 5)* *(GTU., May 2011)*

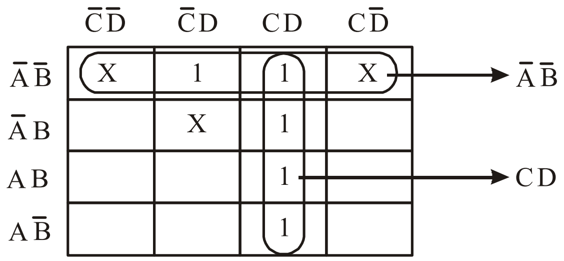
Solution. The function

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

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 1, 3, 7, 11 and 15. The output is *X* corresponding to the term 0, 2 and 5. This is shown in Fig. 4-130. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 4-131. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 4-131.

Decimal Number	***Inputs***				***Outputs***	
	A	***B***	***C***	***D***	***F***	
0	0	0	0	0	x	
1	0	0	0	1	1	$\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	x	
3	0	0	1	1	1	$\bar{A}\bar{B}CD$
4	0	1	0	0	0	
5	0	1	0	1	x	
6	0	1	1	0	0	
7	0	1	1	1	1	$\bar{A}BCD$
8	1	0	0	0	0	
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	1	$A\bar{B}CD$
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	1	$ABCD$

Fig. 4-130



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	X	1	1	X
$\bar{A}B$		X	1	
AB			1	
$A\bar{B}$			1	

Fig. 4-131

We know that when grouping the 1*s*, *Xs* can be treated as 1*s* to make a larger grouping. The group of four squares in the upper column of Karnaugh map produces a product term $\bar{A}\bar{B}$. This is determined by observing that the group contains both *C* and $\bar{C}$ and *D* and $\bar{D}$ so these variables are eliminated.

Another group of four squares in the third row produces term *CD*. This is determined by observing that the group contains both *A* and $\bar{A}$ *B* and $\bar{B}$, so this variable is eliminated. Thus the resulting simplified expression,

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

Example 4-106. *Using K-map simplify the following function*

$$f(A, B, C, D) = \Sigma\ (0, 1, 6, 8, 9, 12, 13) + \Sigma\ (4, 10, 14)$$

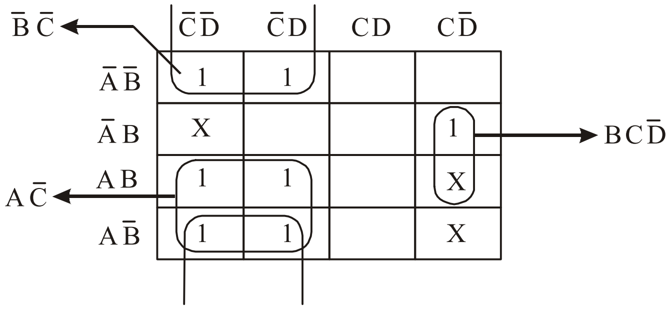
(RGTU., June 2009)

Solution. The function

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 1, 6, 8, 9, 12 and 13. The output is X corresponding to the term 4, 10 and 14. This is shown in Fig. 4-132. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 4-133. In order to simplify the Boolean expression represented on the Karnaugh map, group the Is as shown in the Fig. 4-133.

Decimal Number	***Inputs***				***Outputs***	
	A	***B***	***C***	***D***	***F***	
0	0	0	0	0	1	→ $\bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	1	→ $\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	X	
5	0	1	0	1	0	
6	0	1	1	0	1	→ $\bar{A}BC\bar{D}$
7	0	1	1	1	0	
8	1	0	0	0	1	→ $A\bar{B}\bar{C}\bar{D}$
9	1	0	0	1	1	→ $A\bar{B}\bar{C}D$
10	1	0	1	0	X	
11	1	0	1	1	0	
12	1	1	0	0	1	→ $AB\bar{C}\bar{D}$
13	1	1	0	1	1	→ $AB\bar{C}D$
14	1	1	1	0	X	
15	1	1	1	1	0	

Fig. 4-132



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1		
$\bar{A}B$	X			1
AB	1	1		X
$A\bar{B}$	1	1		X

Fig. 4-133

We know that when grouping the 1*s*, *Xs* can be treated as 1*s* to make a larger grouping. The group of four squares in the lower corner of Karnaugh map produces a product term $A\bar{C}$. This is determined by observing that the group contains both *B* and $\bar{B}$ and *D* and $\bar{D}$ so these variables are eliminated.

Another group of two squares in the first and last row produces term $\overline{B}\overline{C}$. This is determined by observing that the group contains both A and $\overline{A}$ and D and $\overline{D}$, so this variable is eliminated. Similarly another group of two squares in the last column produces term $BC\overline{D}$. This is determined by observing that the group contains both A and $\overline{A}$ so this variable is eliminated. Thus the resulting simplified expression,

$$f(A, B, C, D) = A\overline{C} + \overline{B}\overline{C} + BC\overline{D}$$

Example 4-107. *Simplify the following using K-map*

$$F = \Sigma m\,(0, 5, 8, 10, 11, 14, 15) + \Sigma\, d\,(3, 13)$$

(WBUT., 2011-12)

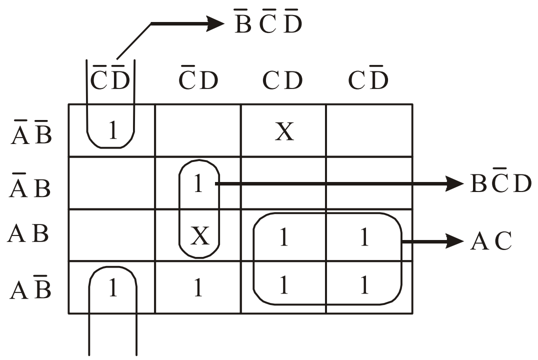
Solution. The function

$$F\,(A, B, C, D) = \Sigma m\,(0, 5, 8, 10, 11, 14, 15) + d\,(3, 13)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 5, 8, 10, 11, 14 and 15. The output is X corresponding to the term 3 and 13. This is shown in Fig. 4-134. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 4-135. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 4-135.

Decimal Number	*Inputs*				*Outputs*	
	A	*B*	*C*	*D*	*F*	
0	0	0	0	0	1	$\overline{A}\overline{B}\overline{C}\overline{D}$
1	0	0	0	1	0	
2	0	0	1	0	0	
3	0	0	1	1	X	
4	0	1	0	0	0	
5	0	1	0	1	1	$\overline{A}B\overline{C}D$
6	0	1	1	0	0	
7	0	1	1	1	0	
8	1	0	0	0	1	$A\overline{B}\overline{C}\overline{D}$
9	1	0	0	1	0	
10	1	0	1	0	1	$A\overline{B}C\overline{D}$
11	1	0	1	1	1	$A\overline{B}CD$
12	1	1	0	0	0	
13	1	1	0	1	X	
14	1	1	1	0	1	$ABC\overline{D}$
15	1	1	1	1	0	$ABCD$

Fig. 4-134



	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1		X	
$\overline{A}B$		1		
AB		X	1	1
$A\overline{B}$	1	1	1	1

Fig. 4-135

We know that when grouping the 1*s*, X*s* can be treated as 1*s* to make a larger grouping. The group of four squares in the lower corner of Karnaugh map produces a product term *AC*. This is determined by observing that the group contains both *B* and $\overline{B}$ and *D* and $\overline{D}$ so these variables are eliminated.

Another group of two squares in the first column produces term $\overline{B}\overline{C}\overline{D}.$ This is determined by observing that the group contains both *A* and $\overline{A}$, so this variable is eliminated. Similarly another group of two squares in the second column produces term $B\overline{C}D.$ This is determined by observing that the group contains both *A* and $\overline{A}$ so this variable is eliminated. Thus the resulting simplified expression,

$$F\,(A,\,B,\,C,\,D) = AC + \overline{B}\overline{C}\overline{D} + B\overline{C}D$$

Example 4-108. *Minimize the following expression using K-map*

$$f = \Sigma m\ (0,\,1,\,2,\,6,\,7) + d\ (0,\,5)$$ *(GBTU., 2010-11)*

Solution. Given: The function

$$f = \Sigma m\,(0,1,2,6,7) + d\,(0,5)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression i.e., 0, 1, 2, 6 and 7. The output is X corresponding to the term 0 and 5. This is shown in Fig. 4-136. We can map these values directly on to the three-variable Karnaugh map as shown in Fig. 4-137. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 4-137.

Inputs			***Outputs***	
A	***B***	***C***	***X***	
0	0	0	X	
0	0	1	1	→ $\overline{A}\overline{B}C$
0	1	0	1	→ $\overline{A}B\overline{C}$
0	1	1	0	
1	0	0	0	
1	0	1	X	
1	1	0	1	→ $AB\overline{C}$
1	1	1	1	→ ABC

Fig. 4-136

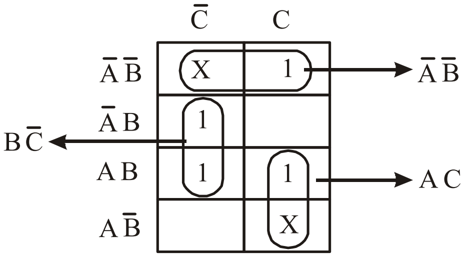


Fig. 4-137

We know that when grouping the 1*s*, *Xs* can be treated as 1*s* to make a larger grouping. The group of two squares in the upper corner of Karnaugh map produces a product term $\overline{A}\overline{B}.$ This is determined by observing that the group contains both *C* and $\overline{C}$ so these variables are eliminated.

Another group of two squares in the first column produces column produces term $B\overline{C}.$ This is determined by observing that the group contains both *A* and $\overline{A}$, so this variable is eliminated. Similarly another group of two squares in the second column produces term *AC*. This is determined by observing that the group contains both *B* and $\overline{B}$ so this variable is eliminated. Thus the resulting simplified expression,

$$f\ (A,\,B,\,C) = \overline{A}\overline{B} + B\overline{C} + AC$$

Example 4-109. *Minimize the following function using Karnaugh map and realize the minimized function using NOR gates.*

$$F\,(A,\,B,\,C,\,D) = B\overline{C} + \overline{A}B + BC\overline{D} + \overline{A}\overline{B}D + A\overline{B}\overline{C}D$$

(RGTU., Dec. 2011)

Solution. First of all, notice that the given *SOP* expression is in the nonstandard *SOP* form because the product terms $B\overline{C}, \overline{A}B, BC\overline{D}$ has a variable. Let us convert the given *SOP* expression into a standard *SOP* expression into a standard *SOP* form as shown below :

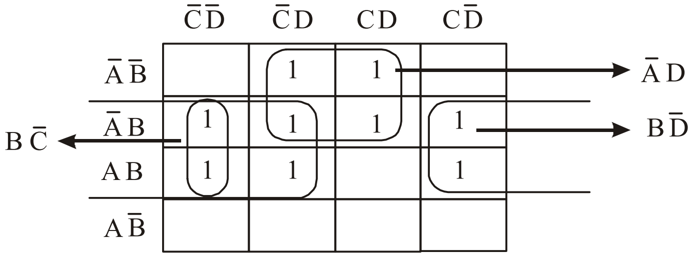
$$F = B\overline{C}(A+\overline{A})(D+\overline{D}) + \overline{A}B(C+\overline{C})(D+\overline{D})BC\overline{D}(A+\overline{A}) + \overline{A}\overline{B}D(C+\overline{C}) + A\overline{B}\overline{C}D$$

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

$$+ \overline{A}BC\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}\overline{C}D + A\overline{B}\overline{C}D$$

$$= AB\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}\overline{C}D$$

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



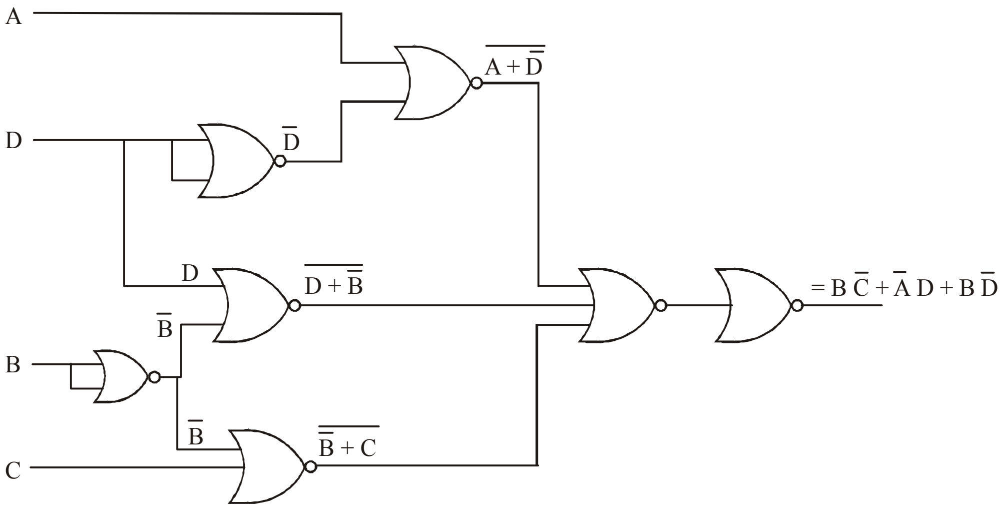
	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$		1	1	
$\overline{A}B$	1	1	1	1
AB	1	1		1
$A\overline{B}$				

Fig. 4-138.

This expression can be mapped on to the four-variable Karnaugh by entering 1 for each product term in the corresponding square as shown in Fig. 4-138.

In order to simplify the *SOP* expression, the 1*s* can be grouped together as shown by the loop around the 1*s*. *The four* 1*s* looped together form the first and second columns, contain both *A* and $\overline{A}$ and *D* and $\overline{D}$, so these variables are eliminated and the resulting product term is $B\overline{C}$. Similarly, the four 1*s* looped together form the second and the third column contain both *B* and $\overline{B}$ and *C* and $\overline{C}$ so these variables are eliminated and the resulting product term is $\overline{A}D$. Similarly, the four 1*s* looped together form the fourth and the first column contain both *A* and $\overline{A}$ and *C* and $\overline{C}$ so these variables are eliminated and the resulting product term,

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



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

Fig. 4-139. Shows the NOR gate implementation of function.

Example 4-110. *Using a K-map simplify the following function and realize using NAND gate*

$$F(A, B, C, D) = \Sigma\,(1, 4, 6, 7, 8, 9, 10, 11, 15)$$

(RGTU., June 2011)

Solution. The function

$$f(A, B, C, D) = \Sigma\,(1, 4, 6, 7, 8, 9, 10, 11, 15)$$

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression *i.e.*, 1, 4, 6, 7, 8, 9, 10, 11 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 4-140. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 4-141.

Decimal	***Inputs***				***Outputs***	
Number	***A***	***B***	***C***	***D***	***F***	
0	0	0	0	0	0	
1	0	0	0	1	1	→ $\overline{A}\overline{B}\overline{C}D$
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	1	→ $\overline{A}B\overline{C}\overline{D}$
5	0	1	0	1	0	
6	0	1	1	0	1	→ $\overline{A}B\overline{C}D$
7	0	1	1	1	1	→ $\overline{A}BCD$
8	1	0	0	0	1	→ $A\overline{B}\overline{C}\overline{D}$
9	1	0	0	1	0	→ $A\overline{B}\overline{C}D$
10	1	0	1	0	1	→ $A\overline{B}C\overline{D}$
11	1	0	1	1	1	→ $A\overline{B}CD$
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	1	→ $ABCD$

Fig. 4-140.

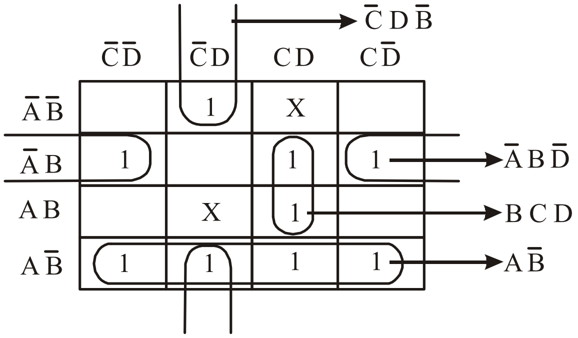


Fig. 4-141.

The group of four squares in the lower corner of Karnaugh map produces a product term $A\overline{B}$. This is determined by observing that the group contains both C and $\overline{C}$ and D and $\overline{D}$ so these variables are eliminated.

Another group of two squares in the first and last row produces term $\overline{B}\overline{C}D$. This is determined by observing that the group contains both A and $\overline{A}$, so this variable is eliminated. Similarly another group of two squares in the third column produces term BCD. This is determined by observing that the group contains both A and $\overline{A}$ so this variable is eliminated. Similarly another group of two squares in the second produces term $\overline{A}B\overline{D}$. This is determined by observing that the group contains both C and $\overline{C}$ so this variable is eliminated. Thus the resulting simplified expression,

$$f = A\overline{B} + \overline{B}\overline{C}D + BCD + \overline{A}B\overline{D}$$

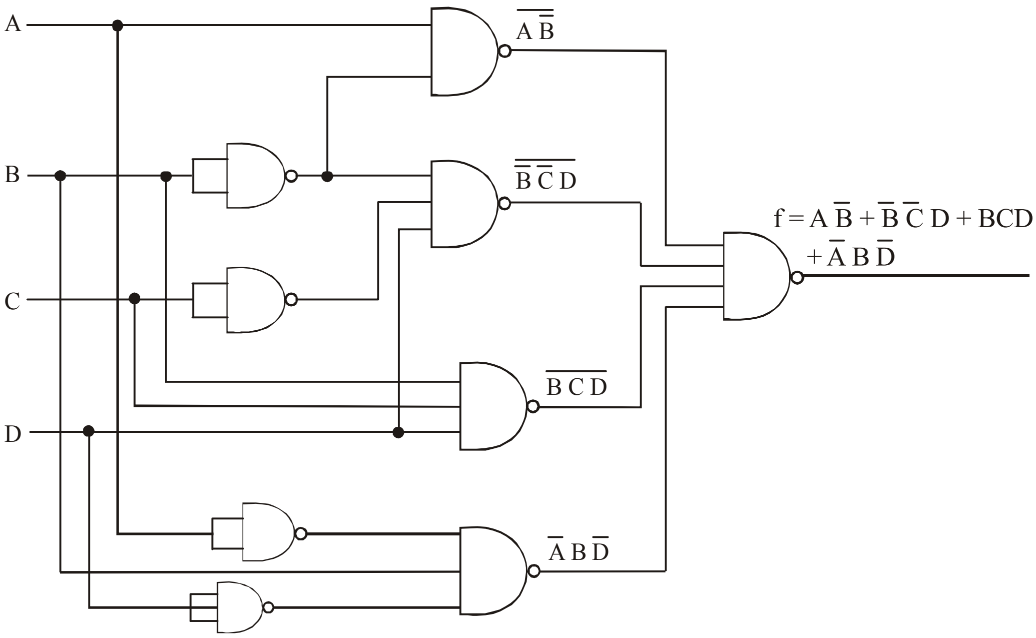

A
B
C
D
$\overline{A\overline{B}}$
$\overline{\overline{B}\overline{C}D}$
$\overline{BCD}$
$\overline{\overline{A}B\overline{D}}$
$f = A\overline{B} + \overline{B}\overline{C}D + BCD$
$+ \overline{A}B\overline{D}$

Fig. 4-142. *Shows the implementation of function using NAND gate.*

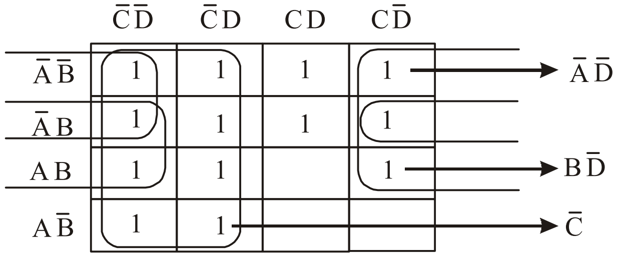
Example 4-111. *Simplify the Boolean function F (A, B, C, D) = Σ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) using K-map and Implement is using (a) NAND gates only (b) NOR gates only.*

(GTU., Dec. 2011)

Solution. The function

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

The given Boolean function indicates that its output is 1 corresponding to the term indicated within the expression *i.e.*, 0, 1, 2, 4, 5, 6, 8, 9, 12, 13 and 14. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1*s* as shown in Fig. 4-143.



	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	1	1	1
$\overline{A}B$	1	1	1	1
AB	1	1		1
$A\overline{B}$	1	1		

Fig. 4-143.

The group of eight squares in the first and second corner of Karnaugh map produces a product term $\overline{C}$. This is determined by observing that the group contains both A and $\overline{A}$ and B and $\overline{B}$ and D and $\overline{D}$ so these variables are eliminated.

Another group of four squares in the first and last column produces term $\overline{A}\overline{D}$. This is determined by observing that the group contains both B and $\overline{B}$ and C and $\overline{C}$, so this variable is eliminated. Similarly another group of four squares in the first and last column produces term $B\overline{D}$. This is determined by observing that the group contains both A and $\overline{A}$ and C and $\overline{C}$ so this variable is eliminated. Thus the resulting simplified expression,

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

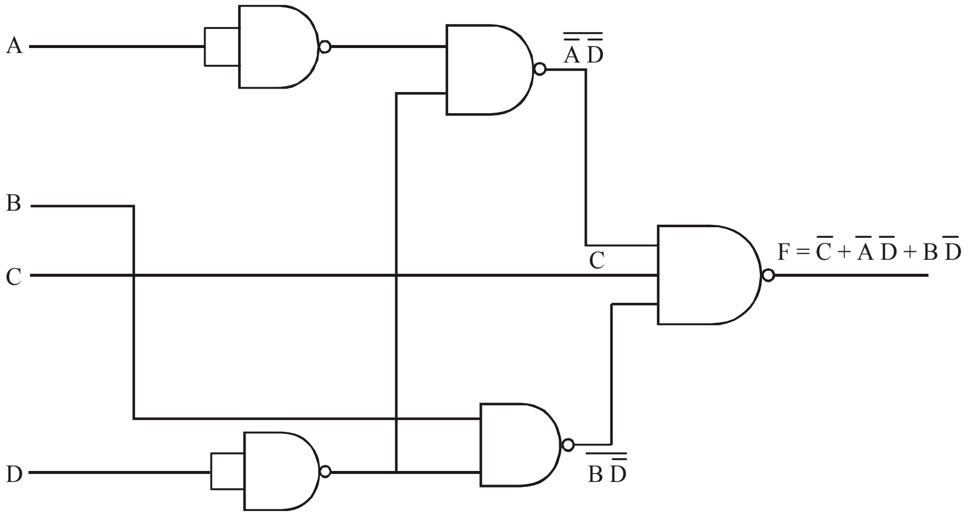

A
$\overline{\overline{A}\,\overline{D}}$
B
C
C
$F = \overline{C} + \overline{A}\,\overline{D} + B\,\overline{D}$
D
$\overline{B\,\overline{D}}$

Fig. 4-144. (a) *Using NAND gates only.*

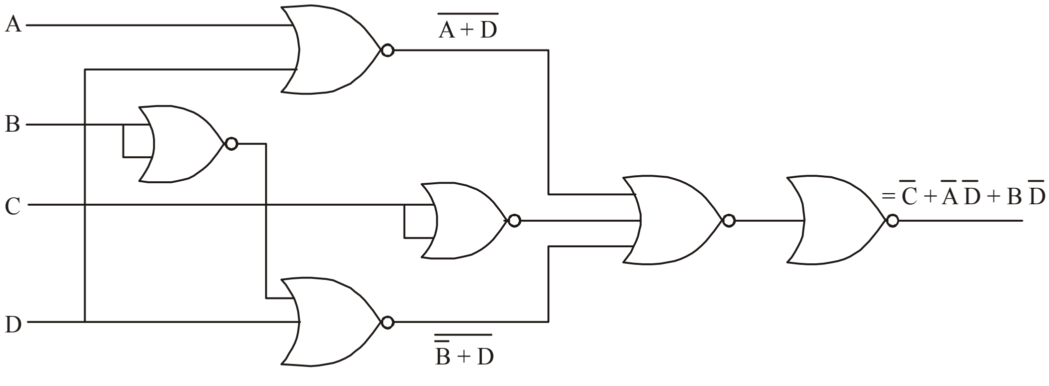
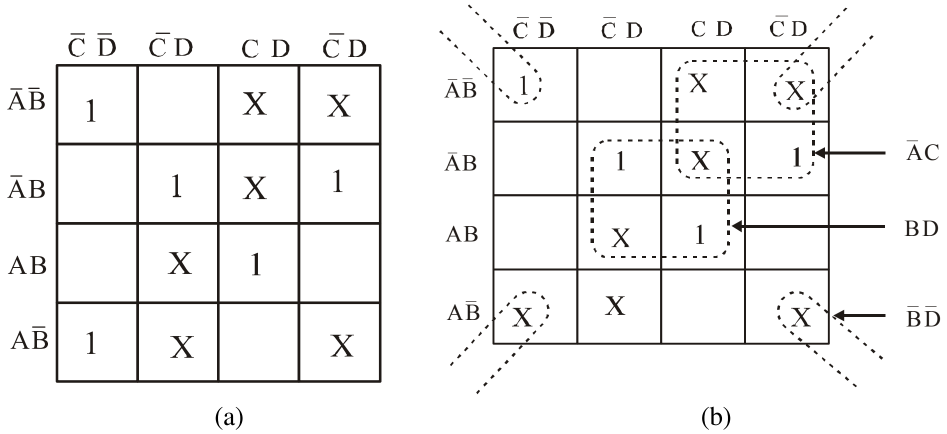

A
$\overline{A + D}$
B
C
$= \overline{C} + \overline{A}\,\overline{D} + B\,\overline{D}$
D
$\overline{\overline{B} + D}$

Fig. 4-144. (*b*) *Using NOR gates only.*

Example 4-112. *Consider the Karnaugh map shown in Fig.* 4-145(*a*). *Determine the logic function represented by the map and simplify it in the minimal form.*

(UPSC Engg. Services 1997)

Solution. We know that when grouping the 1*s*, *Xs* can be treated as 1*s* to make a larger grouping or as 0*s* if they cannot be used to advantage. Recall, the larger the group of 1*s*, the simpler the resulting term will be. Taking advantage of the *Xs* and using them as 1*s*, the grouping of 1*s* and *Xs* and using is as shown in Fig. 4-145 (b).



	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$\bar{C}D$
$\bar{A}\bar{B}$	1		X	X
$\bar{A}B$		1	X	1
AB		X	1	
$A\bar{B}$	1	X		X

(a)

	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$\bar{C}\,D$
$\bar{A}\bar{B}$	1		X	X
$\bar{A}B$		1	X	1
AB		X	1	
$A\bar{B}$	X	X		X

(b)

Fig. 4-145.

Notice the "wrap-around" four-square group that includes the 1*s* and *Xs* on fours corners of the Karnaugh map. This group produces a term $\bar{B}\,\bar{D}$. This is determined by observing that the group contains both *A* and $\bar{A}$ and *C* and $\bar{C}$, so these variables are eliminated.

Another group of four squares containing 1*s* and *Xs* around the centre of Karnaugh map is formed. This group produces a product term *BD*. This is determined by observing that the group contains both *A* and $\bar{A}$ and *C* and $\bar{C}$, so these variables are eliminated.

Another group of four containing 1*s* and *Xs* is formed near the top right corner of Karnaugh map. This group overlaps with the previous group and produces a product term $\bar{A}C$. This is determined by observing that the group contains both *B* and $\bar{B}$ *D* and $\bar{D}$, so these variables are elimiated.

The resulting simplified logic fuction is the sum of the threee product terms : $\bar{B}\bar{D}, BD$ and $\bar{A}C$, *i.e.,*

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

Example 4-113. *For the K-map shown in Fig.* 4-146, *find the minimized expression.*

	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$C\,\bar{D}$
$\bar{A}\,\bar{B}$	0	0	X	1
$\bar{A}\,B$	0	X	X	0
$A\,B$	1	X	1	1
$A\,\bar{B}$	1	1	X	1

X → Don't care

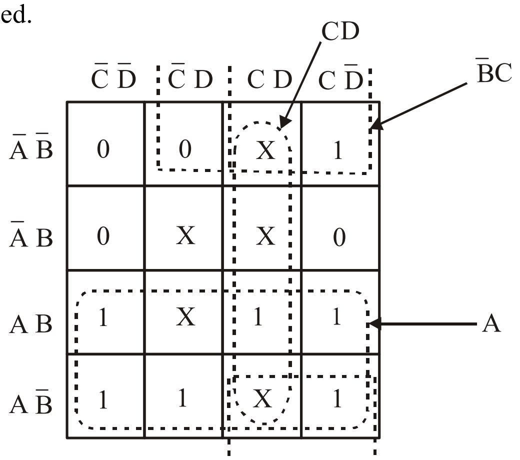
Fig. 4-146.

(U.P.S.C Engg. Service Electronic & Telecom Engg. 2002)

Solution. Given : The K-map shown in Fig. 4-146.

The grouping of 1*s* is as shown in Fig. 4-147. Notice the "wrap around" eight square groups that includes the 1*s* on the lower two rows of the Karnaugh map. The group produces a product term *A*. This is determined by observing that the group contains *B* and $\overline{B}$, *C* and $\overline{C}$, *D* and $\overline{D}$, so all these variables are elminated.

Another group of four with "wrap-around" adjacency is formed from the top and the bottom rows of the Karnaugh map. This group overlaps with the previous group and produces a product term $\overline{B}C$, This is determined by observing that this group contains both *A* and $\overline{A}$, and *D* and $\overline{D}$, so these variables are eliminated.



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	X	1
$\overline{A}B$	0	X	X	0
AB	1	X	1	1
$A\overline{B}$	1	1	X	1

Fig. 4-147.

Another group of four with "wrap-around" adjacency is formed from the third column of the Karnaugh map. This group overlaps with the previous two groups and produces a product term *CD*. This is determined by observing that this group contains both *A* and $\overline{A}$, abd *B* and $\overline{B}$, so these variables are eliminated.

4-41. Quine-McCluskey Tabular Method

The Quine-McCluskey tabular method is a method used for minimization of Boolean functions with more than four variables. It is functionally identical the Karnaugh mapping. However the tabular form makes it more efficient for use in computer algorithms.

The method involves two steps :

1. Finding all the prime implicants of the function, *i.e.* expand the given Boolean expression if it is not in the expanded form.
2. Use these prime implicants in a prime implicant chart or table to find essential prime implicants of the functions, as well as other prime implicants that are necessary to cover the function. It means different terms in the expression are divided into groups depending upon number of 1'*s* they have.

Let's take an example to explain the Quine-McCluskey tabular method in detail

$$ABC + \overline{A}\,BC + A\,\overline{BC} + \overline{A}\ \overline{BC}$$

The grouping of different terms and the arrangement of different terms within the group are shown in Table 4-10.

The Quine-Mccluskey Tabular method is a method used for minimization of Boolean functions with more than four variables. It is functionally identical to karnaugh mapping. However the tabular form makes it more efficient for use in computer algorithms.

The method involves two steps:

1. Finding all the prime implicants of the function, i.e., expand the given Boolean expression if it is not in the expanded form.
2. Use those prime implicants in a prime implicants chart or table to find essential prime implicants of the function, as well as other prime implicants that are necessary to cover the function. It means different terms in the expression are divided into groups depending upon number of is they have.

Table 4-10.

$\bar{A}.\bar{B}.\bar{C}$		000	First group
$\overline{A.\bar{B}.\bar{C}}$		100	Second group
$\overline{\bar{A}.B.C}$	$\longrightarrow$	011	Third group
$A.\bar{B}.C$		101	
$\overline{ABC}$		111	Fourth group

1. The terms of the first group are successively matched with those in the next adjacent higher order group to look for any possible matching and consequent reduction. The terms are considered matched when all literals except for one match. The pairs of matched terms are replaced with a single term where the position of the unmatched literals is replaced with a dash (—). These new terms formed as a result of the matching process find a place in the second table. The terms in the first table that do not find a match are called the prime implicants and are marked with an asterisk (*). The matched terms are ticked (√).
2. Terms in the second group are compared with those in the third group to look for a possible match. Again, terms in the second group that do not find a match become the prime implicants.
3. The process continues until we reach the last group. This completes the first round of matching. The terms resulting from the matching in the first round are recorded in the second table.
4. The next step is to perform matching operations in the second table. While comparing the terms for a match, it is important that a dash $\left(\frac{1}{M}\right)$ is also treated like any other literal, that is, the dash signs also need to match. The process continues on to the third table, the fourth tanle and so on until the terms becomes irreducible any further.
5. An optimum selection of prime implicats to account for all the original terms constitutes the terms for the minimized expression. Althogh optional (also called 'don't care') terms are considered for matching, they do not have to be accounted for once prime implicants have been identified.

Let us consider an example. Consider the following sum-of-product expression, its shown in table 4-11.

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

Converting in sum-of-product

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

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

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

Table 4-11

	A	*B*	*C*	*D*
$\overline{A}.\overline{B}.\overline{C}.D$ (1)	0	0	0	1
$\overline{A}.\overline{B}.C.D$ (3)	0	0	1	1
$\overline{A}.B.\overline{C}.$ (4)	0	1	0	0
$\overline{A}.B.\overline{C}.D$ (5)	0	1	0	1
$\overline{A}.B.C.$ (6)	0	1	1	0
$\overline{A}.B.C.D$ (7)	0	1	1	1
$A.\overline{B}.\overline{C}.D$ (9)	1	0	0	1
$A.B.\overline{C}.$ (12)	1	1	0	0
$A.B.\overline{C}.D$ (13)	1	1	0	1

The formation of groups is shown in Table 4-12.

Table 4-12.

	A	*B*	*C*	**D**	
First Group					
$\overline{A}.\overline{B}.\overline{C}.D$ (1)	0	0	0	1	✓
$\overline{A}.B.\overline{C}.\overline{D}$ (4)	0	1	0	0	✓
Second Group					
$\overline{A}.\overline{B}.C.D$ (3)	0	0	1	1	✓
$\overline{A}.\mathrm{B}.\overline{C}.D$ (5)	0	1	0	1	✓
$\overline{A}.B.C.\overline{D}$ (6)	0	1	1	0	✓
$A.\overline{B}.\overline{C}.D$ (9)	1	0	0	1	✓
$A.B.\overline{C}.\overline{D}$ (12)	1	1	0	0	✓
Third Group					
$\overline{\mathrm{A}}.B.C.D$ (7)	0	1	1	1	✓
$A.B.\overline{C}.D$ (13)	1	1	0	1	✓

The second round of matching is shown in Table 4-13. Each term in the first group is compared with every term in the second group.

Table 4-13

	A	*B*	*C*	*D*	
1,3	0	0	–	1	✓
1, 5	0	–	0	1	✓
1, 9	–	0	0	1	✓
4, 5	0	1	0	–	✓
4, 6	0	1	–	0	✓
4, 12	–	1	0	0	✓
3, 7	0	–	1	1	✓
5, 7	0	1	–	1	✓
5, 13	–	1	0	1	✓
6, 7	0	1	1	–	✓
9, 13	1	–	0	1	✓
12, 13	1	1	0	–	✓

The third round of matching is shown in Table 4-14. Each term in the first group is compared with every term in the second group.

Table 4-14.

	A	*B*	*C*	*D*	
1, 3, 5, 7	0	–	–	1	*
1, 5, 9, 13	–	–	0	1	*
4, 6, 5, 7	0	1	–	–	*
4, 5, 12, 13	–	1	0	–	*

The next table is what is known as the prime implicant table. The prime implicant table contains all the original terms in different columns and all the prime implicants recorded in different rows as shown in Table 4-15 below:

Table 4-15.

Prime implicants	***0001 (1)***	***0011 (3)***	***0100 (4)***	***0101 (5)***	***0110 (6)***	***0111 (7)***	***1001 (9)***	***1100 (12)***	***1101 (13)***		
1, 3, 5, 7	✓	✓		✓		✓				0– –1	$\bar{A}.D$
1, 5, 9, 13	✓			✓			✓		✓	– –01	$\bar{C}.D$
4, 6, 5, 7			✓	✓	✓	✓				01– –	$\bar{A}.B$
4, 5, 12, 13			✓	✓				✓	✓	–10–	$B.\bar{C}$

Now find all the columns that contains a single tick (✓) and circle in then, place an asterisk to the left of those rows in which you have circled a circle. The rows marked with an asterisk are the essential prime implicants as shown in Table 4-16.

Table 4-16.

Prime implicants	***0001 (1)***	***0011 (3)***	***0100 (4)***	***0101 (5)***	***0110 (6)***	***0111 (7)***	***1001 (9)***	***1100 (12)***	***1101 (13)***		
1, 3, 5, 7*	✓	✓		✓		✓				0––1	$\overline{A}.D$
1, 5, 9, 13*	✓			✓			✓		✓	––01	$\overline{C}.D$
4, 6, 5, 7*			✓	✓	✓	✓				01––	$\overline{A}.B$
4, 5, 12, 13*			✓	✓				✓	✓	–10–	$B.\overline{C}$

Include the essential prime implicants in the minimal sum.

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

Example 4-114. *Using Quenie-McCluskey method and simplify the following function*

$f(a, b, c, d) = \Sigma m\ (1, 2, 3, 8, 9)$ *(VTU, Jan./Feb. 2005)*

Solution: Given: The logic function

$$f(a, b, c, d) = \Sigma m\,(1, 2, 3, 8, 9)$$

The given function are arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-17.

Table 4-17.

Group	***Minterm***	***Variable***			
		A	***B***	***C***	***D***
0	0	0	0	0	0
1	1	0	0	0	1
	2	0	0	1	0
	8	1	0	0	0
2	9	1	0	0	1
	3	0	0	1	1

The second round of matching is shown in Table 4-18. Each term in the first group is compared with every term in the second group.

Table 4-18.

Group	***Minterm***	***Variable***			
		a	***b***	***c***	***d***
0	0,1	0	0	0	-
	0,2	0	0	-	0
	0,8	-	0	0	0
1	1,9	-	0	0	1
	1,3	0	0	-	1
	2,3	0	0	1	-
	8,9	1	0	0	-

The third round of matching is shown in Table 4-19. Each term in the first group is compared with every term in the second group.

Table 4-19.

Group	*Minterm*	*Variable*			
		a	***b***	***c***	***d***
0	0, 1, 2, 3	0	0	-	-
	0, 1, 8, 9	-	0	0	-
	0, 2, 1, 3	0	0	-	-
	0, 8, 1, 9	-	0	0	-

The next table is what is known as the prime implicant table the prime implicant table contains all the original terms in different columns and all the prime implicant recorded in different rows as shown in Table 4-20 below:

Table 4-20.

Terms	*Prime Implicants*	*Minterm*					
		0	1	2	3	8	9
$\overline{a}\overline{b}$	0, 1, 2, 3	✓	✓	✓	✓		
$\overline{b}\overline{c}$	0, 1, 8, 9	✓	✓			✓	✓

Now we find the columns that contains a single tick (✓) and circle in then, then place an asterisk to the left of those rows in which you have circled a circle. The rows marked with an asterisk are the essential prime implicants as shown in Table 4-21.

Table 4-21.

Term	*Prime Implicants*	*Minterm*					
		0	1	2	3	8	9
$\overline{a}\overline{b}$ *	0, 1, 2, 3	✓	✓	(✓)	(✓)		
$\overline{b}\overline{c}$ *	0, 1, 8, 9	✓	✓			(✓)	(✓)

We see that marks is in columns 2, 3, 8 and 9. Thus the function

$$f = \overline{a}\overline{b} + \overline{b}\overline{c}$$ **Ans.**

Example 4-115. *Using Quine-Mcluskey Method and prime implicant reduction, determine the minimal product-of-cum (POS) expression for the following using decimal notation*

$$F(w, x, y, z) = \Sigma m\ (1, 2, 3, 5, 9, 12, 14, 15) + \Sigma d\ (4, 8, 11)$$ *(VTU, July 2006)*

Solution. Given: The logic function

$$F(w, x, y, z) = \Sigma m\ (1, 2, 3, 5, 9, 12, 14, 15) + \Sigma d\ (4, 8, 11)$$

All minterm in the binary form is shown in Table 4-22.

Table 4-22.

Minterm	*Variable*			
	W	***X***	***Y***	***Z***
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
8	1	0	0	0
9	1	0	0	1
11	1	0	1	0
12	1	1	0	1
14	1	1	1	0
15	1	1	1	1

The given function are arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-23.

Table 4-23.

Group	*Minterm*	*Vriable*			
		W	***X***	***Y***	***Z***
1	1	0	0	0	1
	2	0	0	1	0
	4	0	1	0	0
	8	1	0	0	0
2	3	0	0	1	1
	5	0	1	0	1
	9	1	0	0	1
	12	1	1	0	0
3	11	1	0	1	1
	14	1	1	1	0
4	15	1	1	1	1

Matching the group of pair according to the only variable difference only. The group pair is shown in Table 4-24.

Table 4-24.

Group	*Minterm*	*Variable*			
		W	*X*	*Y*	*Z*
1	1, 3	0	0	-	1
	1, 5	0	-	0	1
	1, 9	-	0	0	1
	2, 3	0	0	1	-
	4, 5	0	1	0	0
	4, 12	-	1	0	0
	8, 9	1	0	0	-
	8, 12	1	-	0	0
2	3, 11	-	0	1	1
	9, 11	1	0	-	1
	12, 14	1	1	-	0
3	11, 15	1	-	1	1
	14, 15	1	1	1	-

The third round of matching is shown in Table 4-25. Each term in the first group is compared with every term in the second group.

Table 4-24.

Group	*Minterm*	*Variable*			
		W	*X*	*Y*	*Z*
1	1, 3, 9, 11	-	0	-	1
	1, 5	0	-	0	1
	1, 9, 3, 11	-	0	-	1
	2, 3	0	0	1	-
	4, 5	0	1	0	-
	4, 12	-	1	0	0
	8, 9	1	0	0	-
	8, 12	1	-	0	0
2	12, 14	1	1	-	0
3	11, 15	1	-	1	1
	14, 15	1	1	1	-

The next table is what is known as the prime implicant Table The prime implicant table contains all the original terms in different columns and all the prime implicants recoreded in different rows as shown in Table 4-26 below:

Table 4-27.

Terms	***Prime***	***Minterm***										
	Implicants	***1***	***2***	***3***	***4***	***5***	***8***	***9***	***11***	***12***	***14***	***15***
$\overline{x}z$	1, 3, 9, 11	✓		✓				✓	✓			
$\overline{w}\overline{y}z$	1, 5	✓				✓						
$\overline{w}\overline{x}y$*	2, 3		✓	✓								
$\overline{w}x\overline{y}$	4, 5				✓	✓						
$x\overline{y}\overline{z}$	4, 12				✓					✓		
$w\overline{x}\overline{y}$	8, 9						✓	✓				
$w\overline{y}\overline{z}$	8, 12						✓			✓		
$wx\overline{z}$	12, 14									✓	✓	
wxz	11, 15								✓			✓
wxy	14, 15										✓	✓

Now we find the columns that contains a single tick (✓) and circle in then, then place an asterisk to the left of those rows in which you have circled a circle. The rows marked with an asterisk are the essential prime implicants as shown in Table 4.27. Hence the prince corresponding to 2, 3 is included in the final expression. But this expression, does not contain all the terms so search for the multi ✓ minterm column which are not included in the final expression. Minterm 1 terms are not included in the final expression, so we can include either prime implicants 1, 5 or 1, 3, 9, 11. Let us includes prime implicant which has more minterm. Thus minterm 1, 3, 9, 11 is include. Now column of minterm 3 is already included and minterm 4 is don't care condition. Column minterm 5 has which is not included yet, therefore prime implicant 1, 5 is included in final expression. Minterm 8 and 11 are don't care. Minterm 12 has minterm 12 which is not yet included, We include prime implicants 12, 14 because with this inclusion we can also cover minterm 14 in the final expression. The minterm 15 column minterm 15 can be included in the final expression by including minterm 14, 15.

Table 4-27.

Terms	***Prime***	***Minterm***										
	Implicants	***1***	***2***	***3***	***4***	***5***	***8***	***9***	***11***	***12***	***14***	***15***
$\overline{x}z$	**1, 3, 9, 11**	(✓)		(✓)				(✓)	(✓)			
$\overline{w}\overline{y}z$	1, 5	(✓)				(✓)						
$\overline{w}\overline{x}y$	2, 3		(✓)	(✓)								
$\overline{w}x\overline{y}$	4, 5				✓	✓						
$x\overline{y}\overline{z}$	4, 12				✓					✓		
$w\overline{x}\overline{y}$	8, 9						✓	✓				
$w\overline{y}\overline{z}$	8, 12						✓			✓		
$wx\overline{z}$	12, 14									(✓)	(✓)	
wxz	11, 15								✓			✓
wxy	14, 15										(✓)	(✓)

Thus the final expression

$$f(w, x, y, z) = \bar{x}z + \bar{w}\bar{y}z + \bar{w}\bar{x}y + wx\bar{z} + wxy \textbf{ Ans.}$$

Example 4-116. *Using Quine -Mclusky tabulation method, obtain the set of prime implicants for the function*

$$f(a, b, c, d) = \Sigma m(0, 1, 4, 5, 9, 10, 12, 14, 15) + \Sigma d(2, 8, 13)$$

Solution. Given: The logic function

$$F(w, x, y, z) = \Sigma m(1, 2, 3, 5, 9, 12, 14, 15) + \Sigma d(4, 8, 11)$$

All minterm in the binary form is shown in Table 4-18.

Table 4-28.

Minterm	*Variable*			
	a	***b***	***c***	***d***
0	0	0	0	0
1	0	0	0	1
4	0	1	0	0
5	0	1	0	1
9	1	0	0	1
10	1	0	1	0
12	1	1	0	0
14	1	1	1	0
15	1	1	1	1
2	0	0	1	0
8	1	0	0	0
13	1	1	0	1

The given function are arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-29.

Table 4-29.

Group	*Minterm*	*Variable*			
		a	***b***	***c***	***d***
0	0	0	0	0	0
1	1	0	0	0	1
	2	0	0	1	0
	4	0	1	0	0
	8	1	0	0	0
2	5	0	1	0	1
	9	1	0	0	1
	10	1	0	1	0
	12	1	1	0	0
3	13	1	1	0	1
	14	1	1	1	0
4	15	1	1	1	1

Matching the group of pair according to the only variable difference only. The group pair is shown in Table 4-30.

Table 4-30.

Group	*Minterm*	*Variable*			
		a	***b***	***c***	***d***
0	0, 1	0	0	0	-
	0, 2	0	0	-	0
	0, 4	0	-	0	0
	0, 8	-	0	0	0
1	1, 5	0	-	0	1
	1, 9	-	0	0	1
	2, 10	-	0	1	0
	4, 5	0	1	0	-
	4, 12	-	1	0	0
	8, 9	1	0	0	-
	8, 10	1	0	-	0
	8, 12	1	-	0	0
2	5, 13	-	1	0	1
	9, 13	1	-	0	1
	10, 14	1	-	1	0
	12, 14	1	1	-	0
3	13, 15	1	1	-	1
	14, 15	1	1	1	1

The third round of matching is shown in Table 4-31. Each term in the first group is compared with every term in the second group.

Table 4-31.

Group	*Minterm*	*Variable*			
		a	***b***	***c***	***d***
0	0, 1, 8, 9	-	0	0	-
	0, 2, 8, 10	-	0	-	0
	0, 4, 1, 5	0	-	0	-
	0, 8, 4, 12	-	-	0	0
1	1, 9, 5, 13	-	-	0	1
	4, 12, 5, 13	-	1	0	-
	8, 10, 12, 14	1	-	-	0
2	12, 14, 13, 15	1	1	-	-

The fourth round of matching is shown in Table 4-32. Each term in the first group is compared with every term in the second group.

Table 4-32.

Group	*Minterm*	*Variable*			
		a	***b***	***c***	***d***
0	0, 1, 8, 9, 4, 12, 5, 13	-	-	0	-
	0, 2, 8, 10	-	0	-	0
1	8, 10, 12, 14	1	-	-	0
2	12, 14, 13, 15	1	1	-	-

The next table is what is known as the prime implicant Table The prime implicant table contains all the original terms in different columns and all the prime implicant recorded in different rows as shown in Table 4-33 below:

Table 4-33.

Terms	*Prime*	*Minterm*											
	Implicants	*0*	*1*	*2*	*4*	*5*	*8*	*9*	*10*	*12*	*13*	*14*	*15*
$\bar{c}$*	0, 1, 8, 9, 4, 1, 2, 5, 13	✓	(✓)		(✓)	(✓)	✓	(✓)		✓	✓		
$\bar{b}\bar{d}$*	0, 2, 8, 10	✓		(✓)			✓		✓				
$a\bar{d}$	8, 10, 12, 14						✓		✓	✓		✓	
ab*	12, 14, 13, 15									✓	✓	✓	(✓)

Now we find the columns that contains a single tick (✓) and circle in then, then place an asterisk to the left of those rows in which you have circled a circle. This rows marked with an astericsk are the essential prime implicant as shown in Table 4-34.

$$f(a, b, c, d) = \bar{c} + \bar{b}\bar{d} + ab$$ **Ans.**

Example 4-117. *Simplify the following switching function using Quine-Mccluskey method*

$$f(A, B, C, D) = \Sigma m(1, 3, 13, 15) + \Sigma d(8, 9, 10, 11)$$

Solution: Given: The logic function

$$f(A, B, C, D) = \Sigma m(1, 3, 13, 15) + \Sigma d(8, 9, 10, 11)$$

All minterm in the binary form is shown in Table 4-35.

Table 4-35.

Minterm	*Variable*			
	A	***B***	***C***	***D***
1	0	0	0	1
3	0	0	1	1
13	1	1	0	1
15	1	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

The given function are arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-36.

Table 4-36.

Group	*Minterm*	*Variable*			
		A	***B***	***C***	***D***
1	1	0	0	0	1
	8	1	0	0	0
2	3	0	0	1	1
	9	1	0	0	1
	10	1	0	1	0
3	13	1	1	0	1
	11	1	0	1	1
4	15	1	1	1	1

Matching the group of pair according to the only variable difference only. The group pair is shown in Table 4-37.

Table 4-37.

Group	*Minterm*	*Variable*			
		A	***B***	***C***	***D***
1	1, 3	0	0	-	1
	1, 9	-	0	0	1
	8, 9	1	0	0	-
	8, 10	1	0	-	0
2	3, 11	-	0	1	1
	9, 13	1	-	0	1
	9, 11	1	0	-	1
	10, 11	1	0	1	-
	13, 15	1	1	-	1
3	11, 15	1	-	1	1

The third round of matching is shown in Table 4-38. Each term in the first group is compared with every term in the second group.

Table 4-38.

Group	*Minterm*	*Variable*			
		A	***B***	***C***	***D***
1	1, 3, 9, 11	-	0	-	1
	8, 9, 10, 11	1	0	-	-
2	9, 11, 13, 15	1	-	-	1

The next table is what is known as the prime implicant Table The prime implicant table contains all the original terms in different columns and all the prime implicants recorded in different rows shown in Table 4-39 below:

Table 4-39.

Terms	*PrimeImplicants*	*Minterm*							
		1	*3*	*8*	*9*	*10*	*11*	*13*	*15*
$\overline{B}D$	1, 3, 9, 11	✓	✓		✓		✓		
$\overline{B}$	8, 9, 10, 11			✓	✓	✓	✓		
AD	9, 11, 13, 15				✓		✓	✓	✓

Now we find the columns that contains a single tick (✓) and circle in then place an asterisk to the left of those rows in which you have circled a circle. This rows marked with an asterisk are the essential prime implicants as shown in Table 4-40.

Table 4-40.

Terms	*PrimeImplicants*	*Minterm*							
		1	*3*	*8*	*9*	*10*	*11*	*13*	*15*
$\overline{B}D$	1, 3, 9, 11	(✓)	(✓)		✓		✓		
$\overline{B}$	8, 9, 10, 11			✓	✓	✓	✓		
AD	9, 11, 13, 15				✓		✓	(✓)	(✓

Thus the resultant function,

$$f(A, B, C, D) = \overline{B}D + AD \text{ **Ans.**}$$

Example 4-118. *Using the Quine-Mclusky method and prime implication table reductions, determine the minimal sums for the incomplete Boolean function.*

$$f(v.w.x.y.z) = \Sigma m(4, 5, 9, 11, 12, 14, 15, 27, 30) + d_c(1, 17, 25, 26, 31)$$

(VTU, Jan/Feb 2004, 2006)

Solution. Given: The logic function

$$f(v.w.x.y.z) = \Sigma m(4, 5, 9, 11, 12, 14, 15, 27, 30) + d_c(1, 17, 25, 26, 31)$$

The given function is arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-41.

Table 4-41.

Group	*Minterm*	*Variable*				
		A	*B*	*C*	*D*	*E*
1	1	0	0	-	1	1
	4	0	0	1	0	0
2	5	0	0	1	0	1
	9	0	1	0	0	1
	12	0	1	1	0	0
	17	1	0	0	0	1
3	11	0	1	0	1	1
	14	0	1	1	1	0
	25	1	1	0	0	1
	2	1	1	0	1	0
4	15	0	1	1	1	1
	27	1	1	0	1	1
	30	1	1	1	1	0
5	31	1	1	1	1	1

Matching the group of pair according to the one variable difference only. These are shown in Table 4-42.

Table 4-42.

Group	*Minterm*	*Variable*				
		A	*B*	*C*	*D*	*E*
1	1, 5	0	0	-	0	1
	4, 5	0	0	1	0	-
	1, 9	0	-	0	0	1
	1, 17	-	0	0	0	1
	4, 12	0	-	1	0	0
2	9, 11	0	1	0	-	1
	9, 25	-	1	0	0	1
	12, 14	0	1	1	-	0
	17, 25	1	-	0	0	1
3	11, 15	0	1	-	1	1
	11, 27	-	1	0	1	1
	14, 15	0	1	1	1	-
	14, 30	-	1	1	1	0
	25, 27	1	1	0	-	1
	26, 27	1	1	0	1	-
	26, 30	1	1	-	1	0
4	15, 31	-	1	1	1	1
	27, 31	1	1	-	1	1
	30, 31	1	1	1	1	-

Again matching the group of pair in one difference only. This is shown in Table 4-43.

Table 4-43.

Minterm	*Variable*				
	a	***b***	***c***	***d***	***E***
1, 5	0	0	-	0	1
4, 5	0	0	1	0	-
4, 12	0	-	1	0	0
12, 14	0	1	1	-	0
1, 9, 17, 25	-	-	0	0	1
9, 11, 25, 27	-	1	0	-	1
11, 15, 27, 31	-	1	-	1	1
14, 15, 30, 31	-	1	1	1	-
26, 27, 30, 31	1	1	-	1	-

The prime implicant table is shown in Table 4-44 below:

Table 4-44.

Terms	*Prime Implicants*	*Minterms*													
		1	*4*	*5*	*9*	*11*	*12*	*14*	*15*	*17*	*25*	*26*	*27*	*30*	*31*
$\overline{A}\ \overline{B}\ \overline{C}E$	1, 5	✓		✓											
$\overline{A}\ \overline{B}C\overline{D}$	4, 5		✓	✓											
$\overline{A}CD\overline{E}$	4, 12		✓				✓								
$\overline{A}BC\overline{E}$	12, 14						✓	✓							
$\overline{C}\ \overline{D}E$	1, 9, 17, 25	✓			✓					✓	✓				
$B\overline{C}E$	9, 11, 25, 27				✓				✓		✓		✓		
BDE	11, 15, 27, 31						✓		✓				✓		✓
BCD	14, 15, 30, 31							✓	✓					✓	✓
ABD	26, 27, 30, 31											✓	✓	✓	✓

Now we find the columns that contains a single tick (✓) and circle in then, then place an asterisk to the left of those rows in which you have circled a circle. The rows marked with an asterisk are the essential prime implicants as shown in Table 4-45.

Table 4-45.

Terms	*Prime Implicants*	*Minterms*													
		1	*4*	*5*	*9*	*11*	*12*	*14*	*15*	*17*	*25*	*26*	*27*	*30*	*31*
$\overline{A}\,\overline{B}\,\overline{C}E$	1, 5	✓		✓											
$\overline{A}\,\overline{B}C\overline{D}$	4, 5*		(✓)	(✓)											
$\overline{A}CD\overline{E}$	4, 12*		(✓)				(✓)								
$\overline{A}BC\overline{E}$	12, 14						✓	✓							
$\overline{C}\,\overline{D}E$	1, 9, 17, 25	✓			✓					✓	✓				
$B\overline{C}E$	9, 11, 25, 27*				(✓)	(✓)					(✓)		(✓)		
BDE	11, 15, 27, 31					✓			✓				✓		✓
BCD	14, 15, 30, 31*							(✓)	(✓)					(✓)	(✓)
ABD	26, 27, 30, 31											✓	✓	✓	✓

Thus the minimal sum of function

$$f(A, B, C, D) = \overline{A}\ \overline{B}\ C\overline{D} + \overline{A}\ CD\overline{E} + B\overline{C}\ E + BCD \textbf{ Ans.}$$

Example 4-119. *Minimize the following Boolean function by using Quine Mclusky method*

$F(A, B, C, D) = \Sigma(20, 28, 37, 39, 48, 56)$ *(GBTU/MTU, 2009-10)*

Solution: Given: The logic function

$$F(A, B, C, D) = \Sigma(20, 28, 37, 39, 48, 56)$$

The given function are arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-46.

Table 4-46.

Group	*Minterm*	*Variable*					
		A	*B*	*C*	*D*	*E*	*F*
2	20	0	1	0	1	0	0
	48	1	1	0	0	0	0
3	28	0	1	1	1	0	0
	37	1	0	0	1	0	1
	56	1	1	1	1	0	0
4	39	1	0	0	0	1	1

Matching the group of pair according to the only variable difference only. The group pair is shown in Table 4-47.

Table 4-47.

Minterm	*Variable*					
	A	***B***	***C***	***D***	***E***	***F***
20, 28	0	1	-	1	0	0
48, 56	1	1	-	0	0	0
37, 39	1	0	0	1	-	1

The prime implicant table is shown in Table 4-48 below:

Table 4-48.

Terms	***Prime***	***Minterm***					
	Implicants	***20***	***28***	***37***	***39***	***48***	***56***
$\overline{A}BD\overline{E}\overline{F}$	20, 28	✓	✓				
$AB\overline{D}\overline{E}\overline{F}$	48, 56					✓	✓
$A\overline{B}\overline{C}DF$	37, 39			✓	✓		

Now we find the columns that contains a single tick (✓) and circle in then, then place an asterisk to the left of those rows in which you have circled a circle. The rows marked with an asterisk are the essential prime implicants as shown in Table 4-49.

Table 4-49.

Terms	***Prime***	***Minterm***					
	Implicants	***20***	***28***	***37***	***39***	***48***	***56***
$\overline{A}BD\overline{E}\overline{F}$ *	20, 28	(✓)	(✓)				
$AB\overline{D}\overline{E}\overline{F}$ *	48, 56					(✓)	(✓)
$A\overline{B}\overline{C}DF$ *	37, 39			(✓)	(✓)		

We see that marks is in columns 20, 28, 37, 39, 48, and 56. Thus the function

$$f = \overline{A}BD\overline{E}\overline{F} + AB\overline{D}\overline{E}\overline{F} + A\overline{B}\overline{C}DF \text{ **Ans.**}$$

Example 4-120. *Using Quenie-Mclusky method and prime implicant reduction and realize the simplified expression using only NAND gates*

$$f(w, x, y, z) = \pi M (0, 4, 5, 9) .d (1, 7, 13)$$

(VTU, July/August. 2004)

Solution: Given: The logic function

$$f(w, x, y, z) = \pi M (0, 4, 5, 9) .d(1, 7, 13)$$

The given function in SOP form can be express as,

$$f(w, x, y, z) = \Sigma m (2, 3, 6, 8, 10, 11, 12, 14, 15) .d(1, 7, 13)$$

All minterm in the binary form is shown in Table 4-50.

Table 4-50.

Minterm	*Variable*			
	w	*x*	*y*	*z*
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
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

The given minterms are arranged in sequence. Grouping the minterms according to the numbers of 1s, its shown in Table 4-51.

Table 4-51.

Group	*Minterm*	*Variable*			
		w	*x*	*y*	*z*
1	1	0	0	0	1
	2	0	0	1	0
	8	1	0	0	0
2	3	0	0	1	1
	6	0	1	1	0
	10	1	0	1	0
	12	1	1	0	0
3	7	0	1	1	1
	11	1	0	1	1
	13	1	1	0	1
	14	1	1	1	0
4	15	1	1	1	1

Matching the group of pair according to the only variable difference only. The group pair is shown in Table 4-52.

Table 4-52.

Group	*Minterm*	*Variable*			
		w	*x*	*y*	*z*
1	1, 3	0	0	-	1
	2, 6	0	-	1	0
	2, 10	-	0	1	0
	8, 10	1	0	-	0
	8, 12	1	-	0	0
2	3, 7	0	-	1	1
	3, 11	-	0	1	1
	6, 7	0	1	1	-
	6, 14	-	1	1	-
	10, 11	1	0	1	-
	10, 14	1	-	1	0
	12, 13	1	1	0	-
	12, 14	1	1	-	1
3	7, 15	-	1	1	1
	11, 15	1	-	1	1
	13, 15	1	1	-	1
	14, 15	1	1	1	-

The third round of matching is shown in Table 4-53. Each term in the first group is compared with every them in the next group.

Table 4-53.

Group	*Minterm*	*Variable*			
		w	*x*	*y*	*z*
1	2, 6, 3, 7	0	-	1	-
	2, 6, 10, 14	-	-	1	0
	2, 10, 3, 11	-	0	1	-
	8, 10, 12, 14	1	-	-	0
2	3, 7, 11, 15	-	-	1	1
	6, 7, 14, 15	-	1	1	-
	10, 11, 14, 15	1	-	1	-
	12, 13, 14, 15	1	1	-	-

The third round of matching is shown in Table 4-54. Each term in the first group is compared with every term in the next group.

Table 4-54.

Minterm	*Variable*			
	w	*x*	*y*	*z*
2, 6, 3, 7, 10, 11, 14, 15	-	-	1	-
8, 10, 12, 14	1	-	-	0
12, 13, 14, 15	1	1	-	-

The next table is what is known as the prime implicant Table. The prime implicants table contains all the original terms in different columns and all the prime implicants recorded in different rows as shown in Table 4-55 below:

Table 4-55.

Terms	***Prime***	***Minterm***											
	implicants	***1***	***2***	***3***	***6***	***7***	***8***	***10***	***11***	***12***	***13***	***14***	***15***
Y	2, 6, 3, 7, 10, 11, 14, 15		✓	✓	✓	✓		✓	✓			✓	✓
$w\overline{z}$	8, 10, 12, 14						✓	✓		✓		✓	
wx	12, 13, 14, 15									✓	✓	✓	✓

Now we find the columns that contain a single tick (✓) and circle in then, then place an asterisk to the left of those rows in which you have circled a circle. This rows marked with an asterisk are the essential prime implicants as shown in Table 4-56.

Table 4-56.

Terms	***Prime***	***Minterm***											
	implicants	*1*	*2*	*3*	*6*	*7*	*8*	*10*	*11*	*12*	*13*	*14*	*15*
Y	2, 6, 3, 7, 10, 11, 14, 15*		(✓)	(✓)	(✓)	(✓)		(✓)	(✓)			(✓)	(✓)
$w\overline{z}$	8, 10, 12, 14*						(✓)	(✓)		(✓)		(✓)	
wx	12, 13, 14, 15									✓	✓	✓	✓

Thus the resultant function.

$$f(w, x, y, z) = y + w\overline{z} \quad \textbf{Ans.}$$

4-42. Map Entered Variable (MEV)

Karnaugh map is a manual technique for Boolean equation simplification. It is convergent when the map size is limited to 5 to 6 variables. A the number of variables increases it is difficult to make judgments about which combinations form the minimum expression. The map Entered Variable (MEV) is a technique which increases the effective size of a karnaugh map, allowing a smaller map to handle a greater number of variables

In normal karnaugh map each square (cell) represents a minterm, a maxterm or a don't care term. The MEV map permit a cell to contain a single variable or a complement of it, in addition to the 1s, 0s and don't care terms. Consider the following two variable k-map simplification of a three variable problem Let,

$$f(A, B, C) = \Sigma m(0, 1, 4, 7)$$

The truth table for the expression is shown in Table 4-57.

Table 4-57

Minterms in Decimal	*Minterms in Binary*			
	A	*B*	*C (MEV)*	*f*
0	0	0	0	1
0				
1	0	0	1	1
2	0	1	0	0
1				
3	0	1	1	0
4	1	0	0	1
2				
5	1	0	1	0
6	1	1	0	0
3				
7	1	1	1	1

The above truth table also shows the mapping of minterms. In a normal case, we have to use 3-variable k- map to simplify the Boolean expression secified by the above truth table. However the mapping of minterms allows us to simplify the same using 2- variable k-map. Let us study the rules for entering the mapped variable in the k-map. Rules are given below:

1. **Rule 1:** If function equals 0 for both values of MEV, then we enter 0 in appropriate cell of MEV map.

Value of MEV	*f*	*Entry in VEM map*
0	0	0
1	0	

2. **Rule 2:** If function follows the values of MEV, then we enter V(variable) in appropriate cell of MEV map.

Value of MEV	*f*	*Entry in VEM map*
0	0	V
1	1	

3. **Rule 3:** If function values are complement of MEV, then we enter $\overline{V}$ (complement of *V*) in appropriate cell of MEV map.

Value of MEV	*f*	*Entry in VEM map*
0	1	$\overline{V}$
1	0	

4. **Rule 4:** If function equals 1 for values of MEV, then we enter 1 in appropriate cell of MEV map.

Value of MEV	*f*	*Entry in VEM map*
0	1	1
1	1	

5. **Rule 5:** If $f = X$ (don't care)for MEV = 0 and $f = 0$ for MEV = 1 or vice-versa, enter 0 in the appropriate cell of MEV map.

Value of MEV	*f*	*Entry in VEM map*
0	X	0
1	0	
0	0	0
1	X	

6. **Rule 6:** If f = x (don't care) for MEV = 0 and f = 1 for MEV = 1 or vice-verse, enter 1 in the appropriate cell of MEV map.

Value of MEV	*f*	*Entry in VEM map*
0	X	1
1	1	
0	1	1
1	X	

7. **Rule 7:** If $f = X$ for both the values of MEV, enter X (don't care) in appropriate cell of MEV map.

Value of MEV	*F*	*Entry in VEM map*
0	X	X
1	X	

Using the rule we can represent the given equation in 2-variable karnaugh map as shown in Fig. 4-148. Here map entered variable is C.

Decimal Number	*Inputs*			*Output Function F*	*Rule*
	A	*B*	*C (MEV)*		
0	0	0	0	1	Rule 4
1	0	0	1	1	
2	0	1	0	0	Rule 1
3	0	1	1	0	
4	1	0	0	1	Rule 3
5	1	0	1	0	
6	1	1	0	0	Rule 2
7	1	1	1	1	

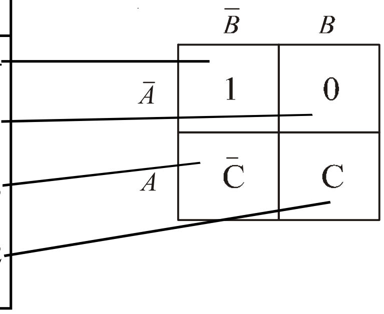


Fig. 4-148.

Grouping the squares of MEV k-map is as follows

1. Group all the squares that have 1s, same as in k-map
2. Group MEV Cell along with 1s and don't cares
3. Group same MEV variables
4. In case of MEV cell 'AND' MEV variable with map variables

Using the above rule we solve the function $f = (A, B, C) = \sum m\ (0, 1, 4, 7)$ with MEV K-map

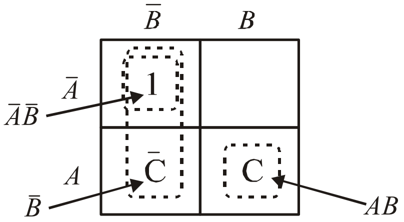


Fig. 4-149.

To simplify the function group the squares as known in Fig. 4-149. A group of one squares in the upper left of the karnaugh map produce the term $\overline{A}\overline{B}$. The group of one square in the lower right of the karnaugh map produce map variables *AB*, this map variable is AND with the MEV variable *C*. The final output term produces *ABC*.

Similarly another group of two squares in the first column produce the map variables B, this map variable is AND with the MEV variable $\overline{C}$, this square produce the term $B\overline{C}$. Thus the resulting simplified expression,

$$f(A, B, C) = \overline{A}\overline{B} + ABC + B\overline{C}$$

Example 4-121. *Write the map entered variable k-map for the Boolean function*

$$f(A, B, C, D) = \Sigma m(2, 9, 10, 11, 13, 14, 15)$$

(VTU, Jan/Feb 2006)

Solution. The given function is

$$f(A, B, C, D) = \Sigma m(2, 9, 10, 11, 13, 14, 15)$$

To solve this function with MEV map method, we have to take 1 variable as a MEV variable, let take the D variable as MEV variable. The truth table is shown in Table 4-58. The MEV map entry is done according to the rule.

Table 4-58.

Minterm in Decimal Number		***Input values***				***Output Function***	***MEV map Entry***
		A	***B***	***C***	***D (MEV)***		
0	0	0	0	0	0	0	0
	1	0	0	0	1	0	
1	2	0	0	1	0	1	$\bar{D}$
	3	0	0	1	1	0	
2	4	0	1	0	0	0	0
	5	0	1	0	1	0	
3	6	0	1	1	0	0	0
	7	0	1	1	1	0	
4	8	1	0	0	0	0	D
	9	1	0	0	1	1	
5	10	1	0	1	0	1	1
	11	1	0	1	1	1	
6	12	1	1	0	0	0	D
	13	1	1	0	1	1	
7	14	1	1	1	0	1	1
	15	1	1	1	1	1	

The MEV K-map entry is shown in Fig. 4-150. To simplify the function group the squares as shown in Fig. 4-150.

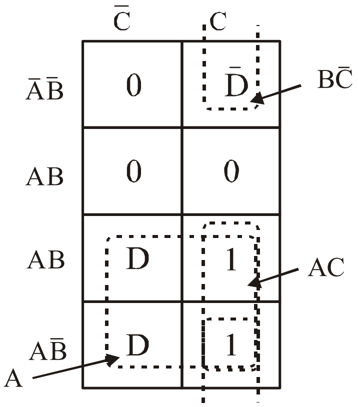


Fig. 4-150.

The group of two squares in the second column of the map produces the term AC. Another group of two squares in the second row of the map produce variable term $B\overline{C}$, this map variable is AND with the MEV variable of the same group, MEV variable is $\overline{D}$, this group produces the term $B\overline{C}\ \overline{D}$.

Similarly another group of four squares in the lower part of the map produce variable term A this map variable is AND with the MEV variable of the same group, MEV variable is D, this group produces the term AD. Thus the resulting simplified expression,

$$f(A, B, C, D) = AC + B\overline{C}\ \overline{D} + AD \textbf{ Ans.}$$

Example 4-122. *Simplify the function with two variable MEV K-map method.*

$$f(a, b, c, d) = \Sigma(2, 3, 4, 5, 13, 15) + d(8, 9, 10, 11) \qquad \textit{(VTU Jan/Feb 2004)}$$

Solution. The given function is

$$f(a, b, c, d) = \Sigma(2, 3, 4, 5, 13, 15) + d\ (8, 9, 10, 11)$$

The truth table for the given function is shown in Table 4-59. Here map entered variable is d.

Table 4-59.

Minterms in Decimal		*a*	*b*	*c*	*d*	*f*
0	0	0	0	0	0	0
	1	0	0	0	1	0
1	2	0	0	1	0	1
	2	0	0	1	1	1
2	4	0	1	0	0	1
	5	0	1	0	1	1
3	6	0	1	1	0	0
	7	0	1	1	1	0
4	8	1	0	0	0	X
	9	1	0	0	1	X
5	10	1	0	1	0	X
	11	1	0	1	1	X
6	12	1	1	0	0	0
	13	1	1	0	1	1
7	14	1	1	1	0	0
	15	1	1	1	1	1

Using the rules we map the function in three variables MEV k-map. This is shown in Fig. 4-151.

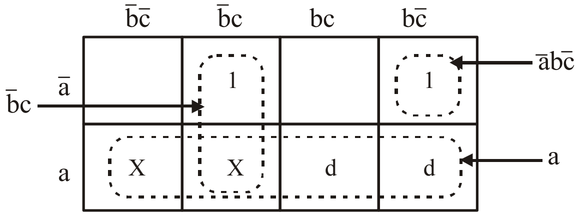


Fig. 4-151.

To simplify the function group the squares as shown in Fig. 4-151. A group of one squares in the fourth column of the karnaugh map produces the term $\bar{a}b\bar{c}$. The group of two squares in the second column of the karnaugh map produce map variable $\bar{b}c$.

Another group of four squares in the second row of the map produce variable term *'a'*, This map variable is AND with the MEV variable of the same group, MEV variable is *'d',* this group produces the term *ad.* Thus the resulting simplified expression.

$$f\,(a,\, b,\, c,\, d) = \bar{a}b\bar{c} + \bar{b}c + ad.$$

Let us simplify this function using two variable MEV map. Here we treat variable *c* and d as a map entered variable. The truth table is shown in Table 4-60. In 0 minterm MEV variable '*c*' is same as the output function so we enter '*c*' in the MEV map. In 1 minterm MEV variable '*c*' is complement of the output function so we enter '$\bar{c}$' in the MEV map.

In 2 minterm the output function is 'x' for all value of MEV variable, so we enter '*x*' in the MEV map. In minterm 3 the output function is same as the MEV variable '*d*' so we enter '*d*' in the MEV map.

Table 4-60.

Minterm in Decimals		***a***	***b***	***c***	***d***	***f***	***MEV map entry***
0	0	0	0	0	0	0	c
	1	0	0	0	1	0	
	2	0	0	1	0	1	
	3	0	0	1	1	1	
1	4	0	1	0	0	1	$\bar{c}$
	5	0	1	0	1	1	
	6	0	1	1	0	0	
	7	0	1	1	1	0	
2	8	1	0	0	0	X	X
	9	1	0	0	1	X	
	10	1	0	1	0	X	
	11	1	0	1	1	X	
3	12	1	1	0	0	0	*d*
	13	1	1	0	1	1	
	14	1	1	1	0	0	
	15	1	1	1	1	1	

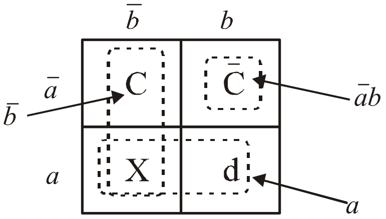


Fig. 4-152.

To simplify the function group the squares as shown in Fig. 4-152. A group of one square in the second column of the karnaugh map produce the variable term $\bar{a}b$, this map variable is AND with the MEV variable of the same group, the MEV variable is $\bar{c}$. This group produces the term $\bar{a}b\bar{c}$.

Another group of two squares in the second row of the map produce variable term '*a'*, this map variable is AND with the MEV variable of the same group, the MEV variable is '*d*', this group produces the term *ad*.

Similarly another group of two squares in the first column of the map produce variable term '$\bar{b}$' this map variable is AND with the MEV variable of the same group, the MEV variable is '*c*', this group produces the term $\bar{b}c$. Thus the resulting simplified expression.

$$f(a, b, c, d) = \bar{a}b\bar{c} + \bar{b}c + ad \quad \textbf{Ans.}$$

SUMMARY

In this chapter, you have learnt that,

1. There are several Boolean laws and rules that provide the means to form equivalent logic circuits and/or simplification of logic circuits.
2. De Morgan's theorems are used in logic simplification whenever the inversion bars cover more than one variable in the original Boolean expression. The two theorems are :

$$\overline{X+Y} = \bar{X}\bar{Y}$$

and $\overline{XY} = \bar{X} + \bar{Y}$

3. The Sum-of-Products (SOP) and Product-of-Sums (POS) are two general forms for the logic expressions.
4. The Karnaugh map is a graphical method for representing a logic circuit's truth table and generating a simplified expression for the circuit output.

GLOSSARY

Adjacency. Characteristic of cells in the Karnaugh map in which there is a single variable change from one cell to another cell next to it any of its four sides.

Associative Law. In addition (ORing) and multiplication (ANDing) of three or more variables, the order in which the variables are grouped makes no difference.

Cell. An area on the Karnaugh that represents a unique combination of variables in a product term.

Commutative law. In addition (ORing) and multiplication (ANDing) of two variables, the order in which the variables are ORed and ANDed makes no difference.

Distributive law. ORing several variables and then ANDing the result with the single variable is equivalent to ANDing the single variable with each of the several variables and then ORing the products.

Domain. All the Boolean variables in any given Boolean equation or an expression.

Don't care. A combination of input literals (or variables) that cannot occur and can be used as a 1 or a 0 on a Karnaugh map.

Karnaugh map (or K-map). An arrangement of cells representing the combination of literals (or variables) in a Boolean expression and used for a systematic simplicfication of the expression.

Literal. A variable or a complement of a variable.

Product-of-sums (POS). A Boolean expression that is basically the ANDing of ORed terms.

Sum-of-products (SOP). A Boolean expression that is basically the ORing of ANDed terms.

Sum term. The Boolean sum of two or more literals or product terms) equivalent to an OR operation.

Variable. A symbol used to represent a logical quantity that can have a value of 1 or 0. It is usually designated by an italic letter *e.g. A, B, C* and *D.*

DESCRIPTIVE QUESTIONS

1. Define each of the following terms :

   (*a*) Karnaugh map (*b*) Sum-of-Products form

   (*c*) Product-of-Sums form (*d*) Don't care state.

2. Use Boolean algebra to prove than Exclusive NOR (XNOR) output expression is the exact inverse of the XOR output expression,
3. Why is De Morgan's theorem important in the simplification of Boolean expressions?
4. Use De Morgan's theorem to prove that a NOR gate with inverted inputs is equivalent to an AND gate.
5. Explain briefly how De Morgan's theorem is applied in the simplification of Boolean expression.
6. Sketch a 3-variable Karnaugh map and label each cell according to its binary value.
7. Sketch a 4-variable Karnaugh map and label each cell according to its binary value.
8. What do you understand by min terms and max terms? Explain briefly with suitable truth table.

   [**Hint :** minterm is the same as product term and max term is the same as sum term]

9. What do you understand by the term "Canonical form".?

   [**Hint :** The Boolean function expressed as a sum of min terms or product of max. terms are said to be in canonical form]

10. With the help of truth tables prove De Morgan's theorem for 3 variables. What is meant by don't care condition?

    (*Gauhati University, 2006*)

11. What are the theorems of Boolean algebra?

    (*Mahatma Gandhi University, Jan. 2007*)

12. What is meant by standard SOP and standard POS forms of logical expression?

    (*Gauhati University, 2006*)

13. Mention two categories of Boolean expression based on their structure. Write these forms for any give three variable functions.

(*VTU, Jan./Feb. 2004*)

14. Explain Demorgan's therorems in Boolean Algebra.

(*VTU, Jul./Aug. 2005*)

15. Explain Demorgan's theorem.

(*Anna University, Nov./Dec. 2006*)

16. List out any four basic rules that are used in Boolean algebra expression.

(*Anna University, Nov./Dec. 2006*)

17. Explain the basic laws of Boolean algebra with sample.

(*Anna University, Nov./Dec. 2006*)

18. Mention any two application of Demorgan's theorem.

(*Anna University, May./Jun. 2006*)

19. Explain the fundamental rules used in Boolean expression.

(*Anna University, May./Jun. 2006*)

20. Elaborate the basic laws of Boolean algebra with example.

(*Anna University, May./Jun. 2006*)

21. What code is used to label the row headings and column headings of k map and why?

(*VTU, Jan./Feb. 2004*)

22. Mention the different methods available for manipulating Boolean formulas. Explain ant three in details.

(*VTU, Jan./Feb. 2006*)

23. What are don't care condition? What are its advantages?

(*VTU, Jul. 2006*)

24. Write the steps for simplifying a logic expression using a karnaugh map.

(*Anna University, May./Jun. 2006*)

25. Explain the De Morgan's theorem.

(*Gujarat Technological University, Dec. 2009*)

26. Explain the De Morgan's theorem with suitable example.

(*Nagpur University, 2004*)

27. Draw a five variable karnaugh map.

(*Jamia Millia Islamia University, 2007*)

28. Using appropriate example, explain

1. Associative law
2. Distributive law

(*Jamia Millia Islamia University, 2007*)

29. What do you mean by standard SOP, and standard POS term?

(*Nagpur University, 2004*)

TUTORIAL PROBLEMS

1. Using the trugh table, show that the Boolean expression,

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

(*c*) $\overline{A} + B = \overline{A}B$ (*d*) $A\left(\overline{A} + \overline{B}\right) = AB$

2. Simplify the following Boolean expressions to a minimum number of literals :

(*a*) $\bar{X}\bar{Y} + XY + \bar{X}Y$ (*b*) $\bar{X}Y + X\bar{Y} + XY + \bar{X}\bar{Y}$ (**Ans.** (*a*) $\bar{X} + Y$; (*b*) 1)

3. Using the Boolean laws and rules, simplify the following expressions :

(*a*) $Z = (\bar{A} + B)(A + \bar{B})$ (*b*) $Y = A\bar{C} + AB\bar{C}$

(*c*) $Y = \bar{A}\bar{B}\,C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$ (*b*) $Y = \bar{A}D + ABD$

(**Ans.** (*a*) AB, (*b*) $A\bar{C}$, (*c*) $\bar{A}\bar{B}\bar{D}$ (*d*) $(\bar{A} + B)D$)

4. Simplify the following Boolean function using Boolean algebra :

$$X = A + \bar{A}B + \bar{A}\bar{B}$$ (**Ans.** 1)

5. (*a*) Write a logic expression for the output *y* in terms of logic variables *A*, *B* and *C* from the truth table shown in Fig. 4-153.

(*b*) Give a suitable circuit for realizing *Y* with a minimum number of logic gates.

A	*B*	*Y*
0	0	$\bar{C}$
0	1	C
1	0	1
1	1	0

Fig. 4-153.

6. Simplify the Boolean expressions :

(*a*) $X = ABC + AB\bar{C} + A\bar{B}C$

(*b*) $X = (A + B + AB)(A + C)$

(**Ans.** (*a*) $A\,(B + C)$, (*b*) $A + BC$)

7. Prove the following identity in the in the truth table :

$$XY + YZ + Y\bar{Z} = XZ + Y\bar{Z}$$

8. Simplify the following Boolean equations using rules of Boolean algebra :

$$x = (A + \bar{B}\bar{C})(\overline{B + C})$$ (**Ans.** $\bar{B}\bar{C}$)

9. Design a logic circuit to give an output,

$$x = (\overline{AB} + \bar{A}C)(\overline{AC + C})$$

10. Simplify the following Boolean expressions :

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

(*b*) $y = AB\bar{C}\bar{D} + AB\bar{C}D + ABC + ABD + CD$

11. Determine by means of truth table, the validity of De Morgan's theorem for three variables :

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

12. Simplify the following Boolean expression using De Morgan's theorem :

(*a*) $z = \overline{A + \bar{B}C}$ (*b*) $y = \overline{(A + BC)(D + EF)}$

(**Ans.** (*a*) $\bar{A}(B + \bar{C})$; (*b*) $\bar{A}\bar{B} + \bar{A}\bar{C} + \bar{D}\bar{E} + \bar{D}\bar{F}$)

13. Use De Morgan's theorems to simplify the following Boolean expressions :

(*a*) $y = \overline{R\bar{S}T + \bar{Q}}$ (*b*) $y = \overline{A + \bar{B} + \bar{C}D}$

(**Ans.** (*a*) $(\bar{R} + S + \bar{T})Q$; (*b*) $y = \bar{A}B(C + \bar{D})$)

14. Determine the Boolean expression for the circuit shown in Fig. 4-154. Use De Morgan's theorem and then Boolean algebra rules to simplify the equation. Draw the Simplified circuit.

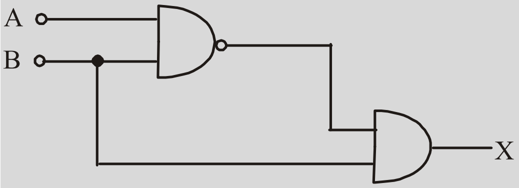


Fig. 4-154.

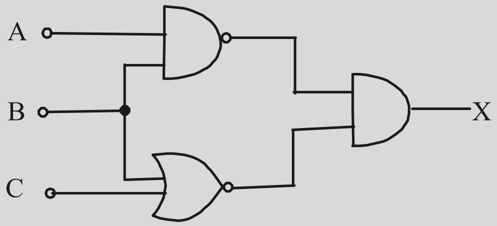


Fig. 4-155.

15. Repeat Problem 6 for the circuit shown in Fig. 4-155. (**Ans.** $\overline{B}+\overline{C}$)

16. Repeat Problem 6 for the circuit shown in Fig. 4-156. $\left(\textbf{Ans. } \overline{A}+\overline{B}+C+D\right)$

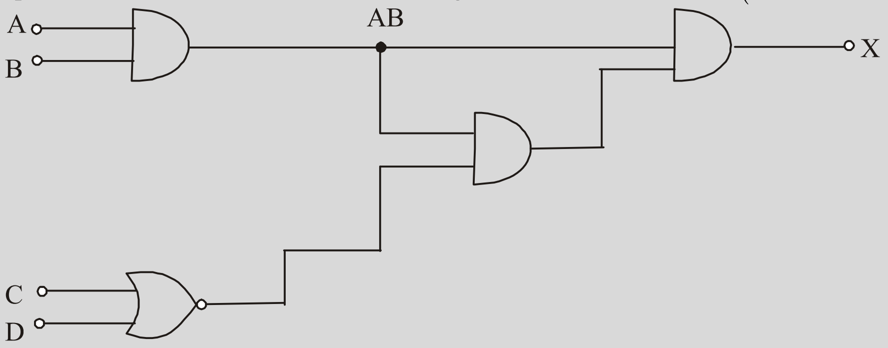


Fig. 4-156.

17. Determine the Boolean expression for the logic circuit shown in Fig. 4-157. Simplify the Boolean expression using Boolean Laws and De Morgan's theorem. Redraw the logic circuit using the simplified Boolean expression.

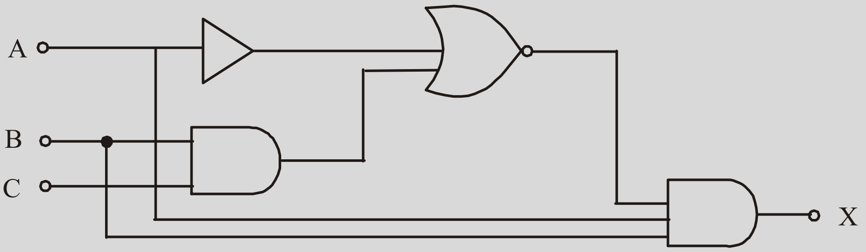


Fig. 4-157.

18. Determine the output, X of a logic circuit shown in Fig. 4-158. Simplify the output expression using Boolean Laws and theorems. Redraw the logic circuit with the simplified expression.

$\left(\textbf{Ans. } \overline{B}\overline{C}D+AC\right)$

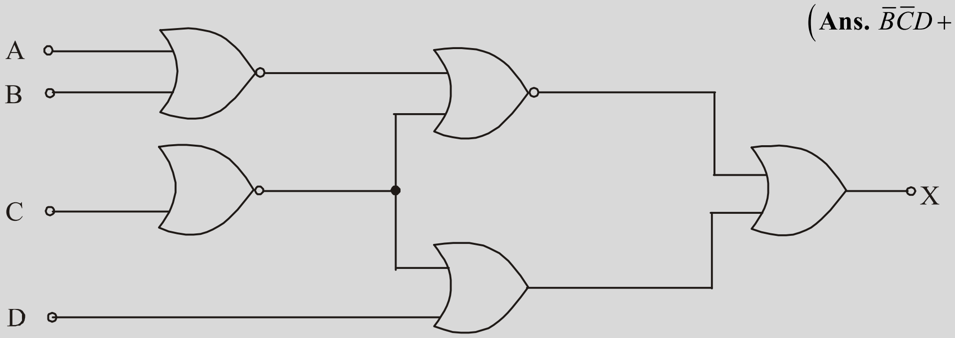


Fig. 4-158.

19. Consider the logic circuit shown in Fig. 4-159. Determine the Boolean expression at the circuit output X, simplify it. Using the simplified expression, redraw the logic circuit.

$\left(\textbf{Ans. } A\left(\bar{B}+C\right)\right)$

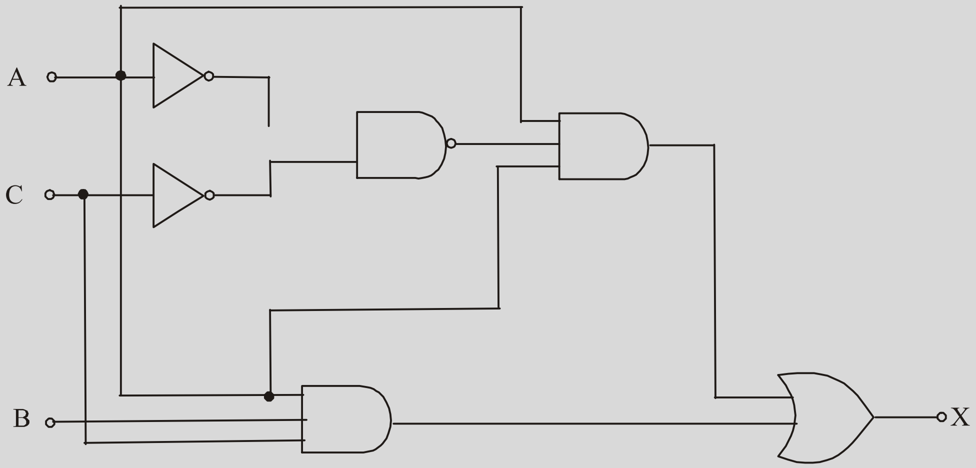


Fig. 4-159.

20. Given the following Boolean expression :

$$f = x\bar{y}z + \bar{x}\bar{y}z + \bar{w}xy + w\bar{x}y + wxy$$

(*a*) Obtain the truth table of the fuction.

(*b*) Draw the logic diagram using original Boolean expression.

21. Express the Boolean function, $f = A + \bar{B}C$ in sum-of min-terms.

22. Express the Boolean function, $f = xy + \bar{x}z$ in product-of max-terms.

23. Express each function in sum of-product terms and product of-sum terms :

(*a*) $\left(AB+C\right)\left(B+AC\right)$ (*b*) $\left(\bar{A}+B\right)\left(\bar{B}+C\right)$

24. Express the Boolean function :

$f\left(A,B,C\right)=\Sigma\left(0,1,2,5,8,9,10\right)$ in product-of-sums form.

25. Express the following in sum-of min terms and product of max terms :

(*a*) $f\left(A,B,C,D\right)=\Sigma\left(0,2,6,11,13,14\right)$

(*b*) $f\left(x,y,z\right)=\Pi\left(0,3,6,7\right)$

$\{$**Ans.** (*a*) sum-of min terms $\bar{A}\bar{B}\bar{C}\bar{D}+\bar{A}\bar{B}C\bar{D}+\bar{A}BC\bar{D}+A\bar{B}CD+A\bar{B}CD+ABC\bar{D}$,

Product-of max-terms : $\left(\bar{A}+B+C+D\right)\left(A+\bar{B}+\bar{C}+\bar{D}\right)\left(A+\bar{B}+\bar{C}+D\right)$

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

(*b*) Sum-of min-terms = $\bar{X}\bar{Y}Z+\bar{X}Y\bar{Z}+X\bar{Y}\bar{Z}+X\bar{Y}Z$

Product of max-terms :

$\left(\bar{X}+\bar{Y}+\bar{Z}\right)\left(\bar{X}+\bar{Y}+Z\right)\left(\bar{X}+Y+Z\right)\left(X+Y+Z\right)\}$

26. Fig. 4-160 shows a three-variable Karnaugh map. Group the 1*s* and hence obtain the minimized Boolean expresion $\left(\textbf{Ans. } AB+BC+\bar{A}\bar{B}\bar{C}\right)$

	$\bar{C}$	C
$\bar{A}\,\bar{B}$	1	
$\bar{A}\,B$		1
$A\,B$	1	1
$A\,\bar{B}$		

Fig. 4-160.

	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$C\,\bar{D}$
$\bar{A}\,\bar{B}$	1	1		
$\bar{A}\,B$	1	1	1	1
$A\,B$				
$A\,\bar{B}$		1	1	

Fig. 4-161.

27. Fig. 4-161 shows a four-variable Kamaugh-map. Group the 1*s* and hence obtain the minimized Boolean expression. $\left(\textbf{Ans. } \bar{A}B + \bar{A}\bar{C} + A\bar{B}D\right)$

28. A truth table has output 1*s* for these inputs : $ABCD = 0011$, $ABCD = 0110$, $ABCD = 1000$ and $ABCD = 1100$, and 0*s* for the other inputs. Draw the Karnaugh map and find the simplified Boolean equation for the truth table. $\left(\textbf{Ans. } A\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}BC\bar{D}\right)$

29. Determine the minimized expression for the expression for the Karnaugh map shown in Fig. 4-162.

$\left(\textbf{Ans. } B\bar{C} + BD + \bar{B}C\right)$

	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$C\,\bar{D}$
$\bar{A}\,\bar{B}$	0	0	1	1
$\bar{A}\,B$	1	1	1	0
$A\,B$	1	X	X	X
$A\,\bar{B}$	0	0	X	X

Fig. 4-162.

	$\bar{C}\,\bar{D}$	$\bar{C}\,D$	$C\,D$	$C\,\bar{D}$
$\bar{A}\,\bar{B}$	1	1		1
$\bar{A}\,B$	1	1		1
$A\,B$				1
$A\,\bar{B}$				

Fig. 4-163.

30. Determine the simplified Boolean expression for the Karnaugh map shown in Fig. 4-163.
(*UPSC Engg. Services 1995*)

$\left(\textbf{Ans. } \bar{A}\bar{C} + \bar{A}\bar{D} + ABC\right)$

31. Simplify the following Boolean functions using Karnaugh map :

(*a*) $f(A,B,C) = \Sigma(0,2,3,4,6)$ $\left(\textbf{Ans. } \bar{C} + \bar{A}B\right)$

(*b*) $f(A,B,C,D) = \Sigma(0,1,2,4,5,7,11,15)$ $\left(\textbf{Ans. } \bar{A}\bar{C} + \bar{A}\bar{B}\bar{D} + \bar{A}BD + ACD\right)$

32. Simplify the Boolean function :

$$f(w,x,y,z)=\Sigma(0,1,2,4,5,6,8,9,12,13,14)$$

and draw the logic diagram for simplified Boolean expression.

$\left(\textbf{Ans. } \overline{C}+\overline{A}\overline{D}+B\overline{D}\right)$

33. Minimize the following function given below using the Karnaugh map :

$$f(A,B,C,D)=\Sigma(0,1,2,3,5,6,8,10,15)$$

and realize the function using NAND gates.

$\left(\textbf{Ans. } \overline{A}\overline{B}+\overline{B}\overline{D}+\overline{A}C\overline{D}+ABCD\right)$

34. Simplify the following Boolean function in product-of-sums form by means of four variable map. Draw the logic diagram using gates :

$$f(A,B,C,D)=\Sigma(2,3,4,5,6,7,11,14,15)$$

$\left(\textbf{Ans. } \overline{A}B+CD+\overline{A}C+BC\right)$

35. Given the logical function of three variables :

$$f(A,B,C)=\left(A+\overline{B}+C\right)\left(\overline{A}+B+C\right)$$

Express f in the standard sum-of-products form.

$\left(\textbf{Ans. } \overline{A}\overline{B}C+\overline{A}\overline{B}C+\overline{A}B\overline{C}+A\overline{B}\overline{C}+AB\overline{C}+ABC\right)$

36. Simplify the following using the Karnaugh map :

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

$\left(\textbf{Ans. } \overline{A}+\overline{B}C\right)$

37. Design a combinational circuit with three inputs *A*, *B*, *C* and three outputs *x*,*y*,z. When the binary input is 0, 1, 2 or 3, the binary output is one greater than the input. When the binary input is 4, 5, 6 or 7, the binary output is one less than the input.

38. Seven switches operate the lamp in the following way :- if switches 1, 3, 5 and 7 are closed and switch 2 is open, or if switches 2, 4 and 5 are closed and switch 3 is open, or if all seven switches are closed, the lamp will light. Use NOT, AND and OR gates to show how the switches must be connected.

39. Design a logic circuit that will allow a signal to pass the output only when control inputs *B* and *C* are both HIGH, otherwise the output will stay LOW.

40. Design a logic circuit with inputs *P*, *Q* and *R* so that the output *S* is high whenever *P* is *0* or whenever *Q* = *R* = 1.

MULTIPLE CHOICE QUESTIONS

1. Boolean algebra is essentially based on

(*a*) symbols (*b*) logic (*c*) truth (*d*) numbers

2. Different variables used in Boolean algebra can have values of

(*a*) 0 or 1 (*b*) LOW or HIGH (*c*) true or false (*d*) ON or OFF

3. According to Boolean algebra, $\left(A+\overline{A}\right)$ equals

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

4. According to Absorbitive law of Boolean algebra, $(A+AB)$ equals,

(*a*) A (*b*) B (*c*) AB (*d*) A

5. When we De Morganise $\overline{A}\overline{B}$, we get,

(*a*) $\overline{AB}$ (*b*) $\overline{A}+\overline{B}$ (*c*) $\overline{A+B}$ (*d*) $A+B$

6. The expression $\overline{ABC}$ can be simplified to

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

7. According to Boolean Algebra

8. While obtaining minimal sum-of-products expression,

(*a*) all don't cares are ignored

(*b*) all don't cares are treated as logic 1*s*

(*c*) all don't cares are treated as logic 0*s*.

(*d*) only such don't cares that help minimization are treated as logic 1*s*.

9. which of the following Boolean algebra rules is correct

(*a*) $A.\overline{A}=1$ (*b*) $A+A.B=A+B$ (*c*) $A+\overline{A}B=A+B$ (*d*) $A(A+B)=B$

(*AMIE., Summer 2002*)

10. Simplified form of Boolean expression $\left(A+\overline{B}+\overline{A}B\right)C$ is

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

(*AMIE., Summer 2002*)

11. Which of the following is not true?

(*a*) $A.A=A$ (*b*) $A.\overline{A}=0$ (*c*) $A+1=1$ (*d*) $A+\overline{A}=0$

(*AMIE., Winter 2001*)

12. The minimum number of NAND gates required to inplement the Boolean function $A+\overline{A}B+A\overline{B}C$ is

(*a*) three (*b*) two (*c*) one (*d*) zero

(*AMIE., Winter 2001*)

13. The Boolean expression $\left(\overline{A}+B\right)\left(A+B\right)$ when simplified yields to

(*a*) A (*b*) B (*c*) $\overline{A}$ (*d*) $\overline{B}$

(*AMIE., Winter 2001*)

14. The Boolean expression for the shaded area in the Venn diagram is shown in Fig. 4-164

(*a*) $XY+YZ+XZ$

(*b*) $XY\overline{Z}+\overline{X}YZ+X\overline{Y}Z$

(*c*) $XYZ+\overline{X}\,\overline{YZ}$

(*d*) $\overline{X}\,\overline{YZ}+X\overline{YZ}$

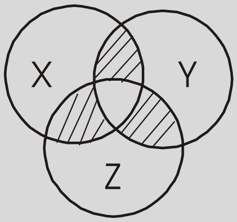


Fig. 4-164.

(*AMIE Winter 2001*)

15. $X+\overline{X}Y$ is reduced to

(*a*) X (*b*) $X+\overline{Y}$ (*c*) $X+Y$ (*d*) $\overline{X}+Y$

(*AMIE., Summer 2001*)

16. A four-variable Karnaugh map has

(*a*) 8 min-terms (*b*) 16 min-terms (*c*) 32 min-terms (*d*) 24 min-terms

(*AMIE., Winter 2000*)

17. Minimum number of literals in the expression, $ABC+AB+BC$ is

(*a*) 1 (*b*) 2 (*c*) 3 (*d*) 4

(*AMIE., Winter 2000*)

18. $\overline{ABCD}+B\overline{C}D+\overline{AC}+A$ is equivalent to

(*a*) 1 (*b*) $\overline{C}$ (*c*) $\overline{A}+C$ (*d*) $A+\overline{C}$

(*AMIE., Summer 2000*)

19. In Karnaugh map, the Gray reflected code is used to number

(*a*) only rows (*b*) only columns

(*c*) both rows and columns (*d*) the diagonal elements.

(*AMIE., Summer 2000*)

20. Boolean expression $\overline{xy}\,z+yz+xz$ can be reduced to

(*a*) $\overline{x}$ (*b*) z (*c*) y (*d*) $x+y$

(*AMIE., Summer 2000*)

21. The Karnaugh map for a Boolean function is shown in Fig. 4-165. The number of essential prime implicants for the function is

(*a*) 4 (*b*) 5 (*c*) 6 (*d*) 8

	$\overline{C}\,\overline{D}$	$\overline{C}\,D$	$C\,D$	$C\,\overline{D}$
$\overline{A}\,\overline{B}$	1	0	1	1
$\overline{A}\,B$	1	0	0	0
$A\,B$	1	0	0	0
$\overline{A}\,B$	0	1	0	1

Fig. 4-165.

(*AMIE, Summer 2000*)

22. In Boolean algebra, if $f=(A+B)(\overline{A}+C)$ then,

(*a*) $f=AB+\overline{A}C$ (*b*) $f=AB+\overline{AB}$

(*c*) $f=AC+\overline{A}B+BC$ (*d*) $f=AA+\overline{A}B$

(*UPSC Engg. Services 2001*)

23. Which one of the following is equivalent to the Boolean expression :

$y = \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{C}\,\overline{A}$

(*a*) $\overline{AB + BC + CA}$ (*b*) $(\overline{A} + \overline{B})(\overline{B} + \overline{C})(\overline{A} + \overline{C})$

(*c*) $\overline{(A + B)(B + C)(C + A)}$ (*d*) $(\overline{A + B})(\overline{B + C})(\overline{C + A})$

(*UPSC Engg. Services 2001*)

ANSWERS

1. (*b*)	**2.** (*a*)	**3.** (*b*)	**4.** (*a*)	**5.** (*d*)	**6.** (*d*)
7.	**8.** (*d*)	**9.** (*c*)	**10.** (*c*)	**11.** (*d*)	**12.** (*a*)
13. (*b*)	**14.** (*b*)	**15.** (*c*)	**16.** (*b*)	**17.** (*c*)	**18.** (*a*)
19. (*c*)	**20.** (*b*)	**21.** (*a*)	**22.** (*c*)	**23.** (*d*)	

5

FLIP - FLOPS AND RELATED DEVICES

OUTLINE

Objectives

Upon completion of this chapter, you will be able to :

- understand the operation of NAND and NOR gates and their use to construct basic latches
- understand the operation of edge-triggered flip-flops.
- describe the difference between S-C, D and J-K flip-flops.
- understand the significance of various flip-flop timing parameters specified by the manufacturers.
- apply flip-flops in basic applications.
- describe the difference between non-retriggerable and retriggerable one-shots
- Connect a 555 timer to operate as either an astable multivibrator or a one-shot.

5-1. Introduction

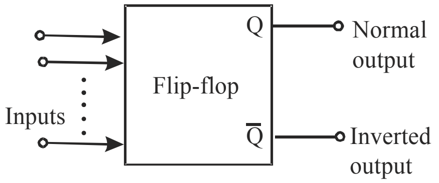
We have already discussed combinational logic circuits in the last chapter. The output logic levels of such circuits at any instant of time are dependent on the logic levels present at the inputs at that time. Any prior input-logic level conditions have no effect on the present outputs. This is due to the fact that combinational logic circuits have no *memory.* However, it will be interesting to know that most digital systems are made up of both combinational logic circuits and memory elements.

The most important memory element is the flip-flop (abbreviated as FF). The FF is made up of an assembly of logic gates. It may be noted that even though a logic gate, by itself, has no storage capability, but several such logic gates can be connected together in ways that permit information to be stored. Several different gate arrangements are used to produce these flip-flops.

The flip-flops are used extensively as a memory cell in static random access memory (SRAM) of a computer.

5-2. Flip-Flop (FF)

We have already discussed in the last article that flip-flop (abbreviated as FF) is made up of logic gates and it permits information to be stored in it. Fig. 5.1 (a) shows a general type of symbol used for a flip-flop.



(*a*) *Flip-flop symbol*

Output states

$Q = 1, \overline{Q} = 0$: It is called HIGH or 1State; also called SET state

$Q = 0, \overline{Q} = 1$: It is called LOW or 0 state; also called CLEAR orRESET state

(*b*) *Operating states of Flip-flop*

Fig. 5-1.

As seen from this diagram, the flip-flop has two outputs labeled as Q and $\overline{Q}$ that are inverse (or complement) of each other. The Q output is called the *normal* flip-flop output and $\overline{Q}$ is the *inverted* flip-flop output. It may be carefully noted that whenever we refer to the state of a flip-flop, we are referring to the state of its *normal* (Q) output. For instance if we say flip-flop is in the HIGH state, we mean that $Q = 1$. On the other hand if we say flip-flop is in LOW state, we mean that $Q = 0$. It is automatically understood that the $\overline{Q}$ state will always be *inverse* of Q, *i. e*., if $Q = 1$, $\overline{Q} = 0$, and if $Q = 0$, $\overline{Q} = 1$.

Fig. 5.1 (b) shows the two possible operating states of a flip-flop. One possible state is $Q = 1$, $\overline{Q} = 0$ and the other $Q = 0$, $\overline{Q} = 1$. The state $Q = 1$, $\overline{Q} = 0$ is called HIGH state or 1 state. It is also called *SET* state. Whenever, the inputs to a flip-flop cause it to go to the $Q = 1$ state, we call this *setting* the flip-flop or the flip-flop is set.

In a similar way, the state $Q = 0$, $\overline{Q} = 1$ is called LOW state or 0 state. It is also called *RESET* or CLEAR state. Whenever, the inputs to a flip-flop cause it to go to the $Q = 0$ state, we call this *resetting* or *clearing* the flip-flop. In the later part of the chapter we will see that many flip-flops will have a *SET* input and/or RESET (CLEAR) input that is used to drive the flip-flop into a specific output state.

It may be noted from the flip-flop symbol (Fig. 5.1 (a)) that a flip-flop can have one or more inputs. These inputs can be used to switch the flip-flop back and forth between its two possible output states.

It will be shown later that most flip-flop inputs need only to be momentarily activated (or pulsed) in order to cause a chage in the output state. The flip-flop output will remain in the new state even after the input pulse is over. This is flip-flop's memory characteristic.

It will be interesting to know that a flip-flop is also known as a latch and a bistable multivibrator. The term "*latch*" is used for certain types of flip-flops that will be described later. The term bistable multivibrator is the more technical name but flip-flop is the one used more frequently among the engineers and technologists.

5-3. Latch

A latch is the most basic type of flip-flop circuit. It can be constructed using NAND or NOR gates. Accordingly the latches are of two types :

1. NAND gate latch and
2. NOR gate latch

Both these types of latch are discussed one by one in the following pages.

5-4. NAND Gate Latch

Construction. Fig. 5.2 shows a latch constructed from NAND gates. It is called NAND gate latch or simply NAND latch. As seen from this diagram, the two NAND gates are cross-coupled. This means output of NAND-1 is connected to one of the inputs of NAND-2. Similarly, the output of NAND-2 is connected to one of the inputs of NAND-1. The NAND gate latch outputs are labeled as Q and $\overline{Q}$ respectively. Under normal conditions, these outputs will always be the inverse (or complement) of each other. The latch has two inputs namely SET and CLEAR. The SET input sets Q output of the latch to 1 state while the CLEAR input sets the Q output to the 0 state.

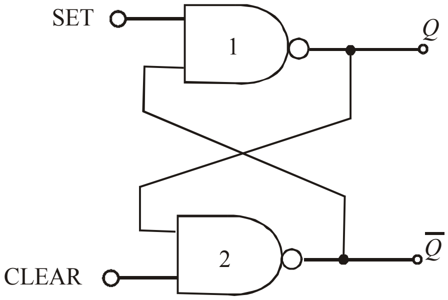
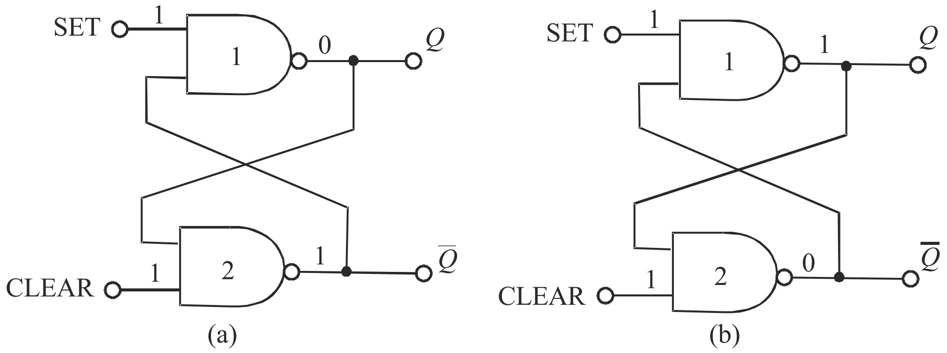


Fig. 5-2. *NAND gate latch*

Operation. Let us now understand the operation of NAND gate latch. Normally, the SET and CLEAR inputs of the latch are resting in the HIGH state. Whenever we want to change the latch outputs, one of the two inputs will be pulsed LOW. Let us begin our study by showing that there are two equally likely output states when SET = CLEAR = 1. One possibility is shown in Fig. 5.3 (a) where we have $Q = 0$ and $\overline{Q} = 1$. With $Q = 0$, the inputs to the NAND-2 are 0 and 1, which produces $\overline{Q} = 1$. The 1 from $\overline{Q}$ causes NAND-1 to have 1 at both inputs to produce a 0 output at Q. Thus we find that a LOW at the NAND-1 output produces a HIGH at NAND-2 output which in turn keeps the NAND-1 output LOW.



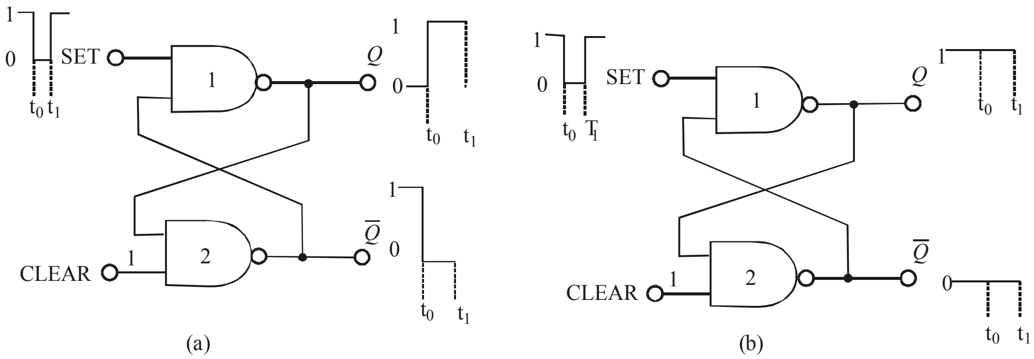
(a) (b)

Fig. 5-3. *Illustrating the two possible output states of NAND gate latch.*

Fig. 5.3 (b) shows the second possibility. Here $Q = 1$ and $\overline{Q} = 0$. As seen from this figure, the HIGH from NAND-1 produces a LOW at the NAND-2 output, which in turn keeps the NAND-1 output HIGH. Thus there are two possible output states when SET = CLEAR = 1. However which one of these two states exist will depend on what has occurred previously at the inputs. This is discussed below in more detail where we will take up three situations : (1) setting the latch, (2) clearing the latch and simultaneous setting and clearing the latch.

1. Setting the Latch. We have already mentioned above that if SET = CLEAR = 1, the output Q can be in two possible states i.e. $Q = 0$ or $Q = 1$. Now we will study as what happens when the SET input is momentarily pulsed LOW while CLEAR is kept HIGH.

Fig. 5.4 (a) shows what happens when $Q = 0$ prior to the occurrence of the pulse. As SET input is pulsed LOW at time 't_0', Q will go HIGH. This will force $\overline{Q}$ to go LOW so that NAND-1 has now two LOW-inputs. Thus when SET returns to 1 state at t_1, the NAND-1 output remains HIGH which in turn keeps the NAND-2 output LOW.

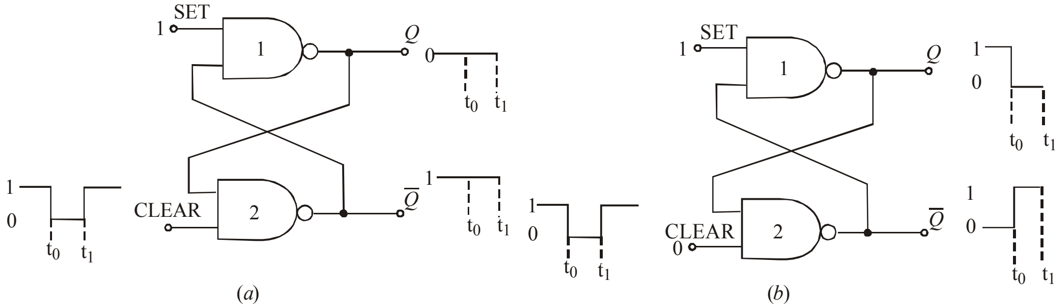


(a) (b)

Fig. 5-4. *Setting the NAND gate latch.*

Fig. 5.4 (b) shows what happens when $Q = 1$ and $\overline{Q} = 0$. Prior to the application of the SET pulse. Since $\overline{Q} = 0$ is already keeping the NAND-1 output HIGH, the LOW pulse at SET will not change anything. Thus when SET returns HIGH, the latch outputs are still in the $Q = 1$, $\overline{Q} = 0$ state (i.e no change in states).

2. Clearing the latch. Now we will study what happens when CLEAR input is pulsed LOW while SET is kept HIGH. Fig. 5.5 (a) shows the situation when $Q = 0$ and $\overline{Q}=1$ prior to the application of the pulse. Since $Q = 0$ is already keeping the NAND-2 output HIGH, the LOW pulse at CLEAR will not have any effect. When CLEAR returns HIGH, the latch outputs are still $Q = 0$ and $\overline{Q} = 1$.



(*a*) (*b*)

Fig 5-5. *Clearing the NAND gate latch.*

Fig 5.5 (b) shows the situation where $Q = 1$ prior to the occurrence of the CLEAR pulse. As CLEAR is pulsed LOW AT t_0, $\overline{Q}$ will go HIGH and this HIGH forces Q to go LOW because NAND-2 now has two LOW inputs. Thus when CLEAR returns HIGH at t_1, the NAND-2 output remains HIGH. This in turn keeps the NAND-1 output LOW.

3. Simultaneous setting and clearing. If the SET and CLEAR inputs are simultaneously pulsed LOW, this will produce HIGH levels at both NAND outputs so that $Q = \overline{Q} = 1$. This is an undesirable condition as the two outputs are supposed to be inverse of each other. Moreover, when the SET and CLEAR inputs return HIGH, the resulting output state will depend on which input returns HIGH first. Simultaneous transitions back to 1 state will produce unpredictable results. Because of these reasons the SET = CLEAR = 0 condition is normally not used for the NAND latch and is considered as invalid condition.

The operation of NAND gate latch may be summarized in the form of a truth table as shown in Fig 5.6. Each line of the truth table is described as below :

Inputs		***Output***
SET	***CLEAR***	
1	1	No change
0	1	$Q = 1$
1	0	$Q = 0$
0	0	$Q=\overline{Q}=1$ (Invalid)

Fig. 5-6. *Truth table for NAND gate latch*

1. **SET = CLEAR = 1** This condition is the normal resting state of the latch.The Q and $\overline{Q}$ outputs will remain in the same state in which they were prior to this input condition.

2. **SET = 0, CLEAR = 1** This condition will always cause the output to go to $Q = 1$ state where it will remain even after SET returns HIGH. This is called setting the latch.
3. **SET = 1, CLEAR = 0** This condition will always produce $Q = 0$. The output will remain in this state even after CLEAR returns HIGH. This is called clearing or resetting the latch.
4. **SET = CLEAR = 0** This condition tries to set and clear the latch simultaneously. It produces invalid results and should not be used.

5-5. Alternative Representations of NAND Gate Latch

We have already discussed in the last article about the NAND gate latch operation. It was mentioned that both SET and CLEAR inputs are active-LOW. Further the SET input will set $Q = 1$ when SET goes LOW. On the other hand, the CLEAR input will clear $Q = 0$ when CLEAR goes LOW. Because of this reason NAND gate latch is often drawn using the alternative representation for each NAND gate as shown in Fig 5.7 (a). (Recall from Art 3-41 that a NAND gate is equivalent to a bubbled or gate). The bubbles on the inputs of OR gate as well as labelling of the signals as $\overline{\text{SET}}$ and $\overline{\text{CLEAR}}$ indicate the active-LOW status of these inputs.

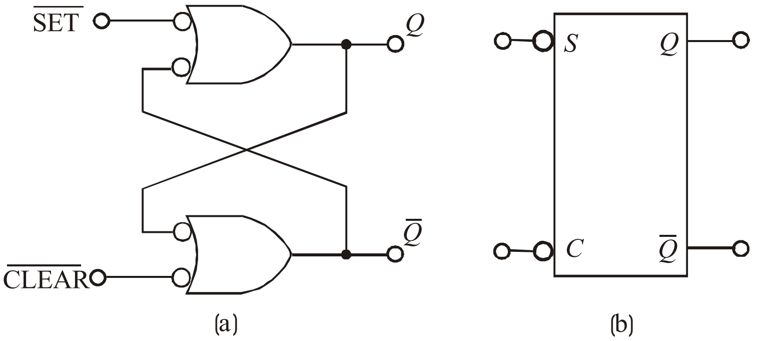


Fig. 5-7. *Alternative representations of NAND gate latch.*

Fig. 5.7 (b) shows a block representation. The S and C labels represent the SET and CLEAR inputs. While the bubbles at S and C inputs indicate the active LOW nature of these inputs. So remember whenever we use this block symbol, it represents a NAND gate latch.

Note. The action of *Clearing* a flip-flop or a latch is also called *resetting*. Both these terms (i.e. clearing or resetting) is used interchangeably in the field of digital electronics. Thus a SET-CLEAR latch can also be refered to as SET-RESET latch (or simply S - R latch).

Example 5-1. *The waveforms shown in Fig* 5-8 (*a*) *are applied to the inputs of the NAND latch shown in Fig.* 5-8 (*b*). *Assume that initially* $Q = 0$ *and determine the* Q *– waveform.*

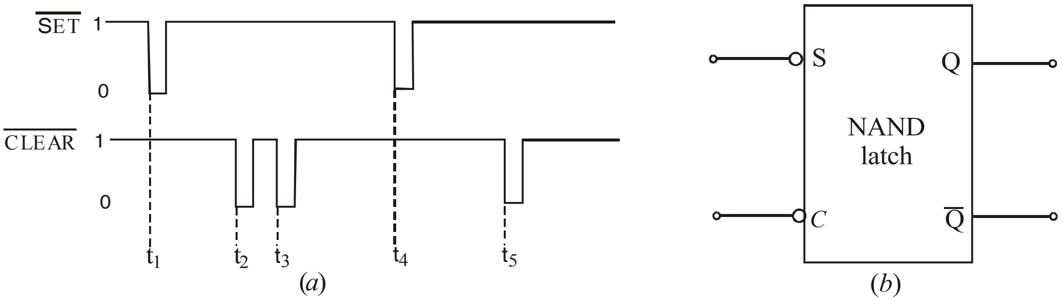


Fig. 5-8.

Solution Given : The waveforms of Fig. 5.8 (a) applied at the inputs of NAND latch.

In order to determine the waveform at *Q*-output, refer to the truth table shown in Fig. 5.6 and the input waveforms. Notice that prior to t = t_1, $\overline{\text{SET}}$ = $\overline{\text{CLEAR}}$ = 1 and *Q* = *0*, At t = t_1, $\overline{\text{SET}}$ goes 0 and $\overline{\text{CLEAR}}$ stays at 1. Referring to the truth table in Fig. 5-6, we find that if $\overline{\text{SET}}$ = 0, $\overline{\text{CLEAR}}$ = 1, *Q* = 1. Therefore the *Q*–output is indicated as 1 at t = t_1 in Fig. 5-9.

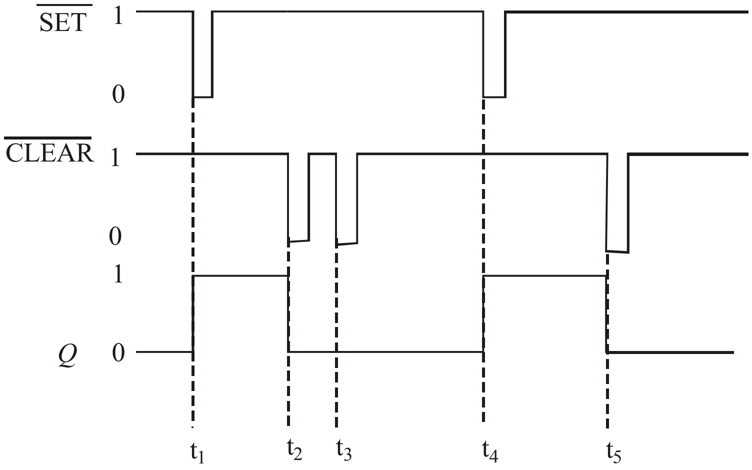
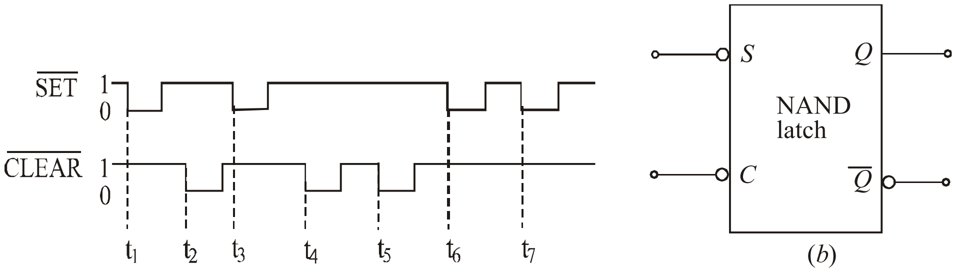


Fig. 5-9.

At t = t_2, $\overline{\text{SET}}$ = 1,and $\overline{\text{CLEAR}}$ goes 0. Again referring to the truth table shown in Fig. 5.6, we find that *Q* = 0, therefore we sketch the *Q* output as 0 in Fig. 5-9. Similarly, at t = t_3, $\overline{\text{SET}}$ = 1, and $\overline{\text{CLEAR}}$ = 0. Since there is no change in inputs, the *Q*-output also remains at 0 as indicated in Fig. 5-9. At t = t_4, $\overline{\text{SET}}$ goes 0, and $\overline{\text{CLEAR}}$ is 1, therefore the output goes 1. At t = t_5, $\overline{\text{SET}}$ = 1 and $\overline{\text{CLEAR}}$ goes 0, therefore output goes 0. The complete *Q*–output waveform is shown in Fig. 5-9.

Example 5-2. *If the* $\overline{SET}$ *and* $\overline{CLEAR}$ *waveforms shown in Fig* 5.10 (*a*) *are applied to the inputs of NAND latch shown in Fig.* 5-10 (*b*) *determine the waveform that will be observed on the Q output. Assume that initially Q* = 0.



(*a*) (*b*)

Fig. 5-10.

Solution : Given the waveforms of Fig 5-10 (a) applied at the inputs of NAND latch.

In order to determine, the Q–output, let us refer to the truth table shown in Fig. 5-6 and the input waveforms. At t = t_1, $\overline{\text{SET}}$ goes from 1 to 0, $\overline{\text{CLEAR}}$ = 1, so the output *Q* goes from 0 to 1 as shown in Fig 5-11. At t = t_2, $\overline{\text{SET}}$ = 1, $\overline{\text{CLEAR}}$ goes from 1 to 0, so does the *Q*–output (*i.e.* it also goes from 1 to 0). At t = t_3, $\overline{\text{SET}}$ goes from 1 to 0, $\overline{\text{CLEAR}}$ = 1, therefore *Q*–output goes 1, At t = t_4 , SET =1, $\overline{\text{CLEAR}}$ goes to 0, therefore *Q*–output goes 0 and so on. The complete waveform at *Q*–output is shown in Fig. 5-11.

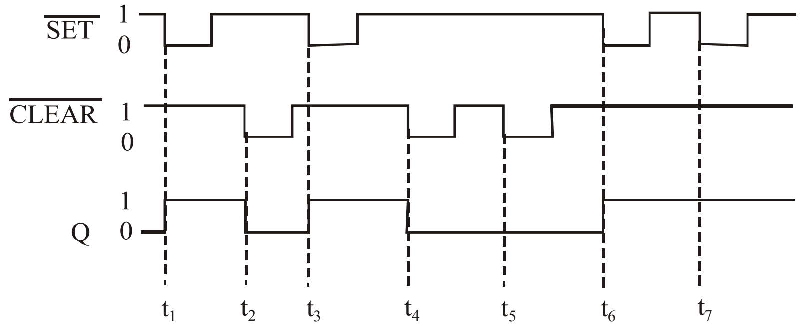


Fig. 5-11.

5-6. Application of NAND Latch to Debounce a Mechanical Switch

Consider a mechanical switch with contact points *A* and *B* connected to V_{cc} (+ 5V) supply and ground respectively as shown in the Fig. 5-12 (a). It has been found experimentally that when the switch moves from contact position *A* to *B*, it produces several output voltage transitions as shown in Fig. 5-12 (b). It is due to a phenomenon called *switch bounce i.e.* the switch makes and breaks contact with contact *B* several times before coming to rest on contact *B* **.** In a similar manner, when the switch moves from contact position *B* to *A*, it again produces several output voltage transitions before coming to rest on contact *A*.

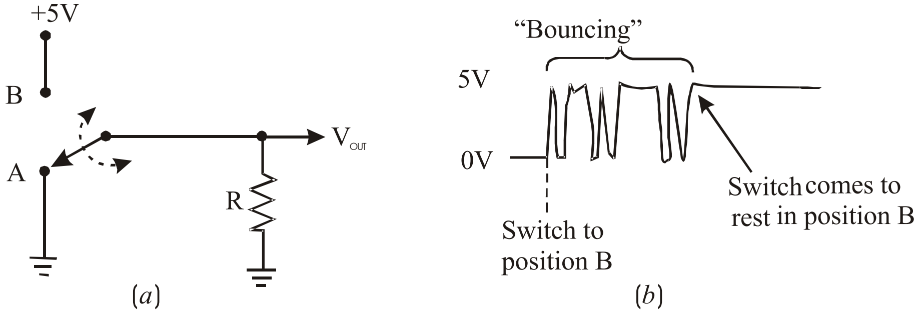


Fig. 5-12. *Illustrating Switch Contact bounce*

As a matter of fact, the multiple transitions on the output signal generally last only for few milliseconds. But such transitions are unacceptable in many applications. A NAND latch can be used as shown in Fig. 5-13 (a) to eliminate the switch bounce.

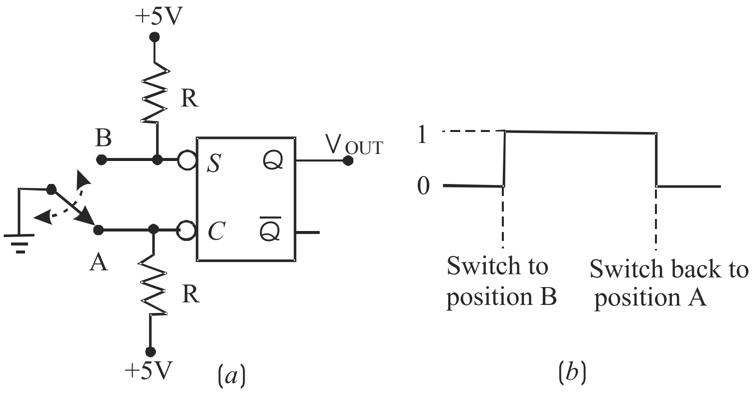


Fig. 5-13. *Application of NAND latch to eliminate switch contact bounce.*

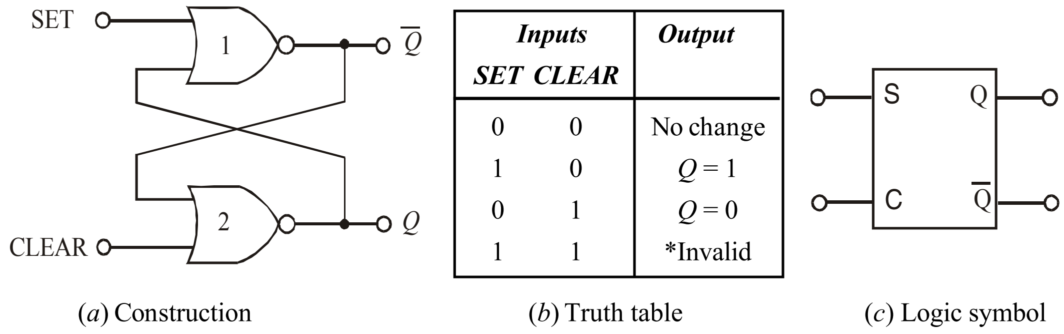
The operation of the switch debouncing circuit may be understood as follows : Let us assume that initially the switch is resting in position A so that $\overline{\text{CLEAR}}$ input is LOW and $Q = 0$. When the switch is moved to position B, $\overline{\text{CLEAR}}$ will go HIGH and a LOW will appear on the $\overline{\text{SET}}$ input as the switch first makes contact. This will set $Q = 1$ within few nanoseconds. Now if the switch bounces off contact B, $\overline{\text{SET}}$ and $\overline{\text{CLEAR}}$ will both be HIGH and Q will not be affected, *i.e.* it will stay HIGH. Thus nothing will happen at Q as the switch bounces on and off contact B before finally coming to rest in position B as shown in Fig. 5-13 (b).

In a similar manner, when the switch is moved from position B back to position A, it will place a LOW on the $\overline{\text{CLEAR}}$ input as it first makes contact. This clears Q to the LOW state. It will remain there ever if the switch bounces on and off contact A several times before coming to rest.

It is evident from the above discussion that the output at Q will consists of single transition each time the switch is moved from one position to the other.

5-7. NOR Gate Latch

Fig. 5-14 (a) shows a latch constructed from two cross-coupled NOR gates. It is called NOR gate latch or simply the NOR latch. As seen in the figure, the arrangement is similar to the NAND gate latch (shown in Fig 5-2 on page 188) except that the Q and $\overline{Q}$ outputs have reversed positions.



Inputs		*Output*
SET	*CLEAR*	
0	0	No change
1	0	$Q = 1$
0	1	$Q = 0$
1	1	*Invalid

(*a*) Construction (*b*) Truth table (*c*) Logic symbol

Fig. 5-14. *NOR gate latch*

The detailed operation of the NOR latch can be understood exactly in the same manner as for the NAND latch. The results are given in the truth table as shown in Fig 5-14 (b) and are summarized as follows :

1. **SET = CLEAR = 0** This condition is the normal resting state for the NOR latch. It has no effect on the output state. In other words, Q and $\overline{Q}$ will remain in the same state in which they were prior to this input condition.
2. **SET = 1, CLEAR = 0** This condition will always cause the output to go to $Q = 1$ state where it will remain even after SET returns to 0.
3. **SET = 0, CLEAR = 1** This condition will always cause the output to go to $Q = 0$ state where it will remain even after CLEAR returns to 0.
4. **SET = 1, CLEAR = 1** This condition tries to set and clear the latch simultaneously. It produces invalid results and should not be used.

It may be carefully noted that the NOR latch operates exactly in the same manner as the NAND latch. However, the SET and CLEAR inputs are active-HIGH rather than active-LOW. Moreover the normal resting state is SET = CLEAR = 0. Further the output Q will be set HIGH by a HIGH pulse on the SET input and it will be cleared LOW by a HIGH pulse on the CLEAR input. Fig 5.14 (c) shows the logic symbol for the NOR latch. Notice that there are no bubbles on the S and C inputs (unlike NAND latch) which indicates that these inputs are active-HIGH.

Example 5-3. *The waveforms shown in Fig.* 5-15 (*a*) *are applied to the inputs of the NOR latch shown in Fig.* 5-15 (b). *Assume that initially*, Q = 0 *and determine the Q–waveform.*

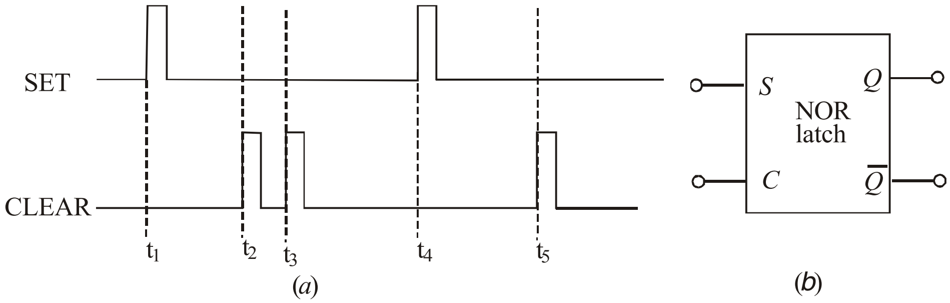


Fig. 5-15.

Solution : Given the waveforms of Fig. 5.15 (a) applied at the SET and CLEAR inputs of NOR latch.

In order to determine the Q-waveform, we will refer to the truth table of the NOR latch (shown in Fig. 5-14 (b) page 174) and the SET and CLEAR waveforms of Fig. 5-15 (a). At t = t_1, SET goes 1 and CLEAR is 0, therefore Q–output also goes 1. AT t = t_2, SET = 0 and CLEAR goes 1, therefore the Q–output goes 0. At t = t_3, SET = 0 and CLEAR goes 1 again, therefore the Q–output remains 0. At t = t_4, SET goes 1 and CLEAR is 0, therefore Q–output goes 1. At t = t_5, SET = 1 and CLEAR goes 1, therefore Q–output goes 0. The complete sketch of the Q-waveform along with SET and CLEAR waveforms is as shown in Fig. 5-16.

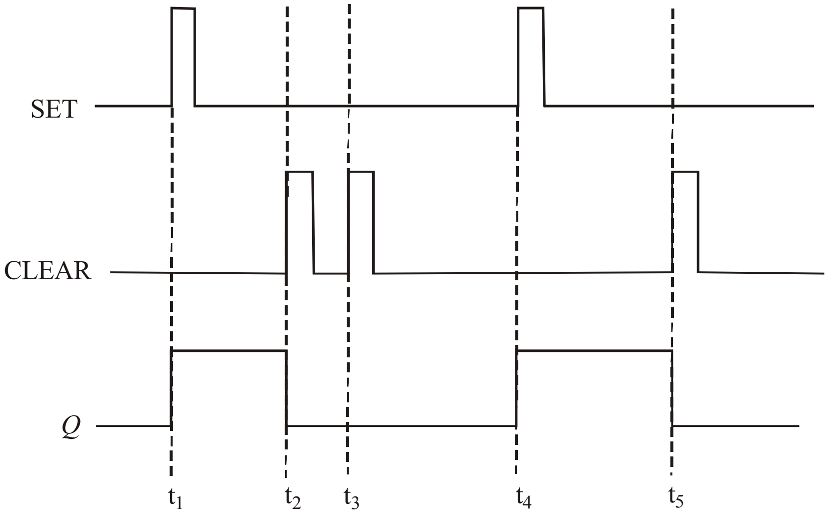


Fig. 5-16.

Example 5-4. *If the SET and CLEAR waveforms shown in Fig.* 5-16 (*a*) *are applied to the inputs of NOR latch shown in Fig.* 5-17 (*b*), *determine the waveform that will be observed on the Q output. Assume that initially Q* = 0.

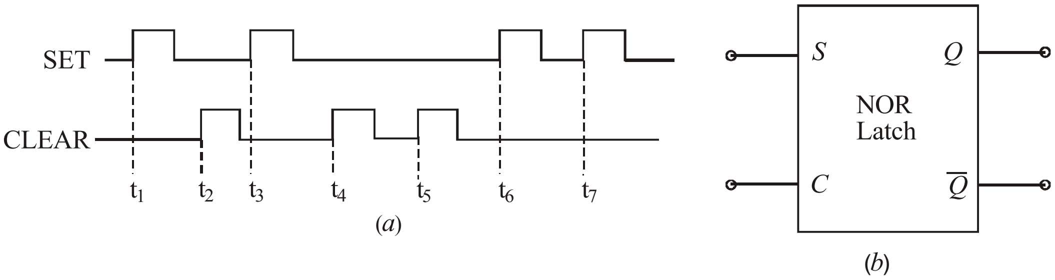


Fig. 5-17.

Solution : Given the waveforms of Fig. 5-17 (a) applied at the SET and CLEAR inputs of NOR latch.

In order to determine the Q-waveform, we will refer to the truth table of the NOR latch (shown in Fig. 5-14 (b) and the SET and CLEAR waveforms of Fig. 5-16. At t = t_1, SET goes 1 and CLEAR = 0, therefore *Q* goes 1. At t = t_2, SET = 0 and CLEAR goes 1, therefore *Q* goes 0. At t = t_3, SET = 1 again, and CLEAR = 0, therefore *Q* goes 1 again. At t = t_4, SET = 0. and CLEAR = 1, therefore *Q* goes 0 and so on. A complete sketch of *Q* waveform along with input waveforms is shown in Fig. 5-18.

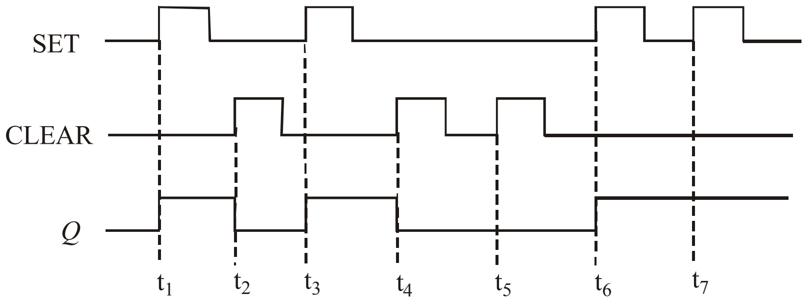


Fig. 5-18.

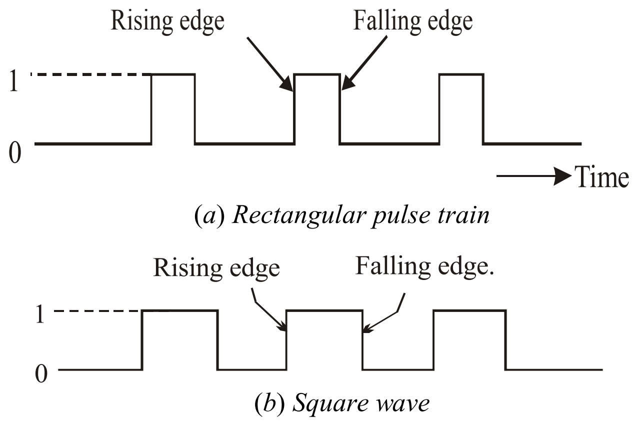
5-8. Clocked Signals

Strictly speaking, the digital systems are of the following two types:

1. **Asynchronous systems** : In these systems, the outputs of logic circuits can change state any time one or more of the inputs change. Generally, an asynchronous system is more difficult to design and troubleshoot than a synchronous system.

2. **Synchronous systems.** In these systems, the exact time at which the output can change states are determined by a signal commonly called a *clock.*

The clock signal is generally a rectangular pulse train as shown in Fig. 5-19 (a) or a square wave as shown in Fig. 5-19 (b). The clock signal is distributed to all parts of the digital system and most of the system outputs can change state only when the clock makes a transition. The transitions are more commonly referred to as ***edges*** and are pointed out in Fig. 5-19.



(*a*) *Rectangular pulse train*

(*b*) *Square wave*

Fig. 5-19.

It may be noted that when the clock changes from 0 to 1, this is called rising edge or *positive-going transition* (PGT). On the other hand, when the clock changes from 1 to 0, this is called falling edge or *negative going transition* (NGT).

As a matter of fact, most digital systems are principally synchronous (although there are always some asynchronous parts). It is due to the fact that synchronous circuits are easier to design and troubleshoot. The synchronization is accomplished through the use of *clocked flip-flops* that are designed to change states on one or the other of the clock's transitions.

5-9. Some Main Ideas Common to Clocked Flip-Flops

There are several types of flip-flops that are used in a wide range of applications in the field of digital electronics. Before we begin our study of the different clocked flip-flops, let us describe the main ideas that are common to all of them.

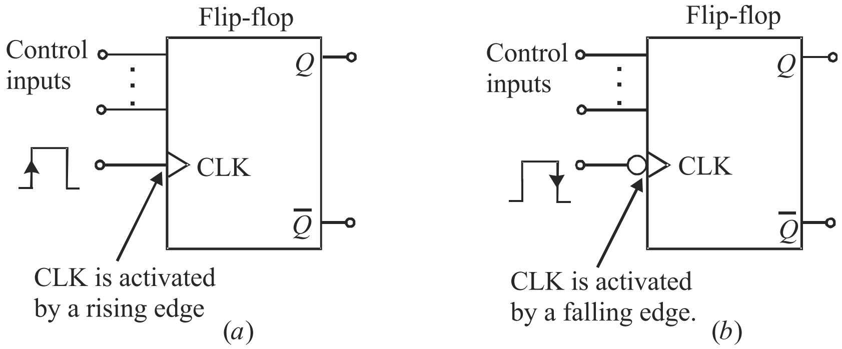


Fig 5-20. *Clocked Flip-Flops*

Fig. 5-20 (a) and (b) shows the logic symbols for a typical clocked flip-flop. As seen, a clocked flip-flop has a clock input and some control inputs. There are described below in more detail.

CLK Input Clocked flip-flops have a *clock* input that is labeled as CLK, CK or CP. However, we will normally use CLK as shown in Fig. 5-19 (a) and (b). In most clocked flip-flops, the CLK input is *edge-triggered*. This means the CLK input is activated by a signal transition. The edge-triggered activation is indicated by the presence of a small triangle on the CLK input. This contrasts with the latches, which are level-triggered.

Fig. 5-20 (a) shows a flip-flop with a small triangle on its CLK input to indicate that the input is activated only when a rising edge occurs. It may be noted that no other part of the input pulse will have an effect on the CLK input.

Fig 5-20 (b) shows a flip-flop symbol which has a bubble (a small circle) as well as a triangle on its CLK input. This signifies that CLK input is activated *only* when a falling edge occurs. Again note that no other part of the input pulse will have an effect on the CLK input.

Control Inputs Clocked flip-flops also have one or more *control inputs* that can have various names depending on their operation. The control inputs will have no effect on *Q* until the active clock transition occurs. In other words, their effect is synchronized with the signal applied to CLK. Because of this reason, the control inputs are refered to as *synchronized control inputs.*

5-10. Setup and Hold Times in Clocked Flip-Flops

Strictly speaking, there are two timing requirements that must be met if a clocked flip-flop is to respond relialably to its control inputs when the active CLK edge occurs. These two requirements are : (1) set up time, t_S and (2) hold time, t_H. Both these requirements are discussed below.

1. Set up time, t_s It is the time interval immediately preceeding the active edge of the CLK signal during which the control input must be maintained at the proper level. The situation is illustrated in Fig 5.21 (a) for a flip-flop that triggers on the rising edge. Usually, the IC manufacturers specify the minimum allowable set up time, t_S (min). If this time requirement is not met, the flip-flop may not respond reliably when the clock edge occurs.

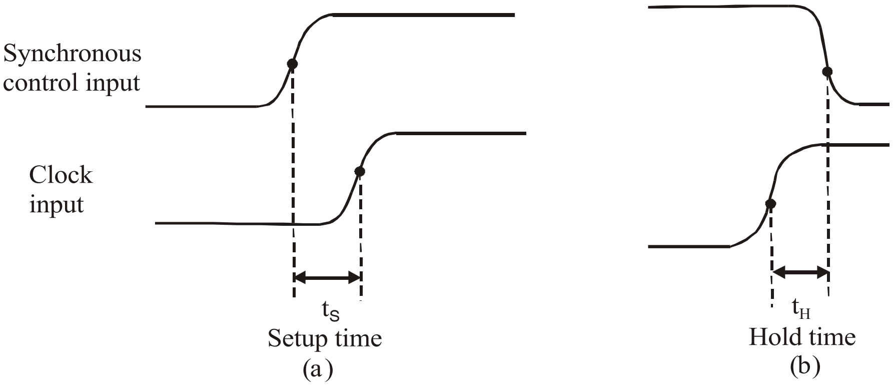


Fig. 5-21. *Illustrating the set-up and hold time requirements, for a flip-flop that triggers on the rising edge.*

2. Hold time, t_H It is the time interval immediately following the active edge of the CLK signal during which the synchronous control input must be maintained at the proper level. Refer to Fig. 5-20 (b) usually the IC manufacturers specify the minimum acceptable value of hold time, t_H (min). If this requirement is not met, the flip-flop will not trigger reliably.

It is evident from the above discussion that in order to ensure that a clocked flip-flop will respond properly when the active clock edge occurs, the control inputs must be stable (*i.e.* unchanging) for at least a time interval.

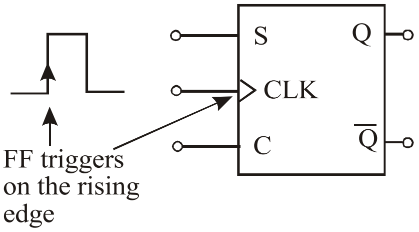
1. t_S (min) prior to the active clock edge.
2. t_H (min) after the active clock edge.

A typical value of t_S (min) is in the range of 5 to 50 ns whereas hold times are generally from 0 to 10 ns. It may be carefully noted that the set up and hold times are measured between the 50 per cent points on the edges.

The set up and hold time requirements are extremely important in synchronous systems. This is due to the reason that there will be many situations where the synchronous control inputs to a flip-flop are changing at approximately the same time as the CLK input.

5-11. Clocked S - C Flip-Flop

Fig. 5-22 (a) shows the logic symbol for a clocked S - C flip-flop that is triggered by the rising edge of the clock signal. In other words this flip-flop can change output states only when a signal applied to its clock input makes a transition from 0 to 1. The S and C inputs control the state of the flip-flop in the same manner as discussed earlier for the NOR latch. But it may be noted that the flip-flop does not respond to these inputs until the occurrence of the rising edge of the clock signal. It also called S-R or R-S flip-flop.



(*a*) Symbol

Inputs			Output
S	*C*	*CLK*	*Q*
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	Ambiguous

(*b*) Truth table

Fig. 5-22. *Clocked S-C Flip-Flop.*

Fig. 5-22 (b) shows the truth table for a clocked S-C flip-flop that is triggered by the rising edge of the clock signal. The truth table indicates how the flip-flop output will respond to the rising edge at the CLK input for the various combinations of S and C inputs. The up arrow (↑) in the truth table indicates that the rising edge is required at CLK input. The label Q_0 indicates the level at Q prior to the rising edge. This nomenclature is used quite often by IC manufacturers in their IC data sheets/manuals.

The operation of S-C flip-flop may be understood with the help of input and output waveforms shown in Fig 5-23. Assuming that the setup and hold time requirements are being met in all cases, the waveforms can be analysed as follows :

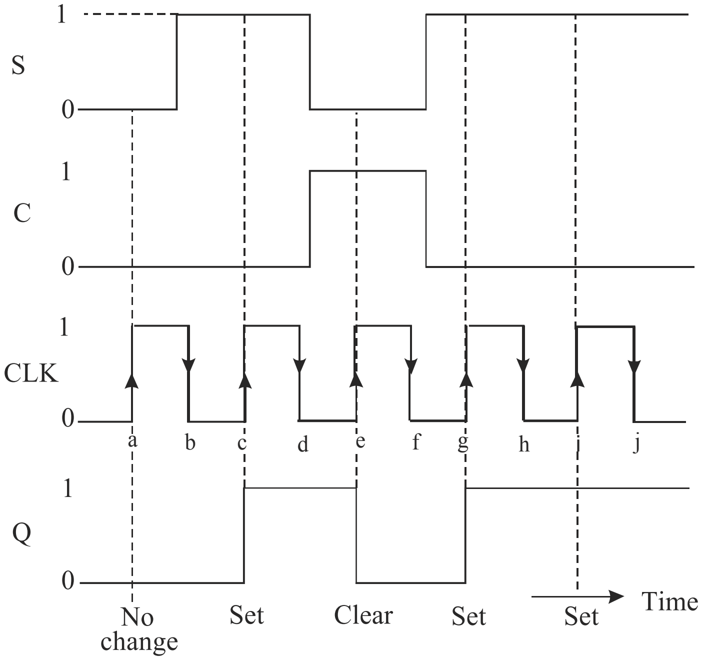
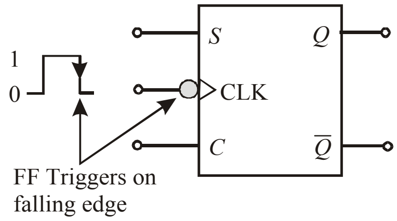


Fig. 5-23. *Typical input and output waveforms of a S-C flip-flop.*

1. Notice that initially all the inputs (S, C and CLK) are 0 and the Q output is assumed to be zero, i.e. $Q_0 = 0$.
2. When the rising edge of the first clock pulse occurs (refer to point 'a'), both the S and C inputs are 0, so the flip-flop output is not affected and remains in the $Q = 0$ state (i.e. $Q = Q_0 = 0$).
3. At the occurrence of the rising edge of the second clock pulse (refer to point 'c'), the S input is now HIGH, with C input still LOW. This causes the flip-flop output to set to 1 state.
4. At the occurrence of the rising edge of the third clock pulse (refer to point 'e'), the S input is LOW with C input HIGH. This causes the flip-flop output to clear to 0 state.
5. The rising edge of the fourth clock pulse sets the flip-flop output to the Q = 1 state (refer to point 'g') because S = 1 and C = 0.
6. The fifth clock pulse also finds that S = 1 and C = 0 at the rising edge (refer to point 'i'). This situation will produce HIGH output. But since Q is already HIGH, so it remains in that state.

It may be carefully noted from the waveforms shown in Fig. 5-23, that the flip-flop is not affected by the falling edge of the clock pulses. It may also be noted that S and C input levels have no effect on the flip-flop except upon the occurrence of a rising edge of the clock signal.



(*a*) Symbol

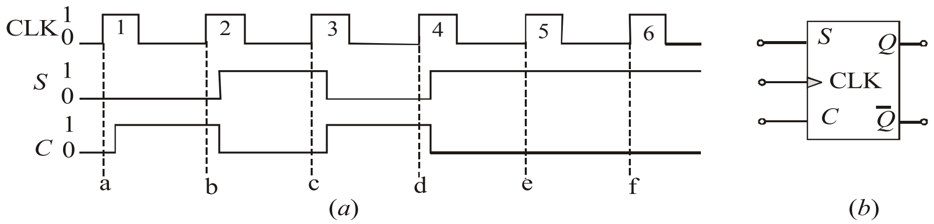
Inputs			*Output*
S	***C***	***CLK***	***Q***
0	0	↓	Q_0 (no change)
1	0	↓	1
0	1	↓	0
1	1	↓	Ambiguous

(*b*) Truth table

Fig. 5-24. *Falling-edge triggered S-C flip-flop.*

Fig. 5-24 (a) shows the symbol and 5-24 (b) the truth table for a clocked S - C flip-flop that triggers on the falling edge of the CLK input. Notice the presence of a small circle and a triangle on the CLK input of the flip-flop. These indicate that this flip-flop will trigger only when CLK input goes from 1 to 0. This flip-flop operates in the same way as the rising-edge flip-flop except that the output can change states only on the falling edge of the clock pulses (refer to points b, d, f, h and j in Fig. 5-22). In actual practice both the rising-edge and falling-edge triggered flip-flops are used in digital systems.

Example 5-5. *Fig.* 5-25 (*a*) *shows the SET and CLEAR waveforms applied at the inputs of the clocked S-C flip-flop shown in Fig.* 5-25 (*b*)



(*a*) (*b*)

Fig. 5-25.

Sketch the waveforms at Q and $\overline{Q}$ outputs of the flip-flop. Assume that initially Q = 0 *and* $\overline{Q}$ = 1.

Solution. Given : The waveforms shown in Fig. 5-25 at CLK, S and C inputs of the rising edge triggered S-C flip-flop. In order to determine the *Q*-output, we will refer to the waveforms applied at the flip-flop inputs shown in Fig. 5-25 (*a*) and its truth table shown in Fig. 5-24 (b) At point 'a', $S = C = 0$, therefore $Q_0 = 0$, At point 'b', $S = 0$ and $C = 1$, therefore *Q* remains 0. At point 'c', $S = 1$ and $C = 0$, therefore $Q = 1$. At point 'd', $S = 0$ and $C = 1$, therefore $Q = 0$. At point 'e', $S = 1$, $C = 0$, therefore $Q = 1$. At point 'f', S and *C* inputs remain the same, therefore *Q* also remains 1. The sketch of *Q*–output along with the CLK, *S* and *C* waveforms is as shown in Fig. 5-26. The $\overline{Q}$–output waveform is determined by inverting the *Q*-waveform.

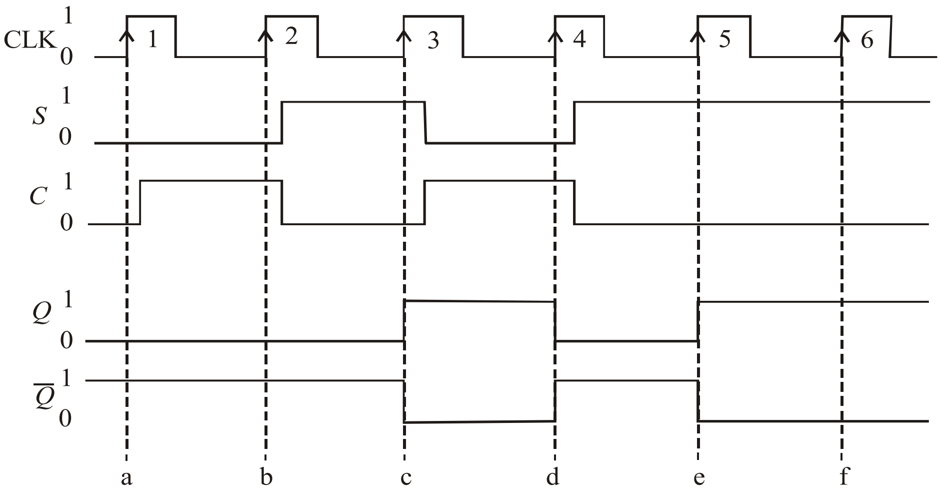


Fig. 5-26.

5-12. Internal Circuitry of an Edge-triggered S-C Flip-Flop

Fig. 5-27 shows a simplified version of an internal circuitry of an edge triggered S-C flip-flop. Notice that the circuit contains the following three sections:

1. A basic NAND latch
2. A pulse steering circuit and
3. An edge-detector circuit

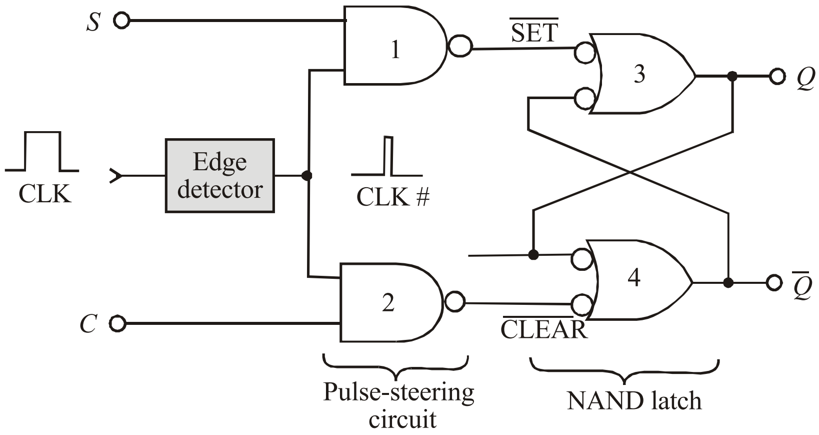
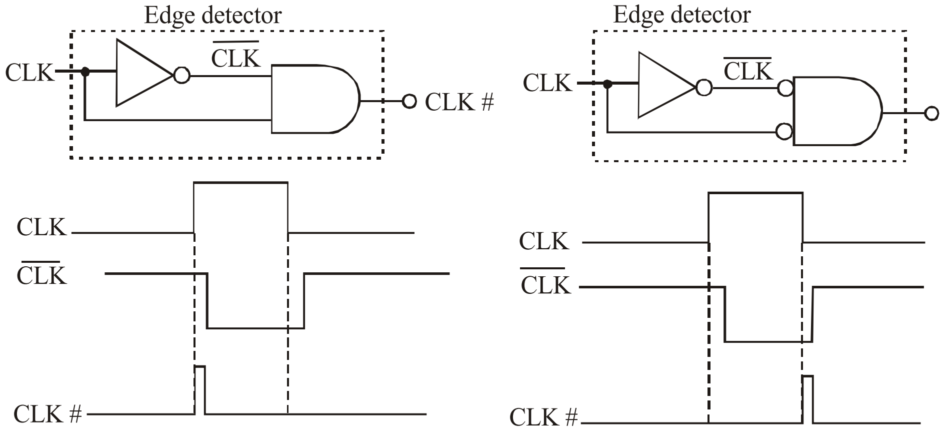


Fig. 5-27. *Internal circuitry of an edge-triggered S-C flip-flop.*

As seen from the Fig. 5-27, the edge detector produces a narrow positive-going spike (CLK #). The pulse-steering circuit "steers" (or moves) the spike through to the SET and CLEAR input of the latch in accordance with the levels present at S and C inputs. For example with $S = 1$ and $C = 0$, the spike is inverted and passed through NAND gate-1 to produce a LOW pulse at the SET input of the latch that sets $Q = 1$. With $S = 0$ and $C = 1$, the spike is inverted and passed through NAND gate-2 to produce a LOW pulse at the CLEAR input of the latch that resets $Q = 0$.



(a) Rising edge

(b) Falling edge

Fig. 5-28. *Implementation of edge-detector circuits used in edge triggered flip-flops.*

Fig. 5-28 shows the implementation of edge-detector circuits used in edge-triggered flip-flops. Thus Fig. 5-28 (a) shows how the spike is generated for edge-triggered flip-flops that trigger on the rising edge of the CLK pulse. As seen from this figure, the INVERTER produces an output with a delay of a few nanoseconds. Because of this the transitions of $\overline{\text{CLK}}$ occur a little bit after that of CLK. Next the AND gate produces an output spike that is HIGH only for the few nanoseconds when both CLK and $\overline{\text{CLK}}$ are HIGH. As a result of this, we get a narrow pulse at edge detector output (CLK #) which occurs on the rising edge of the CLK.

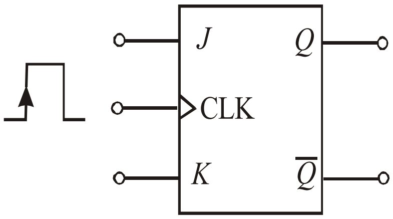
Similarly Fig 5-28(b) produces CLK # on the falling edge of CLK for flip-flops that are to trigger on the falling edge.

It may be noted that since the CLK # signal is HIGH for only a few nanoseconds, the Q output is affected by the levels at S and C only for a short time during and after the occurrence of the active edge of CLK. This gives the flip-flop its edge-triggered property.

5-13. Clocked J-K Flip-Flop

Fig. 5-29 (a) shows the symbol and (b) the truth table for a clocked J-K flip-flop that is triggered by the rising edge of the clock signal. The J and K inputs control the state of the flip-flop in the same manner as the S and C inputs do for the S-C flip-flop. However there is one major difference-the $J = K = 1$ condition in J-K flip-flop does not result in an ambiguous output unlike $S = C = 1$ in S-C flip-flop for, $J = K = 1$ condition, the J-K flip-flop will always go to its opposite state upon the rising edge of the clock signal. This is called *trigger mode.* In this mode, if both J and K are left HIGH, the flip-flop will change states (i.e. toggle) for each rising edge of the clock.

The operation of J-K flip-flop for each combination of J and K is summarized in Fig. 5-29 (b) Notice that the truth table is the same except for $J = K = 1$ condition. This condition results in $Q = \overline{Q}_0$ which means that the new value of Q will be the inverse of the value it had prior to the rising edge of the clock pulse. It is called toggle operation.



(*a*) Symbol

Inputs			*Output*
J	***K***	***CLK***	***Q***
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	Q_0 (toggles)

(*b*) Truth table

Fig. 5-29. *Rising edge triggered J-K flip-flop.*

In order to understand the operation of J-K flip-flop, let us consider the J, K and CLK waveforms as shown in Fig. 5-30. Assume that set up and hold time requirements are met. The operation may be explained as below.

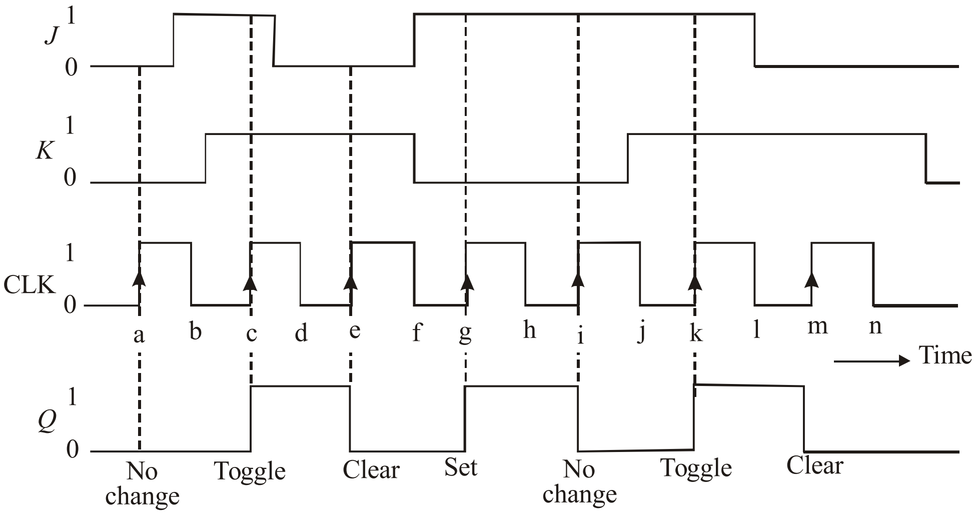


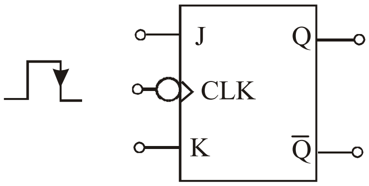
Fig. 5-30.

1. Initially all the inputs are 0, and the Q output is also assumed to be 0, *i.e.* $Q_0 = 0$.
2. When the rising edge of the first clock pulse occurs (refer to point a), the $J = K = 0$ condition exists. Thus the flip-flop does not change its output state, *i.e.* $Q = Q_0 = 0$.
3. When the rising edge of the second clock pulse occurs (refer to point C), the $J = K = 1$ condition exists. Thus the flip-flop toggles to its opposite state *i.e.* $Q = \overline{Q_0} = \overline{0} = 1$.
4. When the rising edge of the third clock pulse occurs (refer to point e), $J = 0$ and $K = 1$ condition exists. Thus the flip-flop is cleared to the $Q = 0$ state.
5. When the rising edge of the fourth clock pulse occurs (refer to point g), $J = 1$ and $K = 0$ condition exists. This condition sets the output Q to 1 state.
6. When the rising edge of fifth clock pulse occurs (refer to point i), $J = 1$ and $K = 0$ condition exists. This is the condition that sets the output Q to 1 state. However since Q is already 1, so it will remain there. Hence no change in the output state.

7. When the rising edge of six clock pulse occurs (refer to point k), $J = K = 1$ condition exists. This condition causes the flip-flop to toggle to its opposite state.

8. When the rising edge of seventh clock pulse occurs (refer to point m), $J = 0$ and $K = 1$ condition exists. This condition causes the flop-flop to clear to $Q = 0$ state.

It may be noted from the waveforms that the flip-flop is not affected by the falling edge of the clock pulses. Also the J and K input levels have no effect except the occurrence of the rising edge of the clock signal. The J and K inputs by themselves cannot cause the flip-flop to change states.



(*a*) Symbol

Inputs			***Output***
J	***K***	***CLK***	***Q***
0	0	↓	Q_0 (no change)
1	0	↓	1
0	1	↓	0
1	1	↓	Q_0 (toggles)

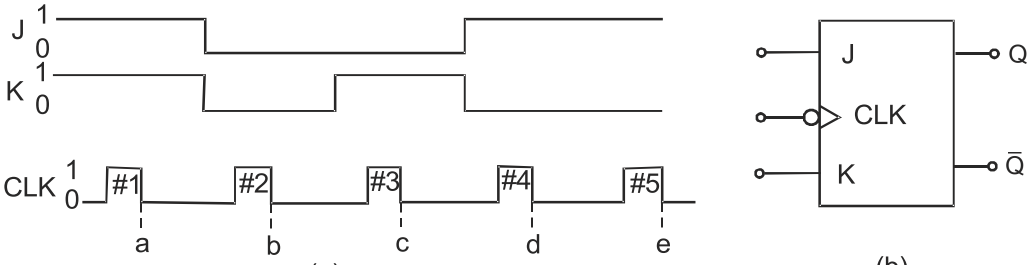
(*b*) Truth table

Fig. 5-31. *Falling edge triggered J-K flip-flop.*

Fig. 5-31 shows the symbol and 5-31 (b) the truth table for a clocked J-K flip-flop that triggers on the falling edge of the clock pulse. The small circle on the CLK input indicates that the flip-flop will trigger when the CLK input goes from 1 to 0. This flip-flop operates in the same way as the rising edge of the flip-flop of Fig. 5-27 except that the output can change states only on the falling edge of CLK signal (i.e. points b, d, f, h, j, l and n). As a matter of fact, both polarities of edge-triggered J-K flip-flops are in common usage in the field of digital electronics.

Strictly speaking, the J-K flip-flop is much more versatile than the S-C flip-flop because it has no ambiguous states. The $J = K = 1$ condition, which produces the toggling operation, finds extensive use in all types of binary counters.

Example 5-6. Fig 5-32 (*a*) *shows the waveforms applied at J, K and CLK inputs of the clocked J– K flip-flop shown in Fig.* 5-32 (*b*). *Sketch the Q output waveform Assume $Q = 0$ initially.*



(a)

(b)

Fig. 5-32.

Solution. Given : The waveforms at J, K and CLK inputs of a J-K flip-flop shown in Fig. 5-32.

Notice that flip-flop shown in Fig. 5-32 (*b*) is a clocked J-K flip-flop. Also notice the presence of a small circle at the CLK input of the flip-flop. This indicates that the flip-flop will trigger coresponding to the falling edge of the clock pulse. So you need to identify the states of J and K inputs coresponding to the falling edge points of the CLK pulse waveform i.e. points a, b, c, d and e shown in Fig. 5-31.

Using the truth table of clocked J-K flip-flop shown in Fig. 5-30, and the given waveforms the *Q*-waveform sketch is shown in Fig. 5-33. On the arrival of first CLK pulse at point 'a' J = K = 1, the a output does not change, *i.e.* it stays at *Q* = 1 level. On the arrival of third CLK pulse J-K flip-flop will toggle, i.e., its Q-output changes from 0 to 1. On the arrival of second CLK pulse, at point 'b', J = K = 0 the Q-at point 'c' J = 0, K = 1, the *Q* output goes 0. On the arrival of fourth CLK pulse, at point 'd' J = 1, K = 0, the *Q*-output goes 1. On the arrival of fifth CLK pulse at point 'e' J = 1, K = 0, the *Q*-output remains at 1 level.

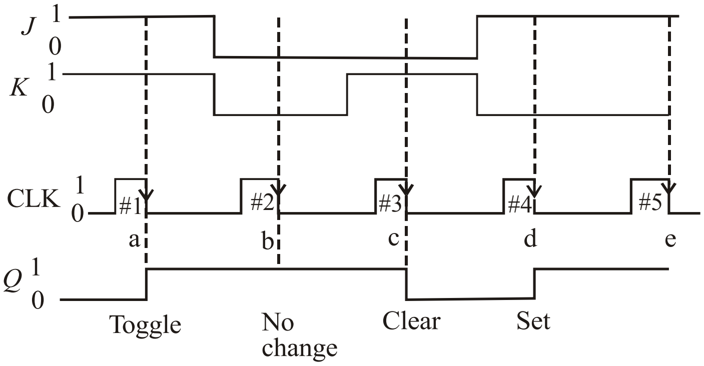


Fig. 5-33.

Example 5-7. *What will be the output waveform Q of a J-K flip-flop if the following waveforms are applied at the input? Assume the flip-flop triggers at the falling edge of clock pulse.*

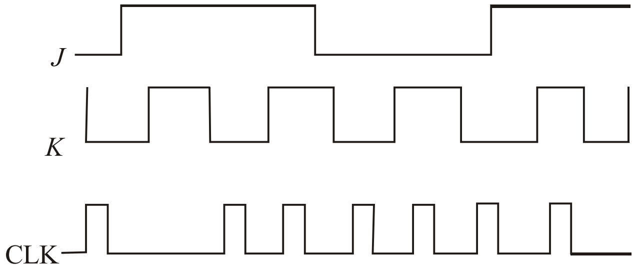


Fig. 5-34.

(*Grad. IETE Dec. 1996*)

Solution. Given : The waveforms at J, K and clock inputs (CLK) of a J-K flip-flop.

Recall the truth table of a clocked J-K flip-flop, the flip-flop triggers corresponding to the logic levels at the J and K inputs. If *J* = *K* = 1, the flip-flop output remains in its previous state. If *J* = 1, *K* = 0 the flip-flop output goes 1, if *J* = 0*, K* = 1*,* the flip-flop goes LOW. However, if *J* = *K* = 1, the flip-flop output toggles. Keeping it in mind, we can sketch the *Q*–output waveform as shown in Fig. 5-35.

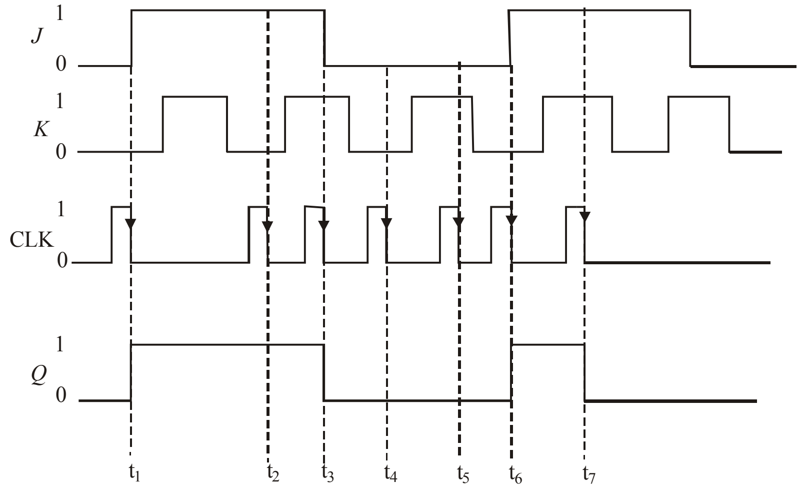


Fig. 5-35.

Notice that at t_1, $J = 1$, $K = 0$ therefore the Q–output goes 1. At t_2 , $J = 1$, $K = 0$ so the Q–output remains 1. At t_3, J goes 0, $K = 1$, therefore the Q-output goes 0. At t_5 , J is still 0, K is still 1, therefore the Q-output stays 0. At t_6 , J goes 1, K is 0, so the Q-output goes 1. At t_6 both J and K are 1, therefore the Q-output toggles and goes 0, as shown in the Fig. 5-35.

5-14. Internal Circuitry of an Edge-triggered J-K Flip-Flop

Fig. 5-36 shows a simplified version of the internal circuitry of an edge-triggered J-K flip-flop. As seen, the flip-flop contains the same three sections as the S-C triggered flip-flop (refer to Fig 5-27 page 181), *i.e.*,

1. A basic NAND gate latch
2. A pulse steering circuit and
3. An edge detector circuit

The only difference between the internal circuitry of edge-triggered J.K. flip-flop and that of S-C flip flop is that in J-K flip-flop circuit, Q and $\overline{Q}$ outputs are fed back to the pulse-steering NAND gates.

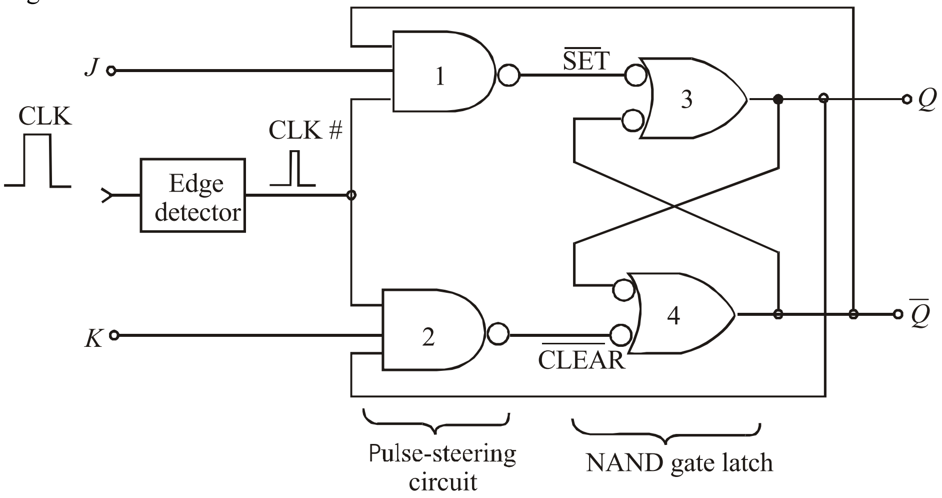


Fig 5-36. *A simplified version of an internal circuitry of an edge-triggered J-K flip-flop.*

It is because of this feedback connection that J-K flip-flop gives the toggle operation for $J = K = 1$ condition.

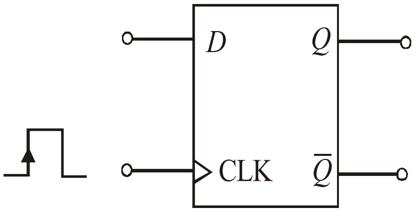
Let us examine the toggle condition in more detail. Assume that $J = K = 1$ and that $Q = 0$ when a CLK pulse occurs. With $Q = 0$ and $\overline{Q} = 1$, NAND gate 1 will steer CLK # (inverted) to $\overline{\text{SET}}$ input of the NAND gate latch to produce $Q = 1$.

If we assume $Q = 1$. When a CLK pulse occurs, NAND gate 2 will steer CLK # (inverted) to the $\overline{\text{CLEAR}}$ input of the latch to produce $Q = 0$. It is evident from the above discussion that Q always ends up in the opposite state.

Note. It may be carefully noted that in order for toggle operation to work, the CLK # pulse must be very narrow. It must return to 0 before the Q and $\overline{Q}$ outputs toggle to their new states. Otherwise the new states of Q and $\overline{Q}$ will cause the CLK # pulse to toggle the latch outputs again.

5-15. Clocked D Flip-Flop

Fig. 5-37 (a) shows the symbol and 5-37 (b) the truth table for a clocked D flip-flop that triggers on the rising edge of the clock pulse. Notice that this flip-flop has only one synchronous control input, D which stands for data.



(*a*) Symbol

Inputs		*Output*
D	***CLK***	***Q***
0	↑	Q
1	↑	1

(*b*) Truth table

Fig. 5-37. *Clocked D flip-flop.*

The operation of the clocked D flip-flop is very simple. The output Q will go to the same that is present on the D input when the rising edge occurs at the CLK. In other words, the level present at D will be stored in the flip-flop at the instant the rising edge occurs.

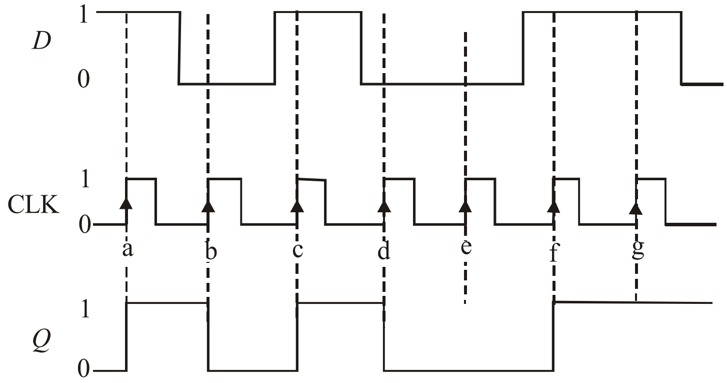


Fig. 5-38. *Input and output waveforms to illustrate the operation of clocked D flip-flop.*

In order to understand the operation of flip-flop in more detail, consider the waveforms at D and CLK input as shown in Fig. 5-38. Assume that *Q* is initially 0.

1. When the first rising edge of the CLK pulse occurs (refer to point 'a'), the D input is 1, therefore *Q* will go to 1 state. Even though the *D* input level changes between the points 'a' and 'b', it has no effect on *Q*. The output *Q* is storing the 1 that was on *D* at point 'a'.
2. When the second rising edge of the CLK pulse occurs (refer to point 'b'), *Q* goes to 0 state since *D* is 0 at that time. The output *Q* stores this 0 value until the rising edge of the third CLK pulse (at point 'c') causes *Q* to go to 1 since *D* is 1 at that time.
3. In a similar manner, the *Q* output takes on the levels present at *D* when the rising edges occur at points 'd', 'e', 'f' and 'g'. Notice that *Q* stays 0 at point 'e' because *D* is still 0.

A falling-edge triggered *D* flip-flop operates in the same way as the rising edge triggered *D* flip-flop. However, the difference is that *Q* will take on the value of *D* when a falling edge occurs at the CLK.

A falling-edge triggered *D* flip-flop operates in the same manner as the *D* flip-flop discussed above except that *Q* will take on the value of *D* when a falling edge occurs at the CLK input. The symbol for D flip-flop that triggers on the falling edge has a bubble on the CLK input as shown in Fig. 5-39.

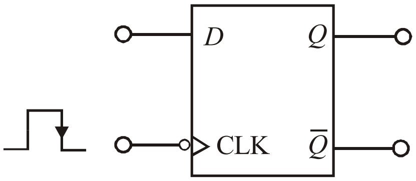


Fig. 5-39. *Symbol for a falling-edge triggered D flip-flop*

IC 7474 is an example of clocked D flip-flop. It contains two rising edge triggered D flip-flops. The pin configuration and some other specification are given in data sheet. Refer to Appendix A(A-14).

5-16. Excitation Table of Flip-Flop

The truth table of a flip-flop is also referred to as the characteristic table of a flip-flop, this table refers to the operational characteristics of the flip-flop. But in designing sequential circuits, we often face situations where the present state and the next state of the flip-flop is specified. By present and next states we mean to say the conditions before and after the clock pulse respectively. For example the output of an SR flip-flop before the clock pulse is 1 and it is desired that the output does not change when the clock pulse is applied.

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

5-17. Excitation table of S-C Flip-Flop

The truth table of S-C flip flop is shown in Fig. 5-40. There are four possible transitions from the present state to the next state. For each transition, the required input condition is derive from the information available in the truth table.

S	C	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Ambiguous

Fig. 5-40.

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

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

Transition from 1 to 0 : The present state is 1 and it change to 0. This can happen only when S = 0 and C = 1 (reset condition). Therefore, S has to be 0 and C has to be 1 for this transition to

Transition from 1 to 1 : The present state is 1 and remains to 1. This can happen either when S = 1 and C = 0 (set condition) or when S = 0 and C = 0 (no change condition). Thus C has to be 0, but S can be either level. The table indicates this with a "X" under S and "0" under C. The excitation table of SC flip-flop is shown in Fig. 5-41.

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

Fig. 5-41.

5-18. Excitation table of J-K Flip-Flop

The truth table of J-K flip flop is shown in Fig. 5-42. There are four possible transitions from the present state to the next state. For each transition, the required input condition is derive from the information available in the truth table.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Fig. 5-42.

Transition from 0 to 0 : The present state of the flip-flop is 0 and is to remain 0 when a clock pulse is applied. Looking at truth table of J-K flip flop, this can be happen either when J = K = 0 or when J = 0 and K = 1. Thus, J has to be 0, but K can be at either level. The table indicates this with a "0" under J and "X" (don't care) under K.

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

Transition from 1 to 0 : The present state is 1 and it change to 0. This can happen only when J = 0 and K = 1 or when J = K = 1. Thus, K has tobe 1 but J can be either level.

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

The excitation table of JK flip-flop is shown in Fig. 5-43.

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

Fig. 5-43.

5-19. Excitation table of D Flip-Flop

The truth table and excitation table for D flip-flop is show in Fig. 5-44. In D flip-flop, the next state is always equal to the D input and it is independent of the present state. Therefore, D must be 0 if Q_{n+1} has to be 0 and 1 if Q_{n+1} has to be 1, regardless of the value of Q_n.

D	Q_{n+1}
0	0
1	1

Truth Table

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

Excitation Table

Fig. 5-44.

5-20. Excitation table of T Flip-Flop

The truth table and excitation table for T flip-flop is show in Fig. 5-45. We know that when T = 1, the state of the flip-flop is complemented. When T = 0, the state of the flip-flop remains unchanged. Therefore, for 0 to 0 and 1 to 1 transitions T must be 0 and for 0 to 1 and 1 to 0 transition T must be 1.

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

Truth Table

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

Excitation Table

Fig. 5-45.

5-21. Implementation of One Type of Flip-Flop to another Type of Flip-Flop

In many case we have to implement the given type of flip-flop to other required flip-flop. We follow a general model for such conversion of flip-flop. This model is shown in Fig. 5-46.

From the model we see that it is required to design the conversion logic for converting new input definitions into input codes that will cause the given flip-flop to work like the desired flip-flop. To design the conversion logic we need to combine the excitation table for both flip-flops and make a truth table with data input and Q as the inputs and the input of the given flip-flop as the output.

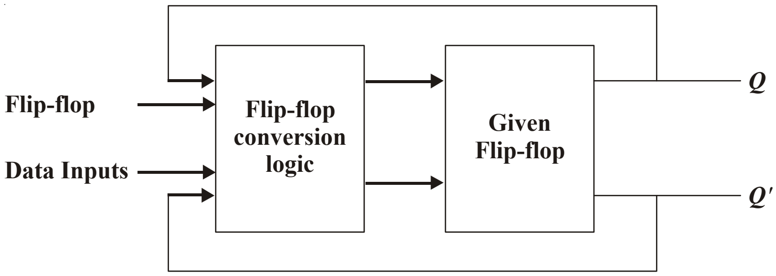


Fig. 5-46.

5-22 Implementation of D Flip-Flop from a J-K Flip-Flop

An edge-triggered D flip-flop can be obtained easily by adding a single INVERTER to an edge-triggered J-K flip-flop as shown in Fig. 5-47. A similar approach can be used to convert a S-C flip-flop to a D flip-flop.

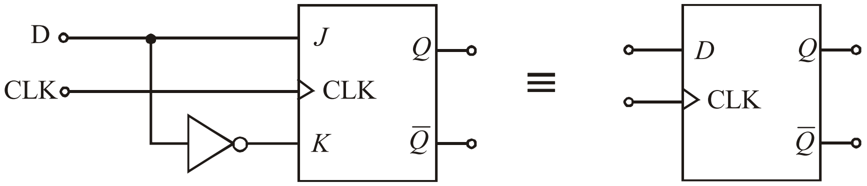


Fig. 5-47. Implementation of D flip-flop from a J-K flip-flop

Example 5-8 *Fig* 5.48 *shows a D flip-flop with the input and clock waveforms applied at their respective inputs. Determine the Q* (*or output*) *waveform.*

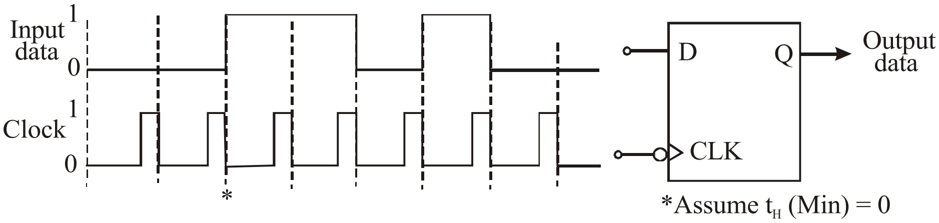


Fig. 5-48.

Solution : Given the D flip-flop with input and clock waveforms.

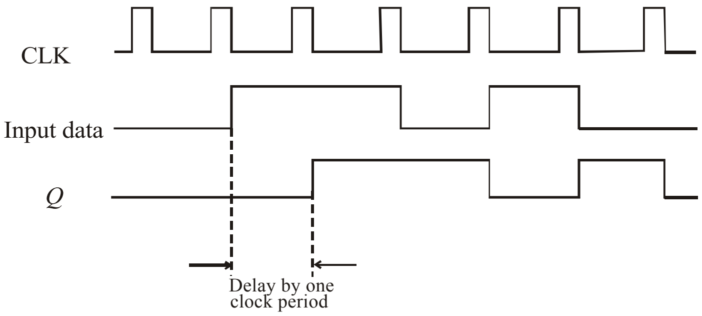


Fig. 5-49.

Note. The output (Q) is delayed from the input by one clock period. This is an advantage as a D-Flip-flop is used sometimes to delay a binary waveform so that the binary information appears at the output a certain amount of time after it appears at the D input.

It is also possible to delay the input by two clock periods. This can be achieved by connecting Q to the D input of a second flip-flop and connect the clock signal to the second flip-flop. The output of the second flip-flop will be delayed by 2 clock periods from the input data.

Example 5-9. *An edge-triggered D Flip-flop can be made to operate in the toggle mode by connecting it as shown in Fig.* 5-50. *Assume Q* = 0 *initially and determine the Q* (*output*) *waveform.*

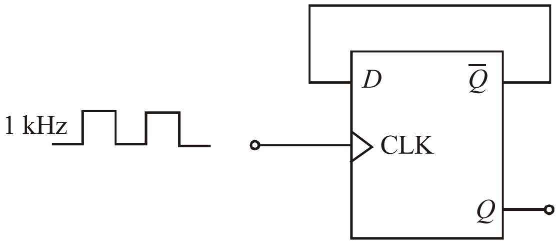


Fig. 5-50.

Solution : Given a D flip-flop whose output, $\overline{Q}$ is connected to the D input. The clock is 1 kHz waveform. The output waveform can be obtained from the input as shown in Fig. 5-51.

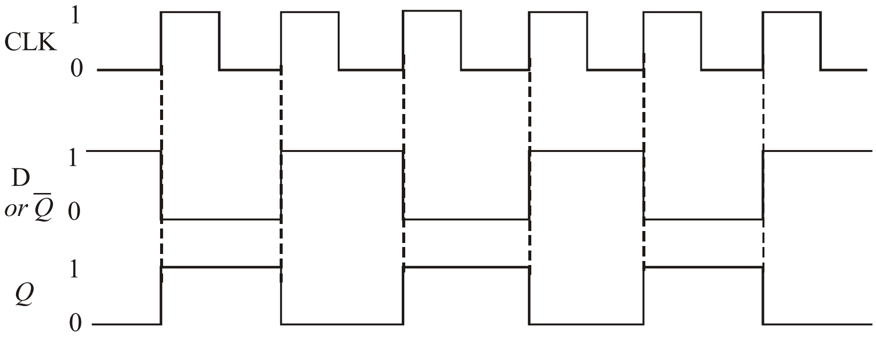


Fig. 5-51.

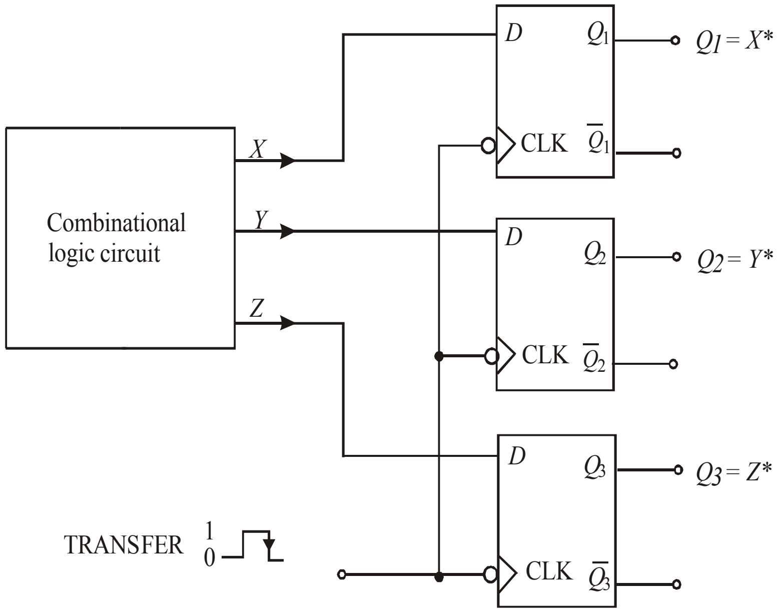
Notice that the output is a square wave of half-the frequency of input *i.e.* 500 kHz. **Ans.**

Note. A D- flip-flop with $\overline{Q}$ tied to its D – input as shown in Fig 5-50 is also known as T flip-flop where T stands for Toggle (or trigger).

5-23. Parallel Transfer of Data Using D-Flip-Flops

We have already discussed in Art 5-15 that the Q-output of a D flip-flop is the same as the D input. Let us now study the usefulness of this flip-flop.

Fig. 5-52 shows an application of D flip-flop used for parallel transfer of binary data from *X*, *Y*, *Z* — the three outputs of a combinational logic circuit to the outputs Q_1, $Q_{2,}$ and Q_3 of the *D* flip-flops for storage. The transfer occurs upon application of TRANSFER pulse to the common CLK inputs. The flip-flops can store these values for subsequent processing.

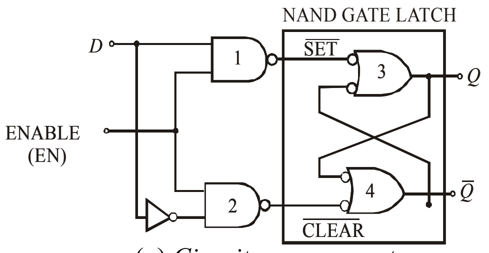


*After occurrence of falling edge

Fig. 5-52. *Illustrating parallel transfer of data using D flip-flops*

5-24. D Latch (Transparent Latch)

We have already discussed in Art. 5-15 about an edge – triggered D flip-flop. Such a flip-flop uses an edge-detector circuit to ensure that the output will respond to the D input only when the active transition of the clock occurs. If this edge detector circuit is not used, the resultant circuit operates somewhat differently. It is called a D latch and has the arrangement shown in Fig 5-53 (a).



(*a*) *Circuit arrangement*

Inputs		***Output***
EN	***D***	***Q***
0	X	Q_0 (no change)
1	0	0
1	1	1

(*b*) *Truth table*

Fig. 5-53. *D latch*

As seen from Fig. 5-53. (a), the circuit contains a NAND gate latch, and the steering NAND gates without the edge-detector circuit. The common input to the steering gates is called an enable input (abbreviated as EN) rather than a clock input because its effect on Q and $\overline{Q}$ outputs is not restricted to occurring only on its transitions. The operation of the D latch may be explained as follows:

1. When EN is LOW, whatever be the value of D input, it is inhibited from affecting the NAND gate latch. It is due to the reason that the outputs of both the steering gates will be held HIGH. Thus the Q and $\overline{Q}$ outputs will stay at whatever level they had just before EN went LOW. In other words, the outputs are "latched" to their current level and cannot change while EN is LOW even if D changes. This is represented as Q_0 in the truth table. Refer to Fig. 5-53 (*b*).
2. When EN is HIGH, the D input will produce a LOW at either $\overline{\text{SET}}$ or the $\overline{\text{CLEAR}}$ inputs of the NAND gate latch to cause Q to become the same level as D. If D changes while EN is HIGH, Q will follow the change exactly. In other words, while EN = 1, the Q output will look exactly like D. Because of this reason, the D latch in this mode is called "transparent".

The operation is summarized in the truth table shown in Fig. 5-53 (b). The logic symbol for the D-latch is shown in Fig. 5-54. It may be noted that even though the EN input operates in the same way as CLK input of an edge-triggered flip-flop, there is no small triangle on the EN input. This is because the small triangle symbol is used strictly for inputs that can cause an output change only when an edge occurs. So remember the D latch is not an edge-triggered device.

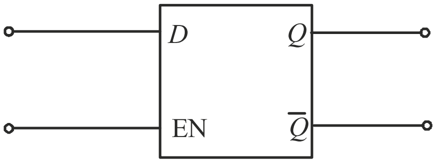


Fig. 5-54. *Logic symbol for a D-latch.*

IC 7475 is an example of D-latch. It contains four transparent D-latches. The pin configuration of IC 7475 is given in Appendix C (C-I).

Example 5-10. *Fig.* 5-55 *shows the waveforms applied at the D and EN inputs of a D latch.*

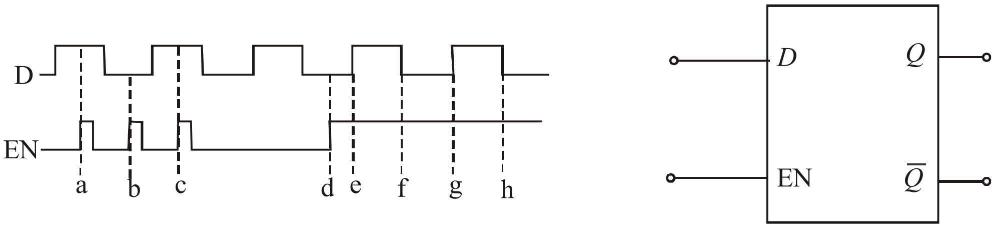


Fig. 5-55.

Sketch the output waveform at Q-output Assume initiually Q = 0

Solution. Given the waveforms applied at D and EN inputs of D-latech shown in Fig. 5-55.

Using the truth table of D-latch shown in Fig. 5-53, and the input waveforms, the output waveform at Q can be obtained and is sketched as shown in Fig. 5-56. Notice that at point 'a', EN goes 1, and D is 1, therefore a output goes 1. At point 'b' EN is 1, but D = 0, therefore Q goes 0, At point 'c', EN is 1, therefore Q goes 1 and so on.

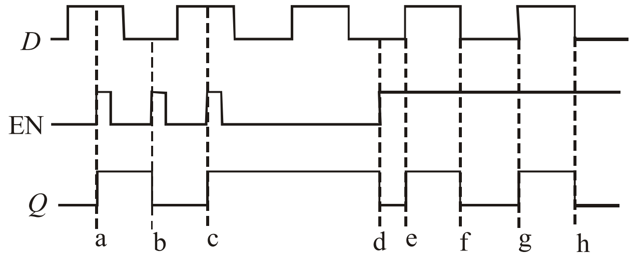


Fig. 5-56.

2-25. Master-Slave Flip-Flop

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a master-slave flip-flop. The logic diagram of an S-C master-slave flip-flop is shown in Fig. 5-57. It consists of a master flip-flop, a slave flip-flop, and an inverter.

When clock pulse CLK is 0, the output of the inverter is 1. Since the clock input of the slave is 1, the flip-flop is enabled and output *Q* is equal to *Y*, while *Q*′ is equal to *Y*′. The master flip-flop is disabled because CLK = *O*. When the pulse becomes 1, the information then at the external C and S inputs is transmitted to the master flip-flop. The slave flip-flop, however, is isolated as long as the pulse is at its 1 level, because the output of the inverter is 0. When the pulse returns to 0, the master flip-flop is isolated, this prevents the external inputs from affecting it. The slave flip-flop then goes to the same state as the master flip-flop.

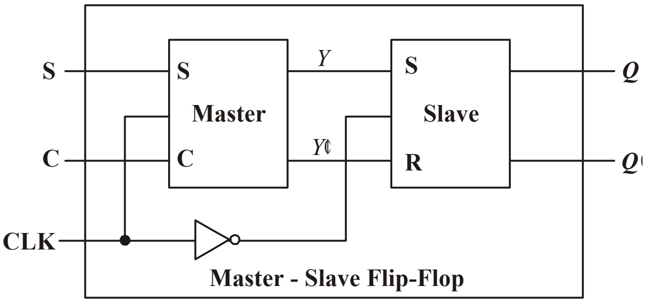
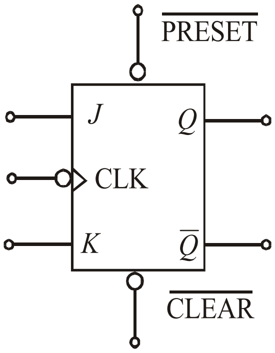


Fig. 5-57.

5-26. Clocked J-K Flip-Flop with Asynchronous Inputs

Strictly speaking for the clocked flip-flops, the S, C, J, K and D inputs discussed in the previous articles are refered to as *control inputs.* These inputs are also known as *synchronous inputs.* These are called synchronous because their effect on the flip-flop output is synchronized with the CLK (*i.e.* clock) input. As discussed earlier, the synchronous control inputs must be used in conjunction with a signal to trigger the flip-flop.

As a matter of fact, most clocked flip-flops also have one-or more *asynchronous* inputs. These inputs operate independently of the synchronous inputs and clock input. The asynchronous inputs of a flip-flop can be used to set its output to 1 state or clear to the 0 state at any time regardless of the conditions at the other inputs. In other words, the asynchronous inputs can be used to *override* all the other inputs in order to place the flip-flop output in one state or the other. Because of this reason, the asynchronous inputs are also called *override inputs.*



(*a*) Symbol

Inputs		*Output / Flip-flop response*
$\overline{PRESET}$	$\overline{CLEAR}$	
1	1	Clocked operation*
0	1	Q = 1 (regardless of CLK)
1	0	Q = 0 (regardless of CLK)
0	0	Not used

(*b*) Truth table

Fig . 5-58.

Fig. 5-58 shows a clocked J-K flip-flop with two asynchronous inputs namely (1) $\overline{\text{PRESET}}$ and (2) $\overline{\text{CLEAR}}$. Both these inputs are active-LOW. This is indicated by the bubbles on the flip-flop symbol. Fig 5-58 (b) shows the truth table for the clocked J-K flip-flop with asynchronous inputs. Let us study the various cases given in the truth table :

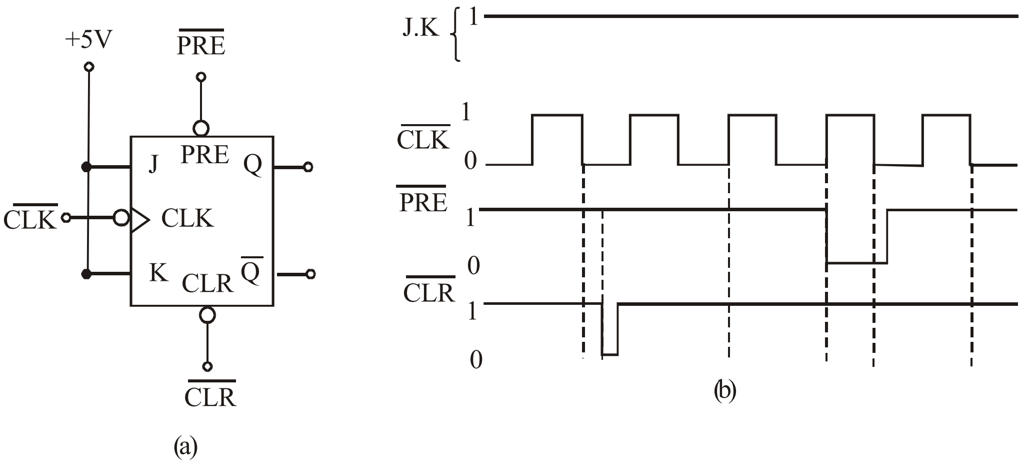
1. **$\overline{\textbf{PRESET}}$ = $\overline{\textbf{CLEAR}}$ = 1.** This condition indicates that both the asynchronous inputs are inactive and the flip-flop is free to respond to the *J*, *K* and CLK inputs. In other words, the flip-flop operates as a normal clocked flip-flop.
2. **$\overline{\textbf{PRESET}}$ = 0, $\overline{\textbf{CLEAR}}$ = 1.** This condition indicates that the asynchronous input, $\overline{\text{PRESET}}$ is activated. Because of this, the output, *Q* is immediately set to 1, no matter what conditions are present at the *J*, *K* and CLK inputs. It may be carefully noted that the CLK input cannot affect the flip-flop while $\overline{\text{PRESET}}$ = 0.
3. **$\overline{\textbf{PRESET}}$ = 1, $\overline{\textbf{CLEAR}}$ = 0.** This condition indicates that the asynchronous input, $\overline{\text{CLEAR}}$ is activated. Because of this, the output, *Q* is immediately set to 0, no matter what conditions are present at the J, K and CLK inputs. It may be carefully noted that the CLK input cannot affect the flip-flop while $\overline{\text{CLEAR}}$ = 0.
4. **$\overline{\textbf{PRESET}}$ = $\overline{\textbf{CLEAR}}$ = 0.** This condition should not be used because it can produce an ambiguous response.

It will be interesting to know that the $\overline{\text{PRESET}}$ and $\overline{\text{CLEAR}}$ inputs respond to dc levels. This means that if a constant 0 is held on the $\overline{\text{PRESET}}$ input, the flip-flop will remain in the *Q* = 1 state regardless of what is occurring at the other inputs. Similarly, a constant 0 at the $\overline{\text{CLEAR}}$ input holds the flip-flop in *Q* = *0* state. Thus the asynchronous inputs can be used to hold the flip-flop in a particular state for any desired interval of time. However, in actual practice, the asynchronous inputs are used to set or clear the flip-flop to the desired state by application of a momentary pulse.

Many clocked flip-flops that are commercially available as ICs, will have both $\overline{\text{PRESET}}$ and $\overline{\text{CLEAR}}$. However some ICs will have only $\overline{\text{CLEAR}}$ input. Some other ICs will have asynchronous inputs that are active-HIGH. For these ICs, the flip-flop symbol would not have a bubble on the asynchronous inputs.

ICs 7476 and 74LS76 are popular J-K flip-flops with asynchronous inputs preset and clear. The preset is designated by $\overline{\text{S}}_\text{D}$ and clear by $\overline{\text{R}}_\text{D}$. Refer to the pin configuration diagram shown in Appendix C (C-I). The IC 7476 is a rising edge triggered while 74LS76 is a falling edge triggered device. Each package contains two J-K flip-flops.

Example 5-11. Fig. 5-59 (*a*) *shows the logic symbol for a J-K flip-flop that responds to the falling edge on its clock pulse and has active-LOW asynchronous inputs. The J and K inputs are tied HIGH.*



(a)

(b)

Fig. 5-59.

Determine the Q–output in response to the waveforms shown in Fig. 5-59 (*b*). *Assume that initially Q* = 0.

Solution. Given : The J-K flip-flop with asynchronous inputs. Notice that the J and K are tied to +5V. The waveforms shown in Fig. 5-59 (*b*) are applied at the inputs of the J-K flip flop.

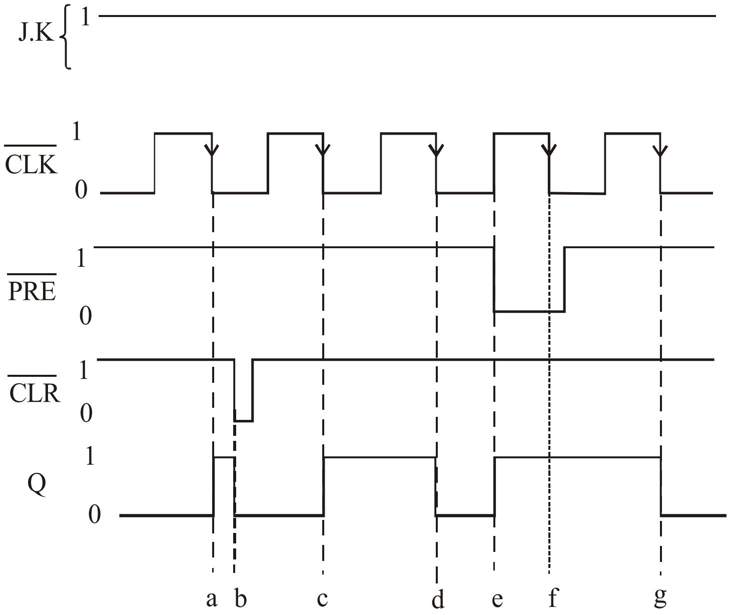


Fig. 5-60.

In order to sketch the Q-output waveform, recall the truth table of J-K flip-flop indicated in Fig. 5-31 (*b*) and Fig. 5-58 (*b*). Notice that when $\overline{PRE}=1$ and $\overline{CLR}=1$, the J-K flip-flop will trigger at the falling edge of the clock pulse. Thus at point 'a', $\overline{PRE}=\overline{CLR}=1$, J = K = 1, therefore the Q-output toggles i.e. it goes from 0 to 1. At point 'b' $\overline{\text{CLR}}$ goes 0, therefore Q goes 0. At point 'c', $\overline{\text{PRE}}=\overline{\text{CLR}}=1$, therefore under normal operation J-K flip-flop toggles i.e. Q goes from 0 to 1 again. At point 'd' there is no change in $\overline{\text{PRE}}$ or $\overline{\text{CLR}}$ inputs. So the Q-output toggles again, i.e. Q goes 0. At point 'e' $\overline{\text{PRE}}$ goes 0, therefore Q-output goes 1. At point 'g', $\overline{\text{PRE}}=\overline{\text{CLR}}=1$, and Q toggles to 0. The complete Q-output waveform is shown in Fig. 5-60 along with the input waveforms.

Example 5-12. *Determine the Q-output for the JK flip-flop shown in Fig.* 5-61, *Assume that* $Q = 0$ *initially and remember that the asynchronous inputs override all other inputs.*

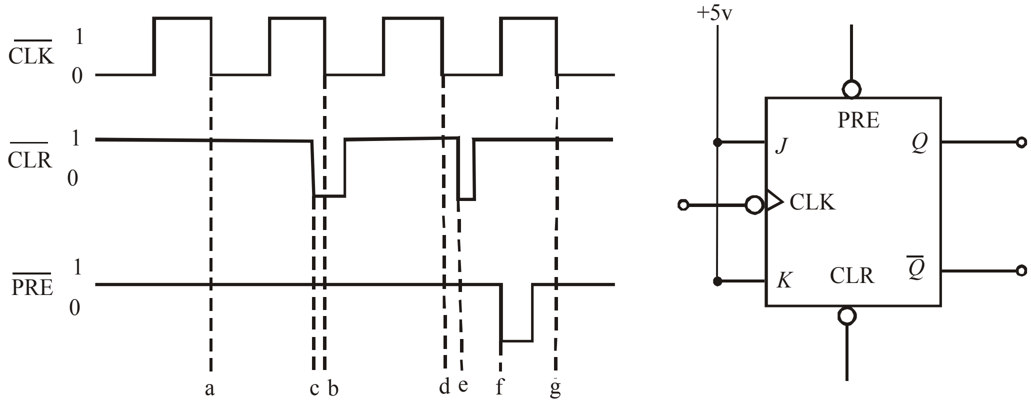


Fig. 5-61.

Solution : Given, the J K flip-flop with the waveforms at CLK, PRE and CLR inputs. Also $J = K = 1$.

The waveform at the Q-output can be obtained by recalling the truth-table of a JK flip-flop. Shown in Fig. 5-31 (*b*) and 5-58 (*b*).

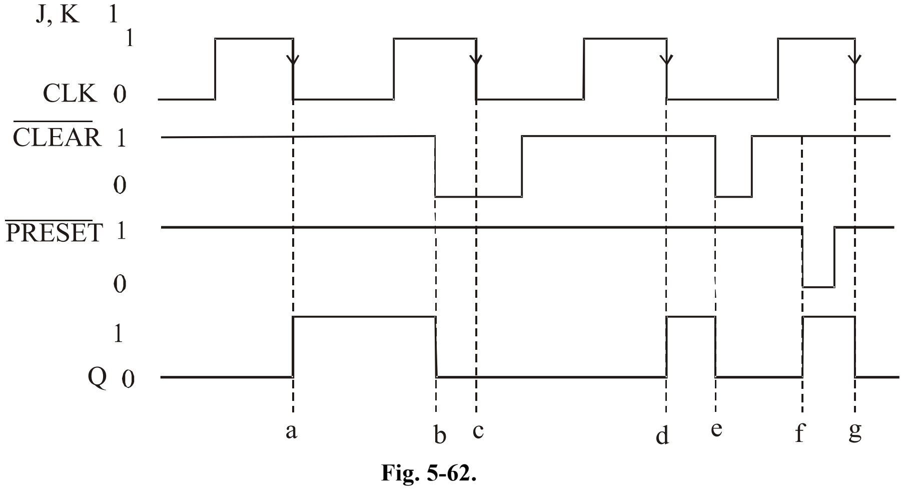


Fig. 5-62.

Notice that at point 'a', J = K = 1, $\overline{\text{PRESET}}=\overline{\text{CLEAR}}=1$, the falling edge of CLK pulse toggles the J-K flip-flop Q output to 1. At point 'b' $\overline{\text{CLEAR}}$ goes 0, therefore irrespective of the other inputs, Q-output goes 0. At point 'c' the ouput remains 0 because $\overline{\text{CLEAR}}$ is still 0. At point 'd', $\overline{\text{PRESET}}=\overline{\text{CLEAR}}=1$, $J=K=1$, the J-K flip-flop output toggles on the arrival of falling edge of CLK, i.e. Q goes 1. At point 'e', $\overline{\text{CLEAR}}$ goes 0 again, therefore Q-output goes 0. At point 'f', $\overline{\text{PRESET}}$ goes 0, therefore Q-output goes 1. Finally at point 'g', $\overline{\text{CLEAR}} = \overline{\text{PRESET}} = 1$, J = K = 1, the output toggles on the arrival of falling edge of CLK pulse, i.e. Q goes 0.

The complete sketch of Q-output waveform along with other input waveforms is shown in Fig. 5-62.

Example 5-13. *In the circuit of Fig.* 5-63, *inputs A, B and C are all initially LOW. Output Y is supposed to go HIGH only when A, B and C goes HIGH in a certain sequence.*

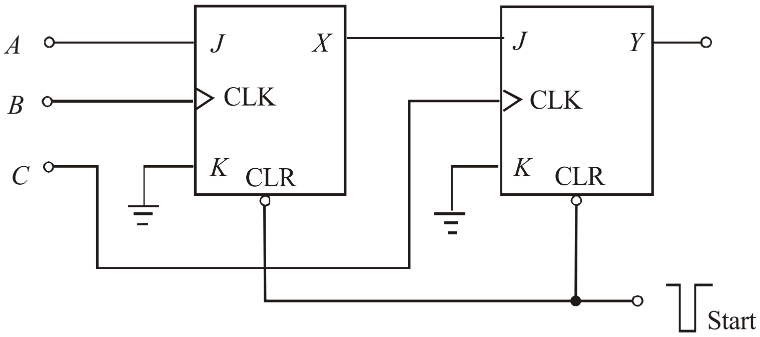


Fig. 5-63.

(*a*) *Determine the sequence that will make Y go HIGH.*

(*b*) *Explain why the START pulse is needed.*

Solution. Given : The circuit using J K flip-flops as shown in Fig. 5-63.

(*a*) **Sequence at the inputs (A, B and C)**

We know that Y can go HIGH only when C goes HIGH while X is already HIGH.

X can go HIGH only if B goes HIGH while A is HIGH.

Thus the correct sequence that makes Y go HIGH is A, B, C. **Ans.**

(*b*) **Need for START pulse**

We know that the outputs X and Y need to be cleared to 0 before applying the A, B, C signals. To clear the outputs, we need a negative going START pulse at CLR input. (notice that CLR input of JK flip-flop is active LOW).

5-27. Alternative Designations for Asynchronous Inputs

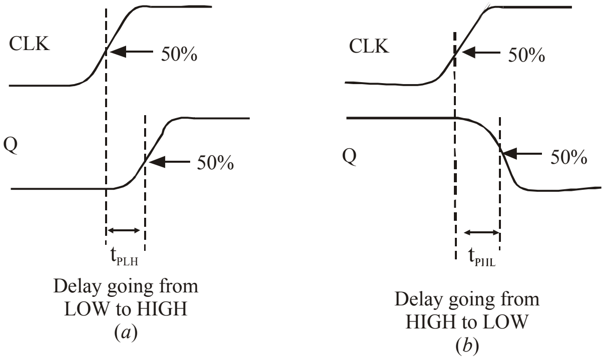
We have already discussed in the last article about the asynchronous clocked J-K flip-flop. These flip-flops are available as ICs and have two asynchronous inputs namely $\overline{\text{PRESET}}$ and $\overline{\text{CLEAR}}$ (or PRESET and CLEAR). IC manufacturers have not agreed on what nomenclature is used for these asynchronous inputs. The most common designations are $\overline{\text{PRE}}$ (short for $\overline{\text{PRESET}}$) and $\overline{\text{CLR}}$ (short for $\overline{\text{CLEAR}}$). The designations $\overline{S}_D$ (direct $\overline{\text{SET}}$) and $\overline{R}_D$ (direct $\overline{\text{RESET}}$) are also used. However we will use the labels $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ to represent asynchronous inputs. Whenever these asynchronous inputs are active–HIGH, we will not use the overbar to indicate their active-HIGH status i.e. PRE and CLR.

5-28. Flip-Flop Timing Parameters

The performance, operating requirements and limitations of flip-flops are specified by several timing parameters found on the data sheet for the device refer to Appendix A. Generally the timing parameters are applicable to all CMOS and TTL flip-flops.

1. Setup and Hold Times. These have already been discussed in Art 5-10 (page 323). They represent requirements that must be met for reliable flip-flop triggering. The manufacturer's IC data sheet will always specify the minimum values of t_S and t_H . For example, IC 7474 has set up time of 20 ns and hold time of 5 ns. On the other hand 74HC112 has set up time of 25ns and zero hold time.

2. Propagation delay. It is the interval of time required after an input signal has been applied for the resulting output change to occur. Fig. 5-64 (a) illustrates the propagation delays that occur in response to a rising edge of the CLK input when *Q*-output changes from 0 to 1. Similarly Fig. 5-64 (*b*) illustrates the propagation delay that occur in response to the rising edge of the CLK input when *Q*-output changes from 1 to 0. It may be noted that these delays are measured between the 50% points (between logic 0 and logic 1 voltage levels) on the input and output waveforms. The same type of delays occur in response to the signals on a flip-flops asynchronous inputs (*i.e.* PRESET and CLEAR). The manufacturer's data sheets usually specify propagation delays in response to all inputs and they usually specify the maximum values for t_{PLH} and t_{PHL} . Notice that t_{PLH} is the delay going from logic LOW to HIGH level whereas t_{PHL} is the delay going from logic HIGH to LOW level.



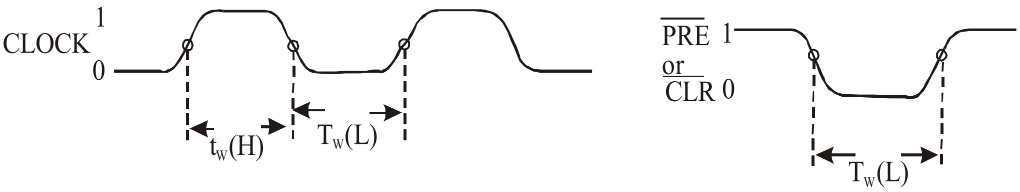
Delay going from LOW to HIGH (*a*)

Delay going from HIGH to LOW (*b*)

Fig. 5-64.

3. Maximum Clock Frequency (f_{max}). This is the highest frequency that may be applied to the CLK input of a flip-flop and still have it triggered reliably. The value of f_{max} limit will vary from flip-flop to flip-flop even with flip-flops having the same device number. For example, f_{max} for IC7474 is 15 MHz, for 74LS112, it is 30 MHz, for 74C74, it is 5 MHz and for 74HC112, it is 20 MHz.

4. Clock Pulse HIGH and LOW Times. The manufacturer will also specify the minimum time duration that the CLK signal must remain LOW before it goes HIGH (represented as t_w (L)) and the minimum time that CLK must be kept HIGH before it returns LOW (represented as t_w (H)). These



(*a*) *Clock Pulse HIGH-LOW times*

(*b*) *Asynchronous active pulse width*

Fig. 5-65.

times are illustrated in Fig. 5-65 (a) Failure to meet these minimum time requirements can result in unreliable triggering. It may be noted that these time values are measured between the halfway (50%) points on the signal transitions.

5. Asynchronous Active Pulse Width. It is the minimum time duration that a PRESET or CLEAR input must be kept in its active state in order to set or clear the flip-flop reliably. Fig. 5-65 (b) shows t_W (L) for active-LOW asynchronous inputs.

6. Clock Transition Times. Strictly speaking, the rise and fall times of a clock waveform should be kept very short for reliable triggering. If the clock signal takes too long to change from one logic level to the other, the flip-flop may trigger erratically or not at all. IC manufacturers usually do not list a maximum transition time requirement for each flip-flop integrated circuit. Instead, it is usually given as a general requirement for all ICs within a given logic family. For example, the transition times should generally be ≤ 50 ns for TTL devices and ≤ 200 ns for CMOS. These requirements will vary among the different manufacturers and among the various subfamilies within the broad TTL and CMOS logic-families.

5-29. IC Flip-Flop Timing Values

We have already discussed the various flip-flop timing parameters in the last article. Table 5-1 lists all these timing values for the various flip-flops such as 7474, 74 LS112, 74C74 and 74 HC112. Notice that 7474 is a dual edge triggered D flip-flop (Standard TTL) device whereas 74LS74 is a dual edge-triggered J-K flip-flop (low-power schottky TTL) device. Similarly 74C74 is a dual edge-triggered D flip-flop (metal-gate CMOS) device whereas 74HC112 is a dual edge-triggered J-K flip-flop (high-speed CMOS) device.

Table 5-1. IC flip-flop timing values

Parameter	*TTL*		*CMOS*	
(Times in ns)	*7474*	*74LS112*	*74C74*	*74HC112*
t_S	20	20	60	25
t_H	5	0	0	0
t_{PHL} from *CLK* to *Q*	40	24	200	31
t_{PLH} from *CLK* to *Q*	25	16	200	31
t_{PHL} from $\overline{CLR}$ to *Q*	40	24	225	41
t_{PLH} from $\overline{PRE}$ to *Q*	25	16	225	41
t_W(L) *CLK* LOW time	37	15	100	25
t_W(H) *CLK* HIGH time	30	20	100	25
t_W(L) at $\overline{PRE}$ or $\overline{CLR}$	30	15	60	25
f_{MAX} in MHz	15	30	5	20

Following are some of the important points which may be noted carefully from Table 5-1.

1. The 74HC series of CMOS devices has timing values that are comparable to those of TTL devices. This can be observed by reading the values below the 7474 column and the values below 74HC112 column.
2. The 74C series devices are much slower than the 74HC series. This can be observed by comparing all the minimum timing requirements below the 74C74 and 74HC112 column.

3. All the flip-flops have nonzero set up time requirement.
4. All of the flip-flops have very low hold-time ($t_{\overline{H}|}$) requirements e.g. The IC7474 has $t_H = 5$ ns whereas IC74LS112, 74C74 and 74HC112 has $t_H = 0$.

Example 5-14. *The data sheet of a certain flip-flop specifies that the minimum HIGH time $t_W(H)$ for the clock pulse is* 30 *ns and the minimum LOW time $t_W(L)$ is 37 ns. What is the maximum operating frequency*?

Solution Given : $t_W(H) = 30$ ns $= 30 \times 10^{-6}$ s and $t_W(L) = 37$ ns $= 37 \times 10^{-6}$ s.

We know that the minimum time period,

$$t_{min} = t_w\ (H) + t_w\ (L)$$

$$= \left(30\times10^{-6} + 37\times10^{-6}\right) = 47\times10^{-6}\,\text{s}$$

and the maximum operating frequency,

$$f_{max} = \frac{1}{t_{min}} = \frac{1}{47\times10^{-6}\,ns} = 21.276\times10^{6}\ Hz$$

$= 21.276$ *MHz.* **Ans.**

Example 5-14(*a*). *Using the flip-flop timing values shown in Table* 5-2, *determine the followings*:

Table 5-2.

Parameter	***TTL***		***CMOS***	
(time in ns)	***7474***	***74LS112***	***74C74***	***74HC112***
t_{PHL} from *CLK* to *Q*	40	24	200	31
t_{PLH} from *CLK* to *Q*	25	16	200	31
t_W(L) at $\overline{PRE}$ or $\overline{CLR}$	30	15	60	25

(*a*) *Assume that initially Q = 0. How long it can take for Q to go HIGH when a rising edge appears at the CLK input of* IC7474?

(*b*) *Assume that initially Q = 1. How long it can take for Q to go LOW in response to the* $\overline{\text{CLR}}$ *input of an* IC 74HC112?

(*c*) *What is the narrowest pulse that should be applied to the* $\overline{\text{CLR}}$ *input of the* IC 74LS112 *flip-flop to clear Q reliably?*

Solution. Given : The flip-flop timing values : t_{PHL}, t_{PLH} and t_W (L) of TTL and CMOS flip-flop devices.

(*a*) The rising edge will cause the *Q* output to go from LOW to HIGH. The delay from CLK to Q is listed as t_{PLH} = 25 ns for IC7474. **Ans.**

(*b*) For the IC 74HC112, the time required for *Q* to go from HIGH to LOW in response to $\overline{\text{CLR}}$ input is listed as $t_{PHL} = 31$ ns. **Ans.**

(*c*) For the IC 74HC112, the narrowest pulse at the $\overline{\text{CLR}}$ input is listed as $t_W(L) = 15$ ns. **Ans.**

5-30. Potential Timing Problems in Flip-Flop Circuits

As a matter of fact, in many digital circuits, the output of one flip-flop is connected either directly or through logic gates to the input of another flip-flop and both flip-flops are triggered by the same clock signal. This situation leads to a potential timing problem. In order to understand this problem, consider a situation shown in Fig. 5-66. Here the output of flip-flop (1) Q_1 is connected to the J input of flip-flop (2). Notice that both the flip-flops are clocked by the same signal at their CLK inputs.

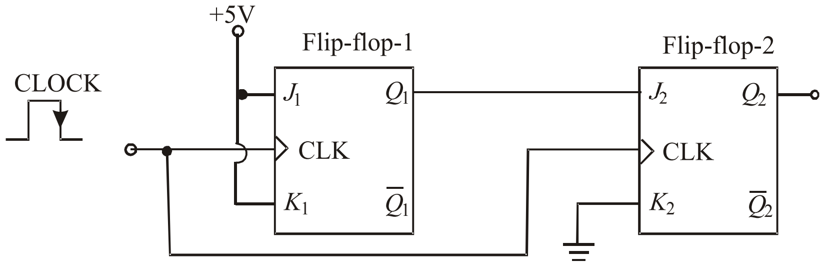


Fig. 5-66.

Let us understand the potential timing problem now. Since Q_1 will change on the rising edge of the clock pulse, the J_2 input will be changing as it receives the same rising edge. This could lead to an unpredictable response at Q_2.

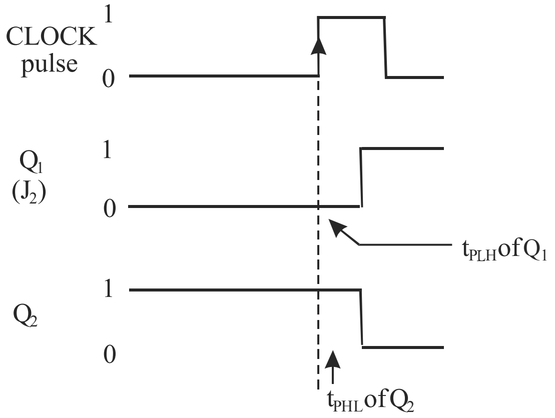


Fig. 5-67.

Assume that initially $Q_1 = 0$ and $Q_2 = 1$. Further the flip-flop (1) has $J_1 = K_1 = + 5V$ (i.e HIGH) and flip-flop (2) has $J_2 = Q_1$ and $K_2 = 0$ prior to the occurrence of rising edge of the clock pulse. When the rising edge occurs, Q_1 will toggle to the HIGH state but it will not actually go HIGH until after its propagation delay, t_{PLH} as shown in Fig. 5-67. The same rising edge will reliably clock Q_2 to the LOW state provided that t_{PLH} is greater than Q_2's hold time requirement, t_H as shown in the figure. If this condition is not met, the response of Q_2 will be unpredictable.

Since all edge-triggered flip-flops have hold time (t_H) requirement that is 5 ns or less. Therefore for those flip-flops, situations like that shown in Fig. 5-67 will not be a problem.

Thus in all of the flip-flop circuits that you will study in this book, we will assume that flip-flop's hold time requirement is short enough to respond reliably according to the following rule:

The flip-flop output will go to a state determined by the logic levels present at its synchronous control inputs just prior to the active clock edge.

Applying the above stated rule to the circuit shown in Fig. 5-67, it says that the output of flip-flop (2), Q_2 will go to a state determined by $J_2 = 1$ and $K_2 = 0$ condition that is present just prior to the occurrence of the rising edge of the clock pulse. The fact that J_2 is changing in response to the same rising edge has no effect.

5-31. Applications of Flip-Flop

Although there is a wide variety of applications of edge-triggered (clocked) flip-flops, yet the following are important from the subject point of view :

1. Flip-flop synchronization
2. Data storage and transfer
3. Serial data transfer-shift registers
4. Frequency division
5. Counting

All these applications are discussed one by one in the following pages.

5.32. Flip-Flop Synchronization

Strictly speaking, most digital systems are principally synchronous in their operation. It means, most of the signals will change states in synchronism with the clock transitions. However, in many cases, there will be an external signal that is not synchronised to the clock. In other words such an external signal is an asynchronous signal. Such signals often occur as a result of a human operator actuating an input switch at some random time relative to the clock signal. This randomness can produce unpredictable and undesirable results. Now we will study how a flip-flop can be used to synchronize the effect of an asynchronous input.

Fig. 5.68 shows a situation where a signal X is generated from a debounced switch that is actuated by an operator. The switch output A goes high when the operator actuates the switch and goes LOW when the operator releases the switch. The switch output X is used as an input to control the passage of the control signal through the AND gate so that clock pulses appear at output Y only as long as X is HIGH.

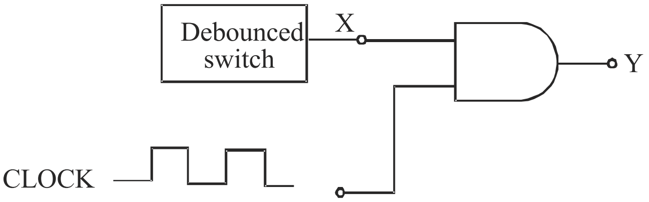


Fig. 5-68.

The problem with the circuit shown in Fig. 5-68 is that signal X is asynchronous. That is it can change states at any time relative to the clock signal because the exact times when the operator actuates or releases the switch are essentially *random*. This can produce partial clock pulses at output *Y* if either transition of *X* occurs while the clock signal is HIGH. It is shown in Fig. 5-69. This type of output is often not acceptable. So a method for preventing the appearance of partial pulses at *Y* must be developed. One possible solution is shown in Fig. 5-70.

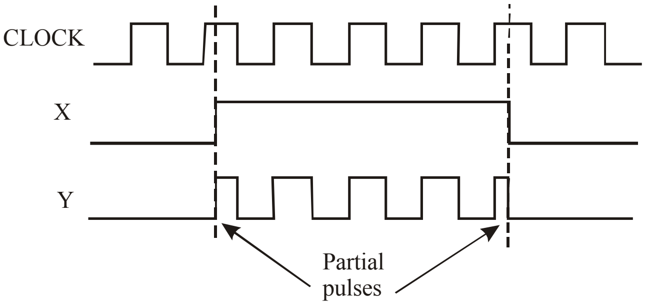


Fig. 5-69.

As seen in Fig. 5-70, the X signal is connected to the D input of a flip-flop. The flip-flop is clocked by the falling edge of the clock signal. Thus when *X* goes HIGH, *Q* will not go HIGH until the next falling edge of the clock at time t_1. This HIGH at *Q* will enable the AND gate to pass subsequent complete clock pulses to *Y* as shown in Fig. 5-69.

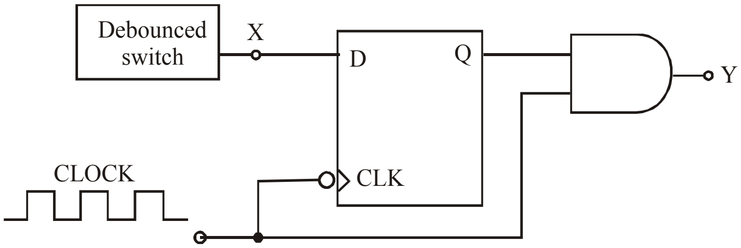


Fig. 5-70.

Similarly, when *X* goes LOW, *Q* will not go LOW until the next falling edge of the clock at t_2. Thus the AND gate will not inhibit clock pulses until clock pulse that ends at t_2 has passed through to *Y*. Therefore, output *Y* contains only complete pulses (see Fig. 5-71).

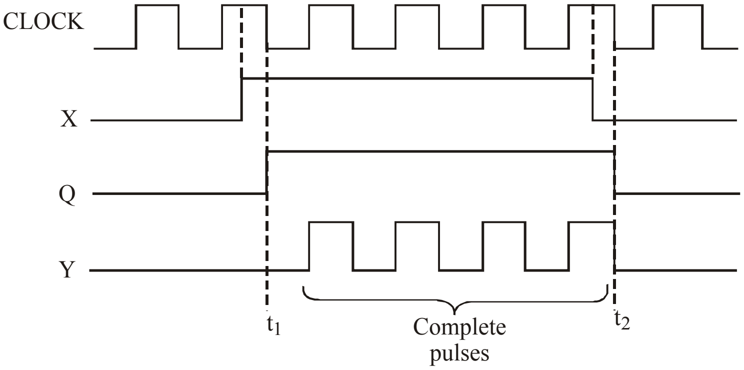
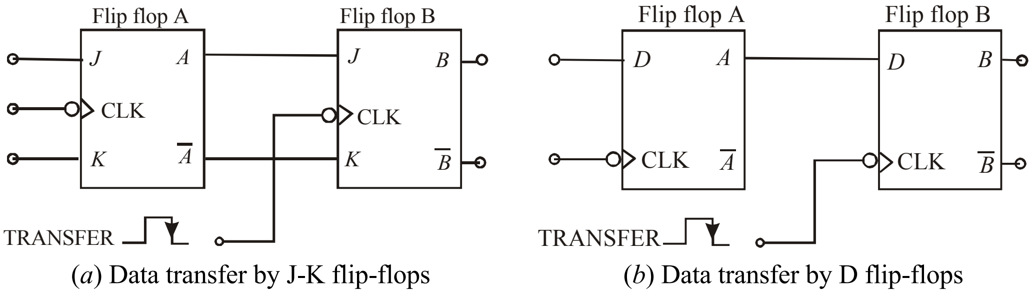


Fig. 5-71.

5-33. Data Storage and Transfer

The most common use of flip-flops in the field of digital electronics, is for the storage of data or information. The data may represent numerical values (e.g. binary numbers, BCD – binary coded decimal numbers or any data that have been encoded in binary. These data are generally stored in groups of flip-flops called *registers.* The operation most often performed on data that are stored in a flip-flop or a register is the *data transfer* operation. This operation involves the transfer of data from one flip-flop (or register) to another. The data transfer is of two types: (*a*) synchronous and (*b*) asynchronous.



(*a*) Data transfer by J-K flip-flops (*b*) Data transfer by D flip-flops

Fig. 5-72.

Fig. 5-72 (a) and (b) shows how synchronous data transfer operation can be accomplished between two flip-flops using J-K and D flip-flops respectively. In each case, the logic value that is currently stored in flip-flop A is transferred to flip-flop B upon the falling edge of the TRANSFER pulse. Thus after this falling edge, the B output will be the same as the A output.

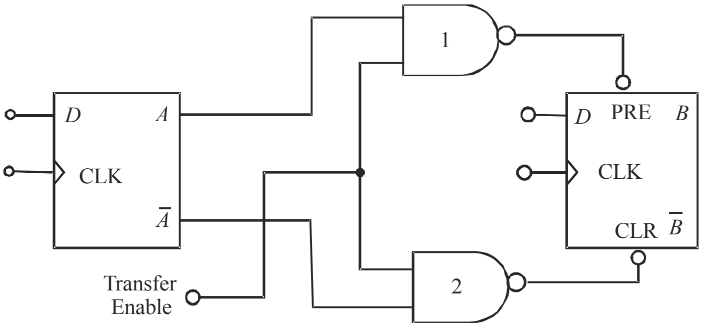


Fig. 5-73. *Illustrating asynchronous data transfer operation*

The data transfer illustrated in Fig. 5-72 is called synchronous transfer because the synchronous control and CLK inputs of a flip-flop are used to perform the transfer operation. Fig. 5-73 shows the data transfer that, can be obtained using the asynchronous inputs of a flip-flop. This method called asynchronous transfer can be accomplished using the PRESET and CLEAR inputs of any flip-flop. Notice that flip-flop outputs A and $\bar{A}$ are connected to PRESET and CLEAR through two –input NAND gates 1 and 2 respectively.

In this method, the asynchronous inputs (*i.e.* PRESET and CLEAR) respond to LOW levels. The operation of the circuit may be explained as follows :

When the *Transfer Enable* line is held LOW, the two NAND outputs are kept HIGH, with no effect on the flip-flop outputs. But when the *Transfer Enable* line is made HIGH, one of the NAND outputs will go LOW depending on the state of A and $\bar{A}$ outputs. This LOW will either set or clear the flip-flop B.

5-34. Parallel Data Transfer

We have already discussed in the last article that flip-flops can be used for data storage as well as for transfer of data from one flip-flop to another. This idea can be further extended for transfer of data from one group of flip-flops (called register A) to another group of flip-flops (called register B). Register A consists of four flip-flops A_1, A_2, A_3 and A_4. Similarly register B also consists of four flip-flops labelled as B_1, B_2, B_3 and B_4 upon application of falling edge of a TRANSFER pulse, the logic level stored in A_1 is transferred to B_1, A_2 is transferred to B_2, A_3 is transferred to B_3 and A_4 is transferred to B_4. It may be noted that the transfer of contents from register A to B occurs at the same time. Because of this reason it is called synchronous transfer or parallel data transfer.

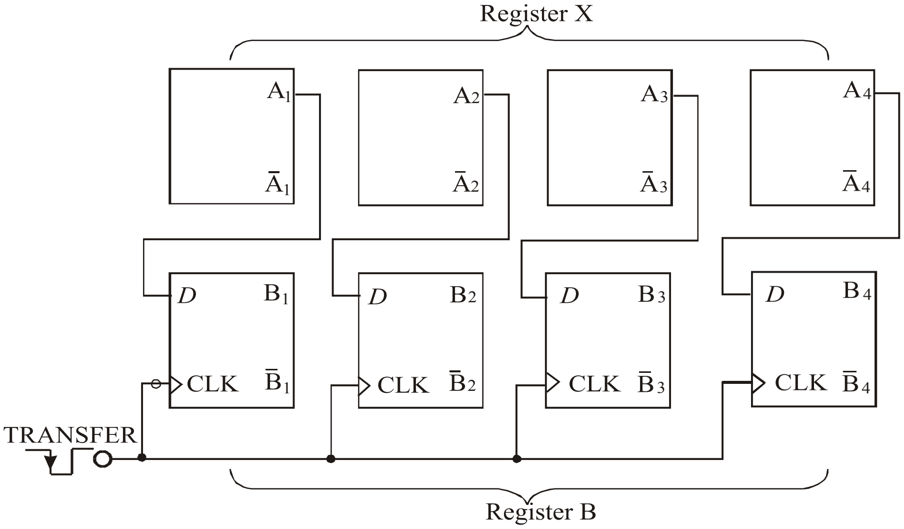


Fig. 5-74. *Illustrating parallel data transfer from register X into register Y.*

It may be carefully noted that parallel data transfer does not change the contents of the register that is the source of data. For example in Fig. 5-74 if $A_1 A_2 A_3 A_4$ = 1011 and $B_1 B_2 B_3 B_4$ = 0100 prior to the occurrence of the TRANSFER pulse, then both registers will be holding 1011 after the TRANSFER pulse. Another point is that the group of four flip-flops indicated above is an example of a basic register used for data storage. In digital systems, data are normally stored in groups of bits (usually 8, 16, 32, 64 ...) that represent numbers, codes or other information.

Example 5-15. (*a*) *Draw a circuit diagram for the synchronous parallel transfer of data from one three-bit register to another using J-K Flip-flops*. (*b*) *Repeat for asynchronous parallel transfer.*

Solution. (*a*) Fig. 5-75 shows the circuit diagram for the synchronous parallel transfer of data from one three-bit register X to another register Y using J-K flip-flops. Notice that the contents of X_1, X_2, and X_3 are transferred simultaneously into Y_1, Y_2, and Y_3. It may be carefully noted that the circuit allows the parallel transfer of data from the normal outputs as well as the complemented outputs of the flip-flops constituting register X .

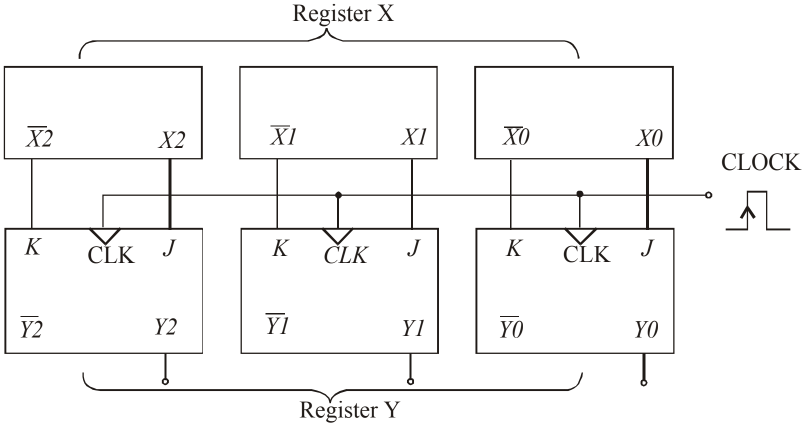


Fig. 5-75. *Snynchronous parallel transfer of data from one three-bit register to another*

(*b*) Fig. 5-76 shows the circuit diagram for the asynchronous parallel data transfer from one three-bit register to another using J-K flip-flops. The circuit allows data transfer from either normal outputs X_0, X_1, and X_2 or the complemented outputs $\overline{X_1}$, $\overline{X_2}$ and $\overline{X_3}$.

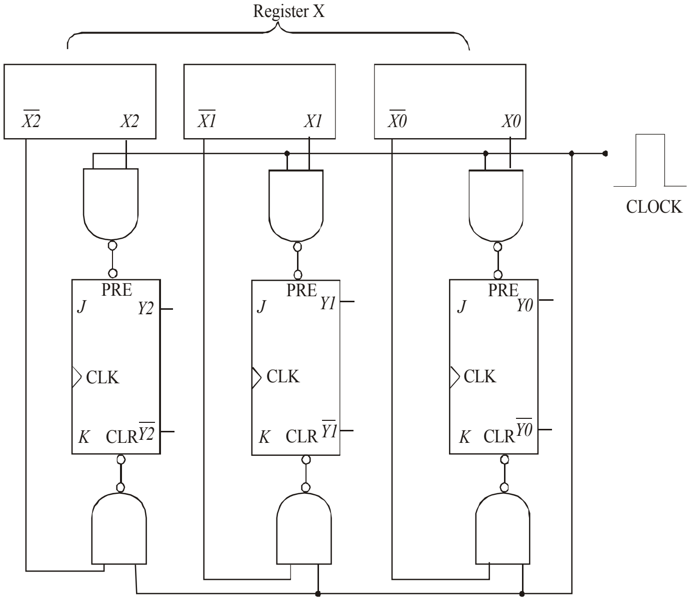


Fig. 5-76. *Asynchronous parallel transfer of data from one three-bit register to another.*

5-35. Serial Data Transfer : Shift Registers

We have already discussed in the last article that a *register* is basically a group of flip-flops to store data. But there is no interconnection between the flip-flops. A *shift register* is a group of flip-flops arranged in such a way that the binary numbers stored in the flip-flops are shifted from one flip-flop to the next for every clock pulse. Fig. 5-77 shows a four-bit shift register using J-K flip-flops. Notice that the J and K inputs of flip-flops '3' are fed by DATA IN waveform. The J and K inputs of flip-flop '2' are fed by the X_3 and $\overline{X_3}$. Similarly J and K inputs of flip-flop '1' are fed by the X_2 and $\overline{X_2}$ and J and K inputs of flip-flop '0' are fed by the X_1 and $\overline{X_1}$. The CLK input of all the four flip-flops are connected together to a common input which receives shift pulses. Incidently notice that all the flip-flops trigger on the falling edge of the CLK pulse.

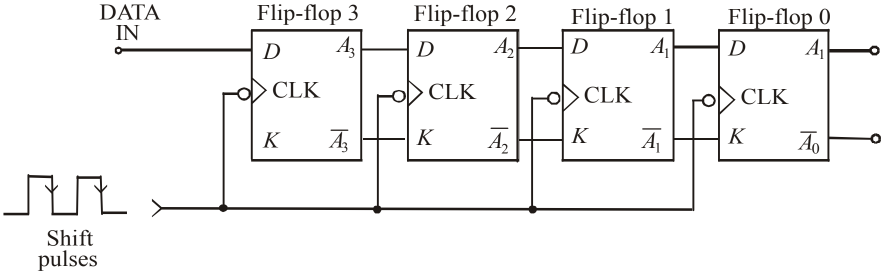


Fig. 5-77. *Four-bit shift register.*

The operation of the shift register may be understood from the explanation given below. Let us assume that initially all the flip-flops are in the 0 state and input of flip-flop '3' is fed by the DATA IN waveform as shown in Fig. 5-78. The other waveforms shown in Fig. 5-78 indicate how the input data are shifted from left to right from flip-flop to flip-flop as shift pulses are applied. When the first falling edge occurs at t_1, each of the flip-flop outputs A_2, A_1 and A_0 will have $D = 0$ condition present at its inputs because of the state of the flip-flop on its left. The flip-flop '3' will have $D = 1$, because of DATA-IN. Thus at t_1 only A_3 will go HIGH while all the other flip-flop outputs remain LOW. When the second falling edge occurs at t_2, flip-flop '3' will have $D = 0$ because of DATA-IN. The Flip-flop '2' will have $D = 1$ because of current HIGH at A_3 . Flip-flops '1' and '0' will still have $D = 0$. Thus at t_2, only flip-flop '2' output A_2 will go HIGH, flip-flop output A_3 will go LOW, and A_2 and A_0 will remain LOW.

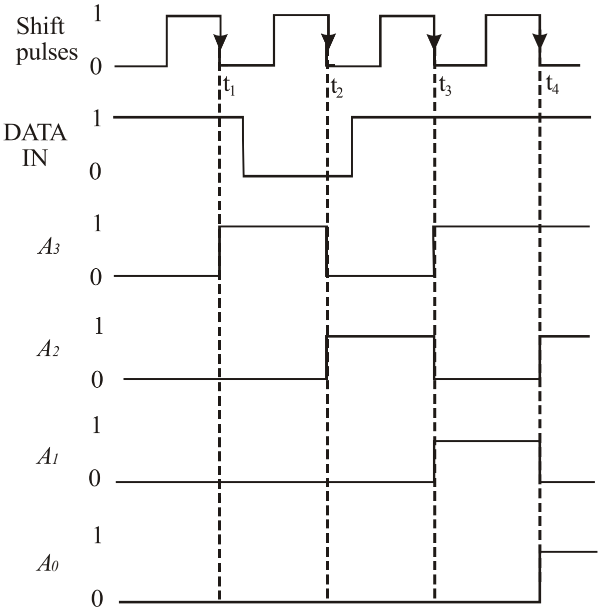


Fig. 5-78. *Illustrating serial data transfer.*

A similar reasoning can be used to determine how the waveforms change at t_3 and t_4. It may be carefully noted that on each falling edge of the shift pulses, each flip-flop output takes on the level that was present at the output of the flip-flop on its left just *prior* to the falling edge. Of course, flip-flop '3' output X_3 takes on the level that was present at DATA IN just prior to the falling edge.

Note. In the shift register arrangement discussed above, it is necessary that the flip-flops have a very small hold time requirement. This is because of the fact that there are times when the J and K inputs are changing at about the same time as the CLK edge. For example, the flip-flop '3' output, X_3 changes from 1 to 0 in response to the falling edge at t_2, causing the J and K inputs of flip-flop '2' output X_2 to change while its CLK input is changing. Actually because of the propagation delay of flip-flop '3', the J and K inputs of flip-flop '2' won't change for a short time after the falling edge. Because of this reason, a shift register should be implemented using edge-triggered flip-flops that have a t_H value less than one CLK-to-output propagation delay. This later requirement is easily satisfied by most modern edge triggered flip-flops.

5-36. Serial Data Transfer Between Registers

We have already discussed in the last article that a shift register is a group of flip-flops arranged in such a manner that the binary numbers stored in the flip-flops are shifted from one flip-flop to the next after every clock pulse. Now we will study as how data can be serially transferred (or shifted) from one register to the other.

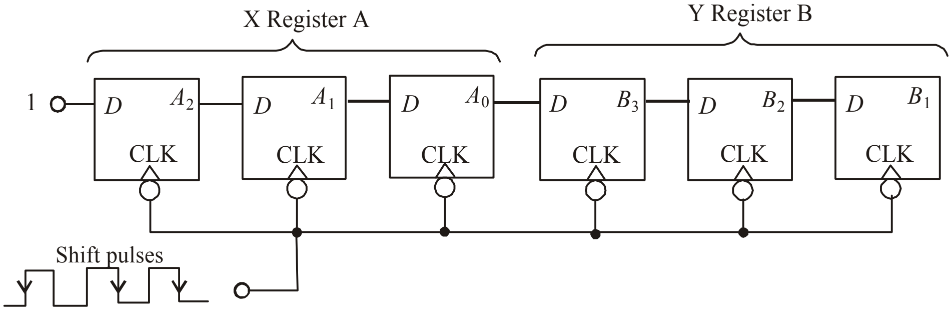
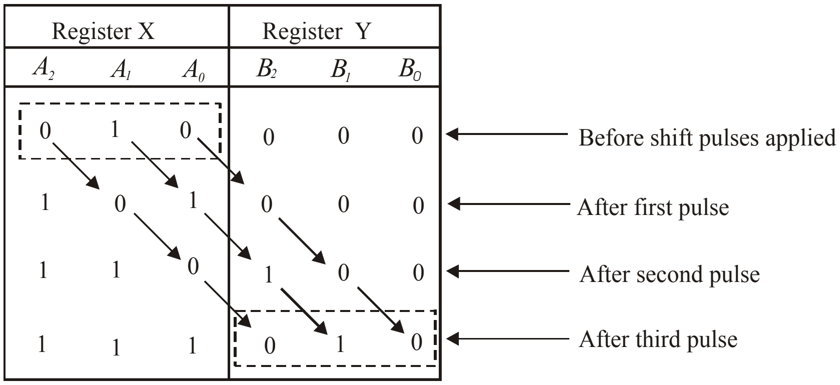


Fig. 5-79.

Fig. 5-79 shows three-bit shift registers connected in such a manner that the contents of the A-register will be serially transferred (shifted) into register B. Notice how A_0, output of last flip-flop of register A is connected to the D-input of the first flip-flop of register B. The CLK input of all the flip-flops is connected together at one point where the shift pulses are applied.



Register X			Register Y			
A_2	A_1	A_0	B_2	B_1	B_0	
0	1	0	0	0	0	Before shift pulses applied
1	0	1	0	0	0	After first pulse
1	1	0	1	0	0	After second pulse
1	1	1	0	1	0	After third pulse

Fig. 5-80.

As the shift pulses are applied, the data transfer takes place as follows:

$$A_2 \rightarrow A_1 \rightarrow A_0 \rightarrow B_2 \rightarrow B_1 \rightarrow B_0$$

The flip-flop output A_2 will go to a state determined by its D input. For discussion purpose let us suppose that D is held HIGH, so that A_2 will go HIGH on the first pulse and will remain there. Further let us assume that before any shift pulses are applied, the contents of A register are 010. (i.e. $A_2 = 0, A_1 = 1, A_0 = 0$) and the B register is at 000. As the shift pulse are applied one after the another, the states of each flip-flop output change are indicated in table as shown in Fig. 5-80. Following are some of the points which may be carefully noted from this table.

1. On the falling edge of each clock pulse, each flip-flop takes on the value that was stored in the flip-flop on its left prior to the occurrence of the pulse.
2. After first pulse, A_2 is set to 1, the 0 that was initially in A_2 is in A_1, the 1 that was initially in A_1 is in A_0, the 0 that was initially in A_0 is in B_2, the 0 that was initially in B_2 is in B_1, the 0 that was initially in B_1 is in B_0. The 0 that was initially in Y_0 is lost.
3. After second pulse, A_2 is still at 1 (because D is HIGH), the 1 that was previously in A_2 is in A_1, the 0 that was previously in A_1 is in A_0, the 1 that was previously in A_0 is in B_2, the 0 that was previously in B_2 is in B_1 and 0 that was previously in B_1 is in B_0.
4. After third pulse, A_2 is still at 1 (because D is held HIGH), the 1 that was previously in A_2 is in A_1, the 1 that was previous in A_1 is in A_0, the 0 that was previously in A_0 is in B_2, the 1 that was previously in B_2 is in B_1, the 0 that was previously in B_1 is in B_0.

It is evident from the above discussion that after these shift pulses, the 010 stored in the A register has now been shifted to B register. The register A is now at 111. Notice that it has lost its original data.

Notes.

1. The flip-flops shown in Fig. 5-79 can also be connected easily so that data shifts from right to left. As a matter of fact, there is no general advantage of shifting in one direction over the other. The direction chosen by the digital system designer depends upon the nature of application.
2. In serial data transfer (refer to Fig. 5-79) of N bits of data requires N clock pulses. This means three bits of data required three pulses, four bits requires four pulses and so on. However in parallel transfer, all the data is transferred simultaneously upon the occurrence of a single transfer pulse (refer to Fig. 5-74). In other words it does not matter how many bits are being transferred. It is obvious that parallel transfer is much faster than serial transfer using shift registers. However, parallel transfer requires more interconnections between the sending register (A) and receiving register (B) than does the serial transfer. This is an important consideration when the sending and receiving registers are at a distance from each other. The choice of either parallel or serial transmissions depends on the particular system application and specifications. In actual practice, a combination of two types is used to take advantage of the speed of parallel transfer and economy and simplicity of serial transfer.

5-37. Frequency Division

Many applications require frequency division. For example a quartz (or digital) watch makes use of a quartz crystal to generate a very stable oscillator frequency. This frequency is usually 1MHz or more. In order to advance the “seconds” display once every second, the oscillator frequency is divided by a value that will produce a very stable and accurate 1Hz output frequency. The frequency division can be achieved by using flip-flops.

Fig. 5-81 (*a*) shows a single J-K flip-flop connected to toggle (*i.e.* J = K = 1). When the clock pulses are applied at its CLK input, the Q output is a square wave with one-half the frequency of the clock input as shown in Fig. 5-81 (*b*). Thus a single flip-flop can be used as a divide-by-2 device. As seen from the figure, the flip-flop changes state on each falling edge of the clock pulse. This results in an output that changes at half the frequency of the clock waveform.

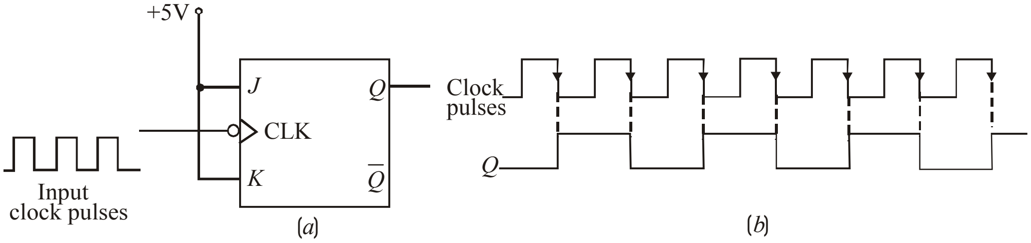
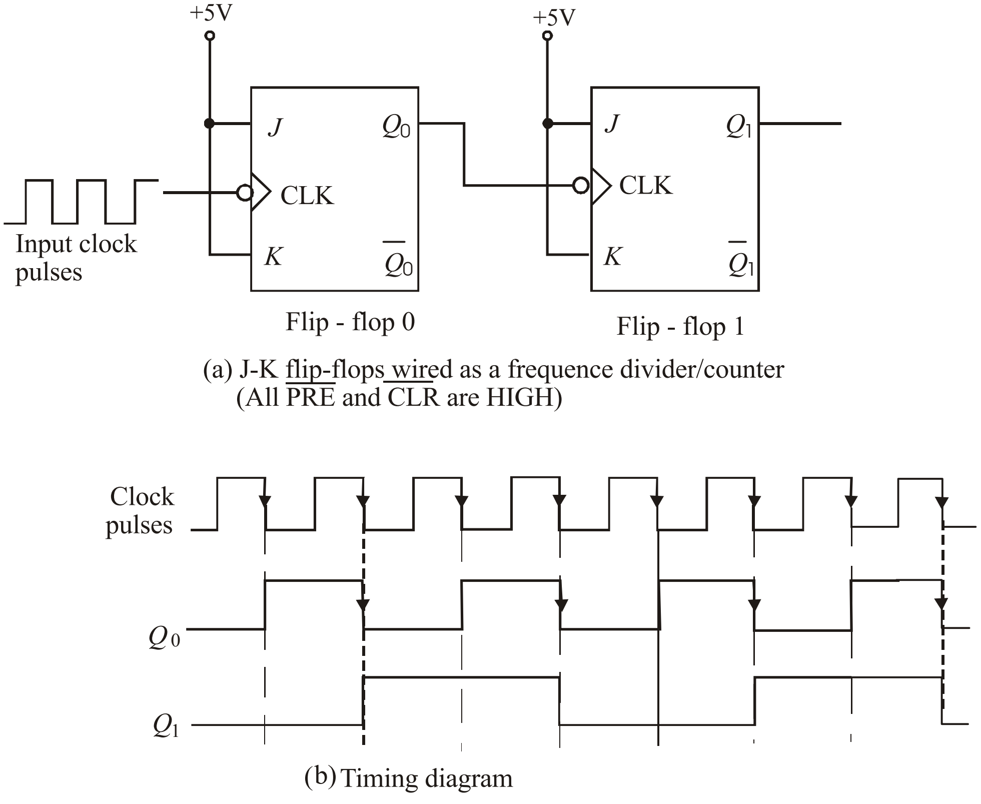


Fig. 5-81.

Further division of the clock frequency can be achieved by using the output of one flip-flop as the clock input to the second flip-flop as shown in Fig. 5-82. Notice that the frequency of Q_0 output is divided by 2 by flip-flop 1. Therefore the Q_1 output is one-fourth the frequency of the original clock input. It may be carefully noted that propagation delays are not shown on the timing diagrams.



(a) J-K flip-flops wired as a frequence divider/counter
(All $\overline{PRE}$ and $\overline{CLR}$ are HIGH)

(b) Timing diagram

Fig. 5-82.

By connecting the flip-flops in the way described above, a frequency division of 2^N is achieved, where N is the number of flip-flops. For example, three flip-flops divide the clock frequency by 2^3= 8, four flip-flops divide the clock frequency by 2^4=16 and so on.

Example 5-16. *How many flip-flops are required to divide a frequency by thirty-two.*

Solution. We know that a single flip-flop can divide the input frequency by 2. Two flip-flops connected together can divide the frequency by 4. In gereal a frequency division of 2^N can be achieved by using N number of flip-flops. Thus,

$$2^N = 32 \Rightarrow N = 5 \quad \textbf{Ans.}$$

Example 5-17. *Fig.* 5-83 *shows three J-K flip-flops wired as frequency divider.*

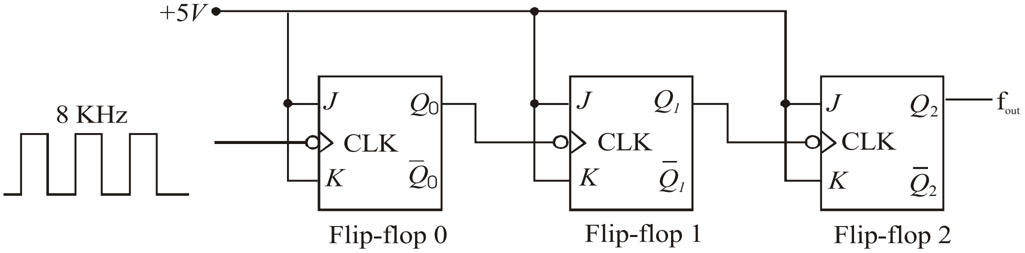


Fig. 5-83.

Sketch the frequency waveform at Q_2 output when an 8 kHz square wave input is applied at CLK input of the flip-flop 'D'.

Solution. Given the circuit of Fig. 5-83 where three J-K flip-flops are wired as a frequency divider. Note that each flip-flop is connected to toggle (i.e. *J = K = 1*). Since these are falling edge triggered flip-flops, the outputs change on the falling edge of clock pulse as shown in Fig. 5-84. There is one output pulse at Q_2 for every eight input pulses, so the output frequency is f_{out} = *8 kHz /8 = 1 kHz.* Fig 5-84 shows the waveforms at Q_0 and Q_1 outputs also.

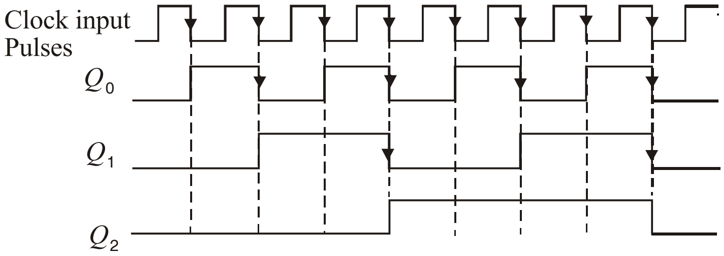


Fig. 5-84.

5-38. Counting

Another important application of flip-flops is in digital counters. These are covered in more detail in chapter 7 on counters and Registers Fig 5-85 illustrates the concept of a 2-bit counter. Here two J-K flip-flops are wired to toggle (i.e. *J = K = 1*). Both the flip-flops are initially RESET. Flip-flop 0 toggles on the falling edge of each clock pulse. The Q-output of flip-flop 0 clocks flip-flop 1, so that each time Q_0 makes a HIGH-to-LOW transition, flip-flop 1 toggles. The resulting Q_0 and Q_1 waveforms are shown in the figure.

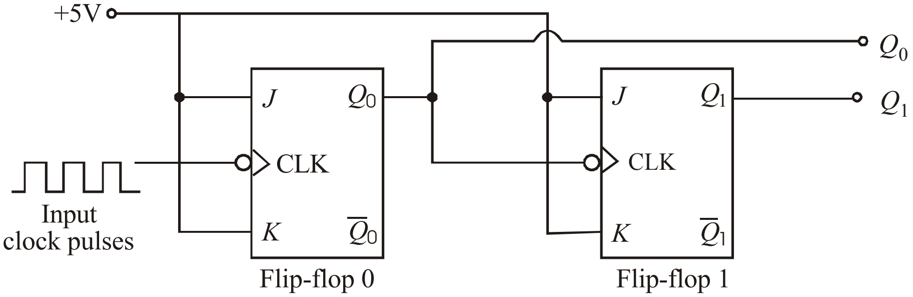


Fig. 5-85. *J-K flip-flops wired as a 2-bit counter*

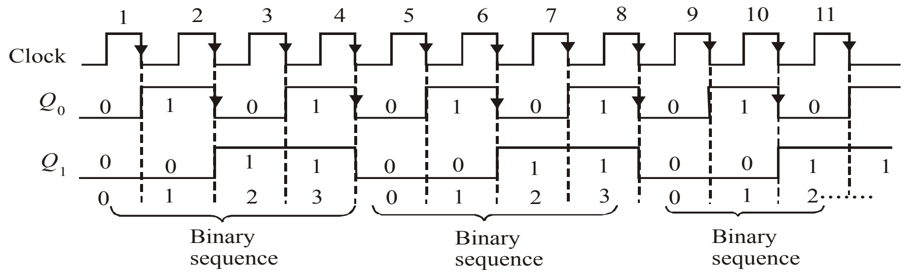


Fig. 5-86.

Examine the sequence of Q_0 and Q_1 in Fig. 5-86. Prior to clock pulse 1, $Q_0 = 0$ and $Q_1= 0$, after clock pulse 1, $Q_0= 1$ and $Q_1= 0$, after clock pulse 2, Q_0= 0 and Q_1= 0, and after clock pulse 3, $Q_0= 1$ and $Q_1= 1$. If we take Q_0 as the least significant bit, a two bit sequence is produced as the flip-flops are clocked. This binary sequence repeats every four clock pulses as shown in the timing diagram of Fig. 5-86. Thus flip-flops are counting in sequence from 0 to 3 (i.e. 00, 01, 10 and 11) and then recycling back to 0 to begin the counting sequence again.

Example 5-18. *Fig.* 5-87 *shows a J-K flip-flops wired as a* 3-*bit counter.*

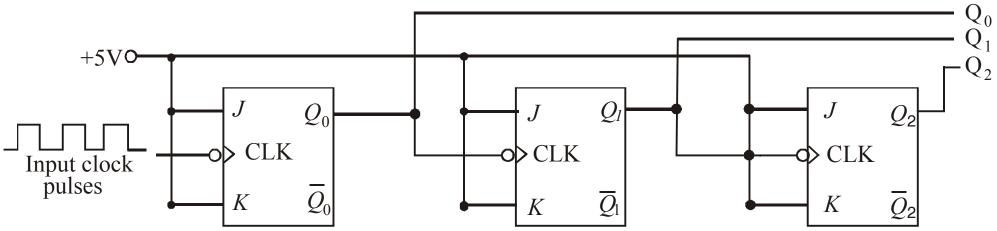


Fig. 5-87.

Sketch the output waveforms at Q_0, Q_1 and Q_2. Also show the binary sequence represented by these waveforms.

Solution. Fig. 5-88 shows the output waveforms at Q_0, $Q_1$and Q_2. Notice that the outputs change on the falling edge of the clock pulses. The outputs go through the binary sequence 000, 001, 010, 011, 100, 101, 110 and 111 as indicated.

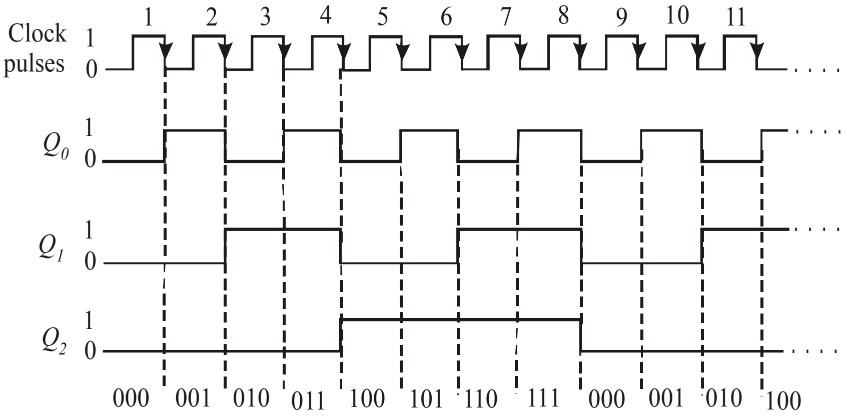
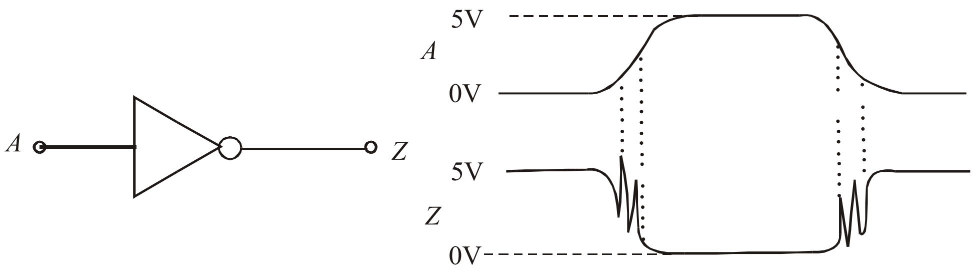


Fig. 5-88.

5-39. Schmitt-Trigger Devices

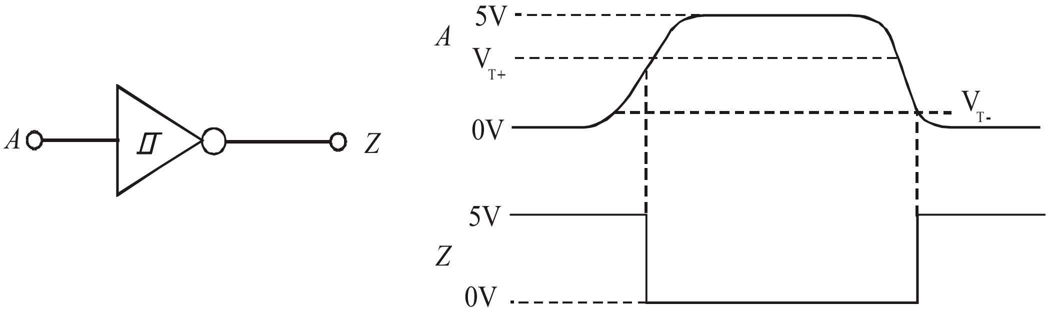
Strictly speaking, a Schmitt –trigger circuit is not classified as a flip-flop but it does exhibit a type of memory characteristic that makes it useful in certain special situations. One of those situations is shown in Fig. 5-89 (a) . Here a standard INVERTER is being driven by a logic input that has relatively slow transition times. When these transition times exceed the maximum allowed values, the outputs of logic gates and INVERTERS may produce oscillations as the input signal posses through the indeterminate range (refer to Fig. 5-89 (b)). The same input conditions can also produce erratic triggering of flip-flops.



(*a*) *Standard INVERTER* (*b*) *Waveforms*

Fig. 5-89.

A device that has a Schmitt-trigger type of input is designed to accept slow changing signals and produce an output that has oscillation free transitions. The output will generally have a very rapid transition times (typically 10ns) that are independent of the input signal characteristics. Fig. 5-90 (a) shows the symbol for a Schmitt-trigger INVERTER and (b) shows its response to a slow-changing input.



(*a*) *Schmitt-trigger INVERTER* (*b*) *Waveforms*

Fig. 5-90.

As seen from Fig. 5-90 (b), we find that the output does not change from HIGH to LOW until the input exceeds the rising-edge threshold voltage, V_{T+}. Once the output goes LOW, it will remain there even when the input drops back below V_{T+} (this is its memory characteristic) until it drops all the way down below the falling edge threshold voltage, V_{T-}. The values of the two threshold voltages will vary from one logic family to another. But V_{T-} will always be less than V_{T+}.

It is evident from the above discussion that schmitt trigger is a special device that is used to transform slowly changing waveforms into sharply defined, jitter-free output singals. They are useful for changing clock edges that may have slow rise and fall times into straight vertical edges.

There are several ICs with Schmitt-trigger inputs. The 7414, 74LS14 and 74HC14 are hex INVERTER ICs with Schmitt-trigger inputs. On the other hand 7413, 74LS13 and 74HC13 are dual four-input NANDs with Schmitt-trigger inputs. The pin configuration for 7414 is shown in Append x C (C-1).

5-40. One-Shot (Monostable Multivibrator)

A one-shot (abbreviated as OS) is also referred to as monostable multivibrator. It is a digital circuit that is somewhat related to a flip flop. For instance, like a flip-flop, the one-shot has two inputs, Q and $\overline{Q}$ which are inverse (or complement) of each other. However, unlike the flip-flop, the one-shot has only one stable output state (normally, $Q = 0$ and $\overline{Q} = 1$). The one-shot remains in this stable state until it is triggered by an input signal. Once triggered, the one-shot outputs switch to the opposite state ($Q = 1, \overline{Q} = 0$). This state is called quasi-stable (i.e. unstable) state. The one-shot remains in this quasi-stable state for a fixed period of time, t_p, which is determined by the RC time constant. The RC time constant in turn is determined by the values of external components connected to one-shot. After a time, t_p, the one-shot outputs return to their resting state until triggered again.

Fig. 5-91 (a) shows the logic symbol for one-shot. As seen from this figure, it has one input labeled as trigger input (T), a normal output (Q), and an inverted output ($\overline{Q}$). It needs two external components: resistor (R_T) and a capacitor (C_T) for its operation. The values of R_T and C_T determines the exact value of time period (t_p) for which the one-shot switches to its quasi stable state. The value of tp can vary from several nanoseconds to several tens of seconds. Fig. 5-91 (b) shows the stable state (where $Q = 0, \overline{Q} = 1$) and the quasi-stable state (where $Q = 1$ *and* $\overline{Q} = 0$).

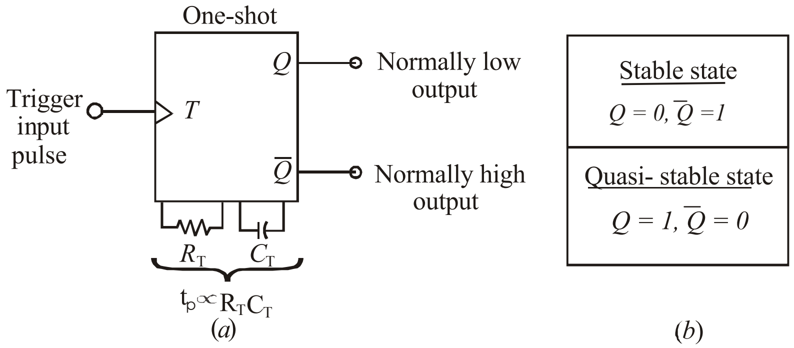


Fig. 5-91. *One-shot symbol and Output states*

5-41. Types of One-Shots

Depending upon the operation, the one-shot is of the following two types :

1. Non-retriggerable one-shot and 2. Retriggerable one-shot

Both the type of one-shots are available in the IC form. Now we shall study both these one-shots one by one in the following pages.

5-42. Non-Retriggerable One-shot

We have already discussed in the last article that one-shot (or a monostable multivibrator) is a device with only one stable state. It is normally in its stable state and will change to its quasi-stable state only when triggered. Once it is triggered, the one-shot remains in its quasi-stable state for a fixed period of time and then automatically returns to its stable state. The time that the device remains in the quasi-stable state determines the pulse width (t_p) of the output.

Fig. 5-92 (a) shows the waveforms of a trigger input signal T and output of one-shot. Notice that the rising edge of the trigger signal will trigger the one-shot to its quasi-stable state. The one-shot will remain in the quasi-stable state for a time, t_p after which it automatically returns to its stable state.

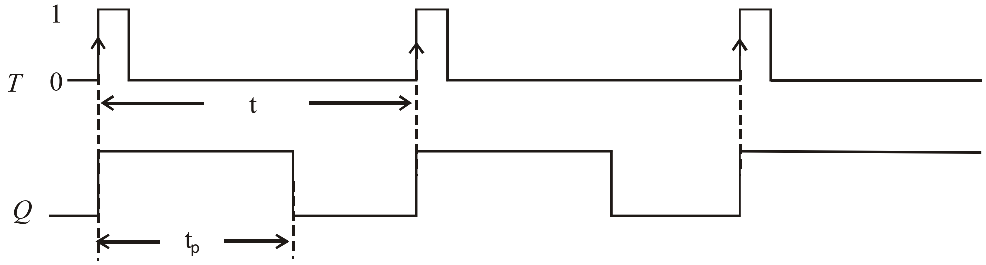


Fig. 5-92.

It may be noted from the diagram shown above that one-shot is being triggered at intervals greater than its pulse width (i.e. $t > t_p$).

Now we will consider a situation where the one-shot is being triggered at intervals less than its pulse width (i.e. $t < t_p$). Refer to Fig. 5-93. Notice that the rising edges of the T signal at points 'b' has no effect on the one-shot because it has already been trigged to the quasi-stable state by the rising edge at point 'a'. The one-shot must return to the stable state before it can be triggered. The same argument is valid for the rising edge at point 'd'. It has no effect on the one –shot because it has already been trigged by the rising edge at point 'c'.

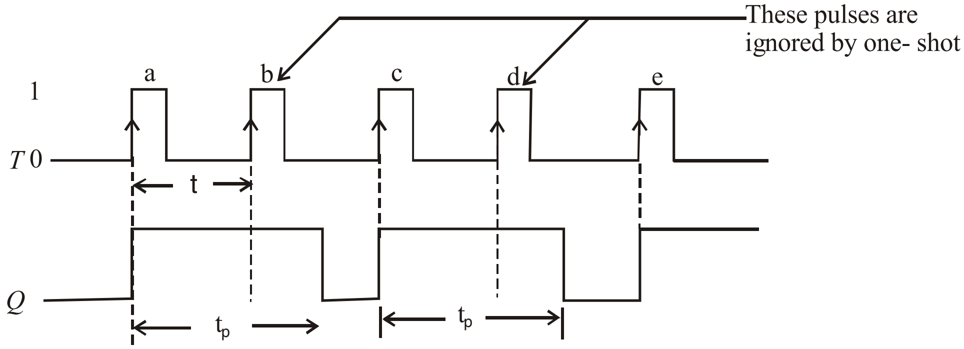


Fig. 5-93.

It may be carefully noted that one-shot output pulse duration (t_p) is always the same regardless of the input pulses. The pulse duration t_p, depends only on the values of external components R_T and C_T and the internal one-shot circuitry. In a typical monoshot, the value of t_p is given by the relation.

$$t_p = 0.7\, R_T C_T$$

Example 5-19. *Fig.* 5-94 *shows three non-retriggerable one-shots connected in a timing chain that produces three sequential output pulses. Note the* "1" *in front of the pulse on each one-shot symbol to indicate non-retriggerable operation.*

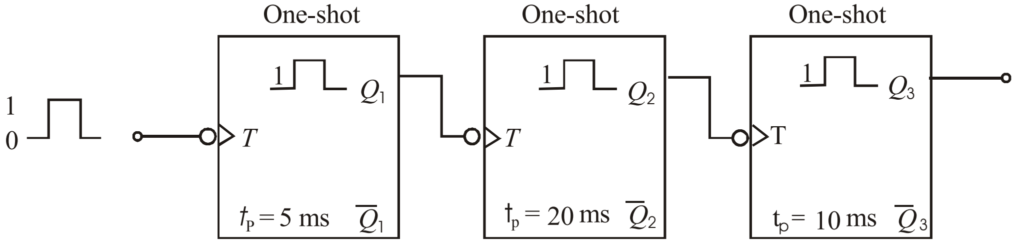


Fig. 5-94.

Draw a timing diagram showing the relationship between the input pulse and the three one-shot outputs. Assume an input pulse duration of 10 *ms.*

Solution. Given : An input pulse duration = 10 ms. Notice that all the one-shots trigger at the falling edge of the input signal. To begin with, the first one-shot triggers at the falling edge of the input signal and the second one-shot triggers at the falling edge of Q_1. The third one-shot triggers at the falling edge of Q_2.

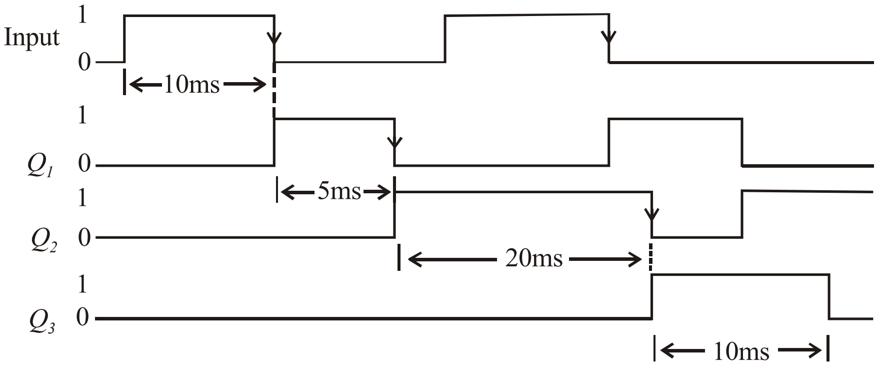


Fig. 5-95.

Fig. 5-95 shows a sketch of the timing waveforms at Q_1, Q_2 and Q_3 along with the input trigger waveform.

Example 5-20. *Sketch the waveforms at Q and $\overline{Q}$ outputs of a one-shot which is being triggered by a signal shown in Fig.* 5-96. *Assume that one-shot triggers at the rising edge of the trigger signal and has a t_p =* 1.5 *ms.*

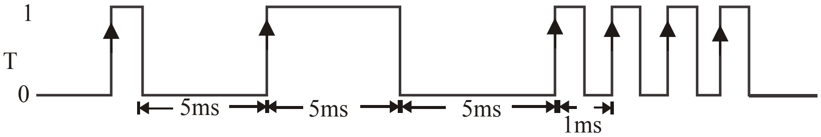
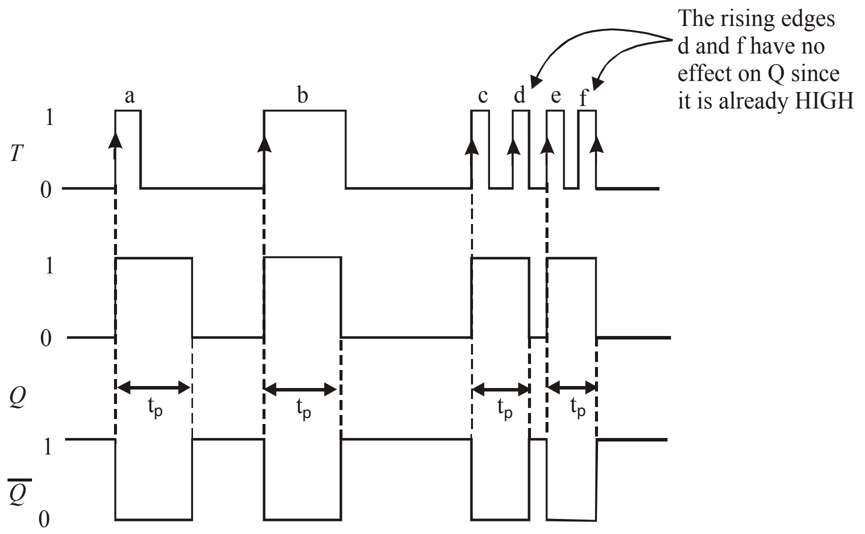


Fig. 5-96.

Solution Given : The trigger signal (T) at the input of one-shot. Fig. 5-97 shows the waveforms at the one-shot outputs Q and $\overline{Q}$ in response to the trigger signal (T). Notice that the rising edges at points a, b, c and e of the T signal, will trigger one-shot to its quasi-stable state. The one-shot will remain in the quasi-stable state for a time, t_p after which it automatically returns to its stable state.



The rising edges d and f have no effect on Q since it is already HIGH

Fig. 5-97.

The rising edges of the T signal at points ‘d’ and ‘f’ have no effect on the one-shot because it has already been triggered to the quasi-stable state. The one-shot must return to the stable state before it can be triggered.

5-43. Retriggerable One-Shot

This one-shot usually operates in the same manner as the non-retriggerable one-shot except for one-major difference. The major difference is that retriggerable one-shot can be retriggered while it is in the quasi-stable state, and it will begin a new t_p interval. Fig. 5-98 compares the response of both types of one-shot using a t_p of 2 ms interval.

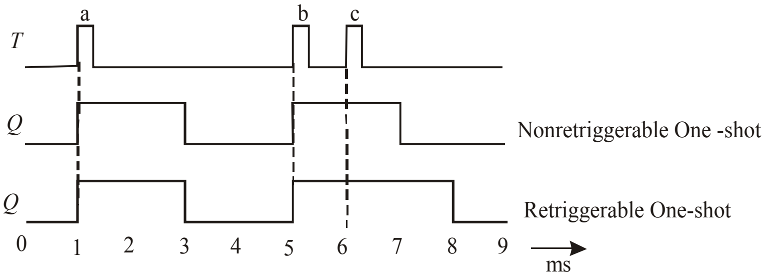


Fig. 5-98.

As seen from the diagram, both types of one-shot respond to the first trigger pulse indicated as ‘a’ at $t = 1$ ms by going HIGH for 2 ms and then returning LOW. The second trigger pulse indicated as ‘b’ at $t = 5$ ms triggers both one – shots to the HIGH state. The third trigger pulse indicated as ‘c’, at t = 6 ms has no effect on the non-retriggerable one-shot because it is already in the quasi-stable state. However, this trigger pulse will retrigger the retriggerable one-shot to begin a new $t_p = 2$ ms internal. Thus it will stay HIGH for 2 ms after this third trigger pulse.

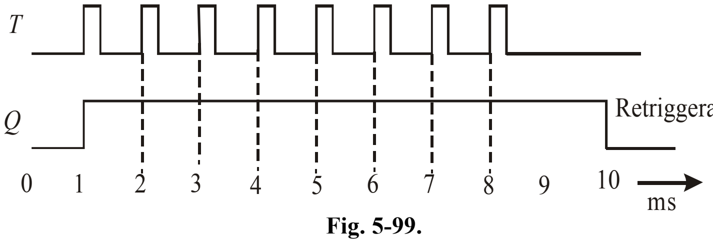


Fig. 5-99.

It is evident from the discussion above that a retriggerable one-shot begins a new t_p interval each time a trigger pulse is applied, regardless of the current state of its Q – output. As a matter of fact, trigger pulses can be applied at a rate fast enough that one-shot will always be retriggered before the end of the t_p interval and Q–output will remain HIGH. Such a situation is shown in Fig. 5-99. As seen from this diagram, eight pulses are applied every 1 ms. The Q – output does not return LOW until 2 ms after the last trigger pulse.

Example 5-21. *A retriggerable one-shot can be used as a pulse-frequency detector that detects when the frequency of a pulse input is below a predetermined value. A simple example of this application is shown in Fig.* 5-100. *The operation begins by momentarily closing switch SW*1.

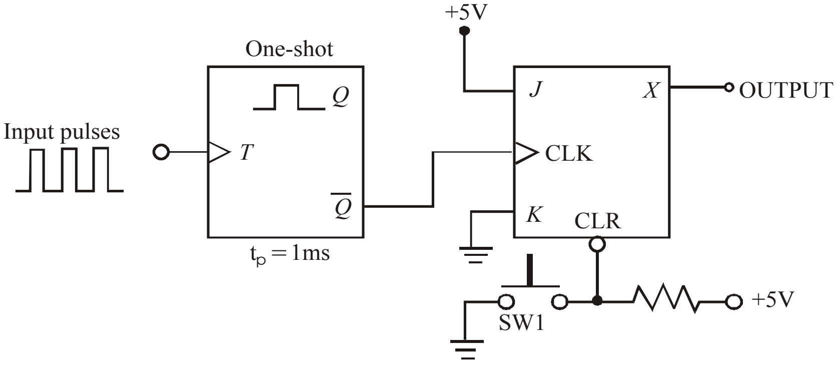


Fig. 5-100.

(*a*) *Describe how the circuit responds to input frequencies above* 1*kHz.* (*b*) *Describe how the circuit responds to input frequencies below* 1*kHz.* (*c*) *How would you modify the circuit to detect when the input frequency drops below* 50 kHz?

Solution. Given, a retriggerable one-shot used as a pulse-frequency detector.

(*a*) **Circuit response to input frequencies above 1kHz.** Closing the switch SW1, clears X to Zero. Since the one-shot has $t_P = 1$ ms, the one-shot will be retriggered before the end of the t_P interval for frequencies greater than 1 kHz. Thus $\overline{Q}$ will stay LOW. As a result of this, there is no clock input to JK flip-flop and hence X will remain LOW. **Ans.**

(*b*) **Circuit response to input frequencies below 1kHz** . If the input frequency falls below 1kHz, the $\overline{Q}$ will return HIGH before the one-shot is triggered again. This positive-going-trigger will clock JK flip-flop and X will change to HIGH. **Ans.**

(*c*) **Modification of the circuit to detect frequencies below 50 kHz.** In order to detect frequencies below 50 kHz, we will have to change t_P of the one-shot to *1/50 kHz = 20 ms*. **Ans.**

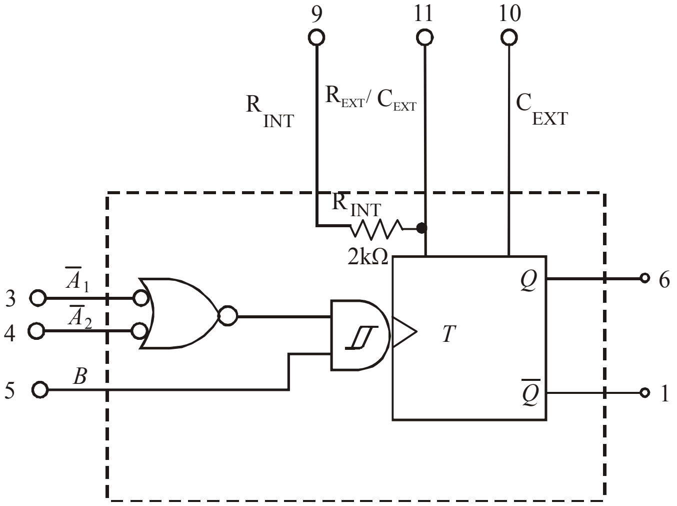
5-44. Actual One-Shot Devices

There are several one-shot ICs that are available commercially in both the non-retriggerable and retriggerable versions. Table 5-3 shows some of the most popular one-short ICs available commercially. As seen from the table 5-3 we find that 74121 and 74221 ICs are non-retriggerable one-shots. Where as 74122 and 74123 ICs retriggerable one-shots.

Table 5-3. Actual one-shot devices

Device	*Description*
74121	A *single non-retriggerable* one-shot IC in standard TTL series.
74221	A *dual non-retriggerable* one-shot IC in standard TTL series.
74LS221	A *dual non-retriggerable* one-shot IC in Low-power SchottkyTTL series
74HC221	A *dual non-retriggerable* one-shot IC in high-speed CMOS series
74122	A *single retriggerable* one-shot IC in standard TTL series
74LS122	A single retriggerable one-shot in IC low-power Schottky TTL series.
74123	A dual retriggerable one-shot IC in standard TTL series
74LS123	A dual retriggerable one-shot IC in lower-power Schottky TTL series
74HC123	A dual retriggerable one-shot IC in high-speed CMOS series.

Fig. 5.101 shows the logic symbol and 5-101 for the 74121 non-retriggerable one-shot IC. As seen from the diagram, the 74121 IC contains internal logic gates to allow inputs $\overline{A}_1$, $\overline{A}_2$, and B to trigger the device in a variety of ways. The B input is a Schmitt-Trigger type of input that is allowed to have slow transition times and still reliably trigger the one-shot. The pins labelled R_{EXT}, R_{EXT}/C_{EXT} are used to connect an external resistor and capacitor to achieve the desired output pulse duration.



Vcc = pin 14
GND = pin 7

Fig. 5-101. *Logic symbol of 74121*

A typical pulse width of 30 ns is produced when no external components are used and the internal timing registor (2 kΩ) is connected to V_{CC}. The pulse width can be set anywhere between 30 ns and 28 ns by the use of external components, (*i.e.* R_{EXT} and C_{EXT}). Mathematically the pulse width.

$$t_p = 0.7\,R\,C_{EXT} \qquad ...(i)$$

where R is either R_{INT} or R_{EXT}. Notice that if R is in kilohms (k Ω) and C_{EXT} is in picofarads (PF), the output pulse width (t_p) is in nanoseconds. Further if R is in kilohms (k Ω) and C_{EXT} is in microfarads (μF), the output pulse width (t_p) is in microseconds (μ S).

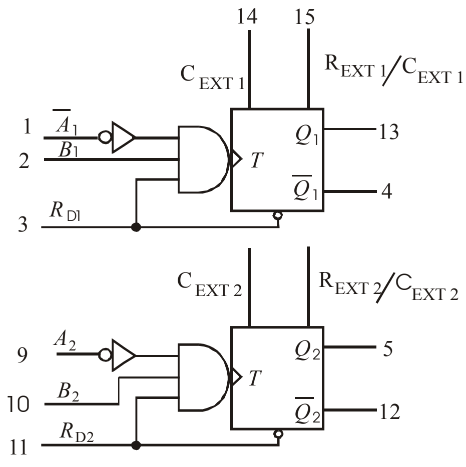
Table 5-4 shows the truth table of 74121. Notice that for the first four rows of the table, the Q-output is LOW.

Inputs			Output	
A_1	$\overline{A}_2$	B	Q	$\overline{Q}$
L	X	H	L	L
X	L	H	Ll	Ll
X	X	L	L	L
H	H	X	L	L
H	↓	H	⊓	⊔
↓	H	H	⊓	⊔
↓	↓	H	⊓	⊔
L	X	↑	⊓	⊔
X	L	↑	⊓	⊔

H = High voltage level
L = Low voltage level
X = Don't case
↑ = Rising edge
↓ = Falling edge

It is because of the fact that no trigger pulse is applie at any one of the three inputs. In the next five lines of the truth table, notice that we have a trigger pulse at one of the inputs $\left(A_1\ A_2 \text{ or } B\right)$ and the other two inputs are connected LOW or HIGH.

Fig. 5-102 (a) shows the logic symbol and 5-102 (b) the truth table for 74123 a dual retriggerable one–shot. As seen from the diagram, the 74123 IC contains two retrigger able one-shots. Each retriggerable one-shot contains $\overline{A}_1$, B_1 and R_D to trigger the device in a variety of ways.



(a)

INPUTS			***OUTPUTS***	
R_D	$\overline{A}$	B	Q	$\overline{Q}$
L	X	X	L	H
X	H	X	L	H
X	X	L	L	H
H	L	↑	⊓	⊔
H	↓	H	⊓	⊔
↑	L	H	⊓	⊔

H = HIGH voltage level
L = LOW voltage X= Don't care
X= Rising edge
↑ = falling edge
↓

⊓ = One HIGH-level pulse
⊔ = one LOW-level pulse

(b)

Fig. 5-102.

$$t_p = 0.28\, R\, C_{EXT} \left(1 + \frac{0.7}{R} \right) \quad ....(ii)$$

where 0.28 is a constant determined by a particular type of one-shot, R is either the internal or external resistor, C_{EXT} is in pF and t_p is in ns.

Notes.

1. While selecting the value of R_{EXT}, care should be taken that its value must be in kilohms (kΩ) so that the circuit current is in milli amperes (mA). If the resistor value is chosen in ohms (Ω), the circuit current will be in amperes which the IC will not be able to handle it. On the other hand, if the value of R_{EXT} is in megohms (MΩ), the current will be in micro amperes (μA) which may be too small for internal operation of 74121. Moreover, resistances with values in the range of megohms are susceptible to electrostatic noise.
2. For a desired pulse width t_w, we can determine the values of external components in two ways.
   (*i*) Arbitrarily select the value of capacitor, C_{EXT} and then using equation $t_W = 0.7\, R_{EXT} C_{EXT}$, determine the value of $R_{EXT.}$ be in microfarads (μF).
   (*ii*) Arbitrarily select the value of R_{EXT} and then using the equation, $t_W = 0.7\, R_{EXT}\, C_{EXT}$, determine the value of C_{EXT}.
   (*iii*) If t_W is required to be in nanoseconds, choose R_{EXT} in kilohms (kΩ) so that C_{EXT} is in picofarads. However if the desired t_W is required to be in milliseconds, still select R_{EXT} is in kilohms (kΩ), but C_{EXT} will now be in microfareds (μF). Care should be taken while selecting the component values in the sense that the components are availably only in certain standard values. For complete listing refer to Appendix E gives at the end of this book.
3. While selecting the value of C_{EXT}, care should be taken that it is much larger than any stray capacitance that might be encountered in a typical electronic circuit. Values of capacitance less than 100 pF (0.0001 μF) may be unsuitable because it is not uncommon for there to be 50 pF of stray capacitance between traces in a printed-circuit board.

Example 5-22. *A certain digital circuit requires a one-shot with a pulse width of 10* ms. *Determine the values of external components to be connected with an* IC *74121 to produce the desired pulse width. Also show the connections of the external components to 74121.*

Solution. Given the pulse width, $t_W = 10\ ms = 10 \times 10^{-3} s$

Let R_{EXT} = The value of external resistor and
C_{EXT} = The value of external capacitor.

Then we know that the pulse width for a 74121 IC (t_p),

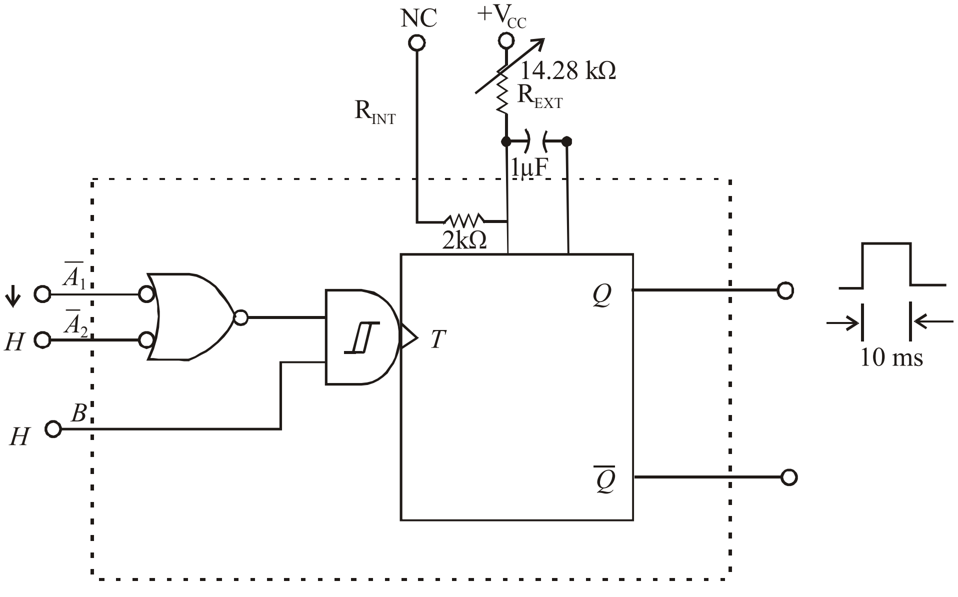
$$10 \times 10^{-3} = 0.7\, R_{EXT} C_{EXT} \quad ...\ (i)$$

Since the desired pulse width is in microseconds, let us arbitrarily select $C_{EXT} = 1\ \mu$F. Then from equations (*i*).

$$10 \times 10^{-3} = 0.7 \times R_{EXT} \times (1 \times 10^{-6})$$

$$\therefore \quad R_{EXT} = \frac{10 \times 10^{-3}}{0.7\,(1 \times 10^{-6})} = 14.28 \times 10^{-3}\ \Omega = 14.28\ k\Omega$$

In order to select a resistor value of 14.28 kΩ, we can use a 10 kΩ fixed resistor with a 5 kΩ potentiometer.



NC = No connection.

Fig. 5-103.

Fig. 5-103 shows a 14.28 kΩ resistor with 1 μF capacitor connected to the 74121 IC for producing a desired pulse width of 10 ms. Notice the settings on $\overline{A}_1$, $\overline{A}_2$ and B inputs of 74121 as well.

Example 5-23. *Design a circuit using a* 74121 *to convert a* 50 *kHz,* 80% *duty cycle square wave to a* 50 *kHz,* 50% *duty cycle square wave.*

Solution. Given a 50-kHz 80% duty cycle square wave.

Fig. 5-104 shows the given square wave. Notice that it has a time period of 1/50 kHz (= 20 μs) and has a pulse width of 80% of 20 μS (= 16 μs).

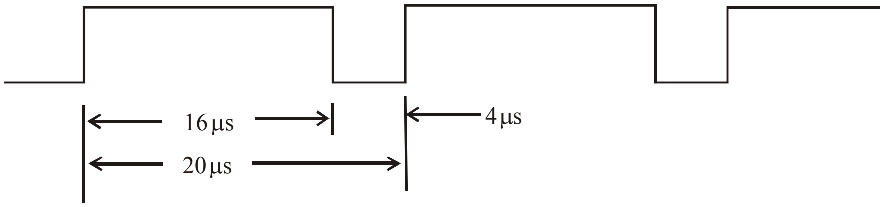


Fig. 5-104.

Now we have to stretch the 4 μs negative pulse to 10 μs to make the duty cycle 50 %. If we use the falling edge on the negative pulse to trigger $\overline{A}_1$ input to a 74121 and set the output pulse width (t_p) to 10 μs, we should have the solution as shown in 5-105 (b). The output will be taken from $\overline{Q}$ because it provides a negative pulse when triggered.

Let R_{EXT} = The value of external resister and

C_{EXT} = The value of external capacitor,

Then the pulse width (t_p)

$$10 \times 10^{-6} = 0.7 R_{EXT} C_{EXT}$$

Let us arbitrarily select $C_{EXT} = 0.001\ \mu F$ (equal to 1000pf); then

$$10\ \mu s = 0.7\ R_{EXT}$$

or

$$R_{EXT} = \frac{10 \times 10^{-6}}{0.7 \times 0.001 \times 10^{-6}}$$

$$= 14.4\ k\Omega$$

In order to select a resistor of 14.4 kΩ, we can choose a fixed resistor of 10 k Ω with a 5 kΩ potentiometer.

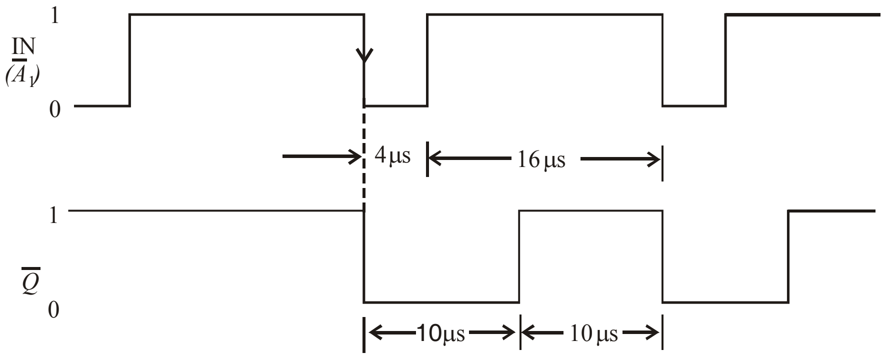


Fig. 5-105.

Fig. 5-106. shows the circuit connections for producing the desired waveform. Notice that the input waveform is applied at $\overline{A}_1$ input while $\overline{A}_2$ and B are tied HIGH. The output is obtained from $\overline{Q}$.

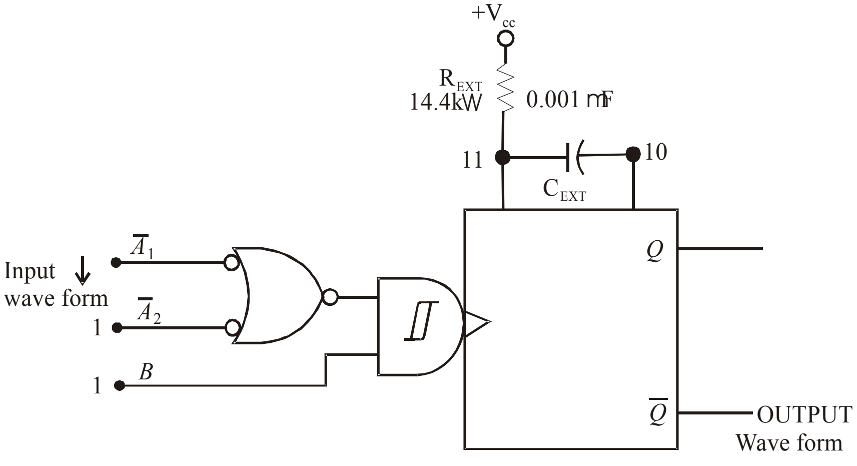


Fig. 5-106.

Example 5.24. *Determine the value of* R_{EXT} *and* C_{EXT} *that will produce a pulse width of* 50 µS *when connected to a* 74123.

Solution. Given a pulse width, $t_p = 50\ \mu s = 50 \times 10^{-6}\ s.$

Let R_{EXT} = The value of external resistor and

C_{EXT} = The value of external capacitor

Then we know that the pulse width for a 74123 IC (t_p),

$$50 \times 10^{-6} = 0.28\, R_{EXT}\, C_{EXT} \left(1 + \frac{0.7}{R_{EXT}}\right) \qquad ...(i)$$

Let us arbitrarily select C_{EXT} = 0.01 μF (recall that C_{EXT} has to be greater than 1000 pF), then, from equation (i),

$$50 \times 10^{-6} = 0.28\, R_{EXT}\, (0.01 \times 10^{-6}) \left(1 + \frac{0.7}{R_{EXT}}\right)$$

$$= 0.28\ (R_{EXT} + 0.7)\,(0.01 \times 10^{-6})$$

$$\therefore \qquad R_{EXT} + 0.7 = \frac{50 \times 10^{-6}}{0.28 \times (0.01 \times 10^{-6})} = 17857\ \Omega$$

or $$R_{EXT} = 17857 - 0.7 = 17856.3\ \Omega\ = 17.9\ k\ \Omega$$

Thus we can use a fixed resistor of 15 k and 5 k potentiometer to select R_{EXT} = 17.9 k Ω .

5-45. Clock Generator Circuits

We have already discussed in Art. 5-2 that flip-flop has two stable states due to which they are known as bistable multivibrators. Further in Art. 5-40 we have discussed that one-shot has one stable. state due to which they are referred to as monostable multivibrators. Now we shall discuss a third type of multivibrator which has no stable states. Such a multivibrator is called an astable or free-running multivibrator. An astable multivibrator switches back and forth (i.e. oscillates) between two unstable states without any external triggering.

There are several types of astable multivibrators that are in common use but the following three are important from the subject point of view;

1. Schmitt-Trigger oscillator
2. 555 Timer used as astable oscillator.
3. Crystal-controlled clock generator.

Now we shall discuss all the three oscillators one by one in the following pages.

5-46. Schmitt-Trigger Oscillator

Fig. 5-107 shows a Schmitt-trigger INVERTER connected as an oscillator. The signal at V_{OOT} is an approximate square wave with a frequency that depends upon the values of R and C values. The relationship between the frequency and RC values is shown in Fig. 5-108 for three different Schmitt-trigger INVERTERS.

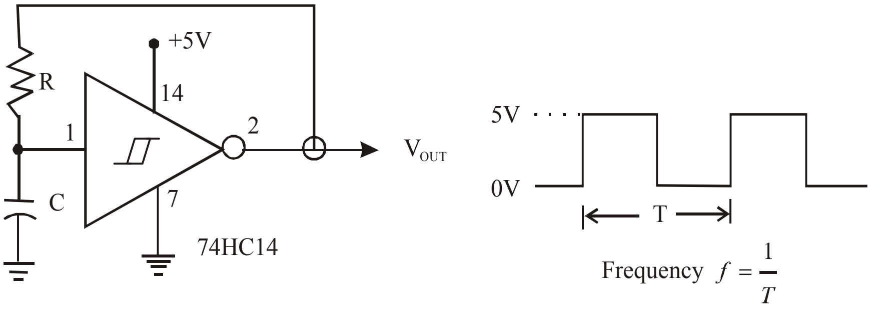


Fig. 5-107.

As seen from this diagram, for 7414 Schmitt-trigger INVERTER, the frequency is given by, $f = 0.8/RC$. Notice that for this IC, the value of R ≤ 500 Ω. For 74LS14 IC, the frequency of the output square wave is again 0.8/RC but the value of R ≤ 2 kΩ. On the other hand for 74HC14, the frequency of the output square wave is 1.2/RC, where the value of R must be 10 MΩ.

IC	*Frequency*	*Remarks*
7414	≈ 0.8/RC	(R ≤ 500 Ω)
74LS14	≈ 0.8/RC	(R ≤ 2 kΩ)
74HC14	≈ 1.2/RC	(R ≤ 10 MΩ)

Fig. 5-108.

5-47. 555 Timer Used as an Astable Multivibrator

The 555 IC is a very popular, general purpose timer IC. It can be connected as a one-shot or as an astable multivibrator or as a free running oscillator. Fig. 5-109 shows how external components can be connected to a 555 so that it operates as a free-running oscillator. The output of the oscillator is a repetitive rectangular waveform as shown in Fig. 5-109 (b).

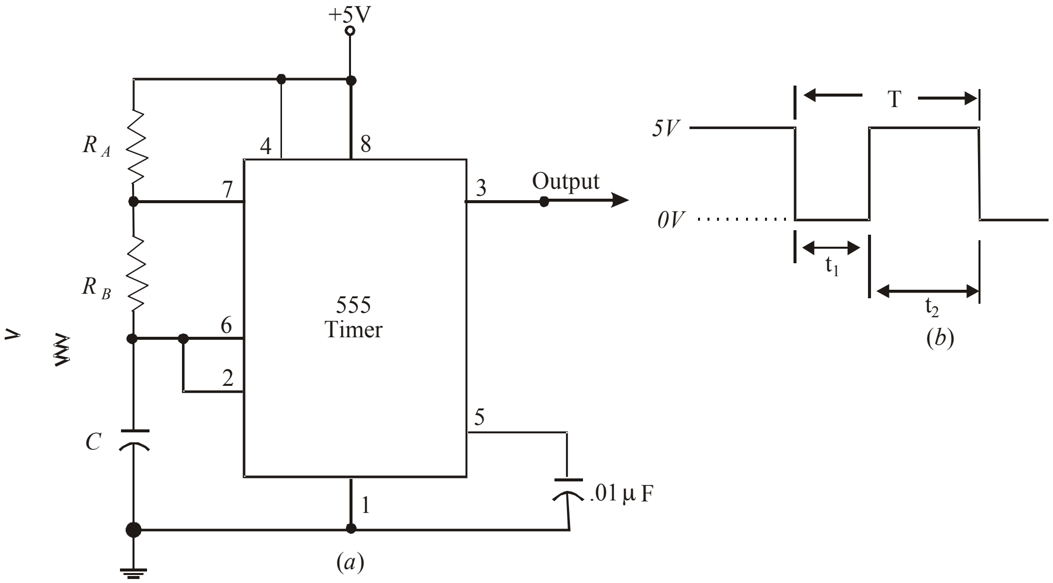


Fig. 5-109. *555 timer used as an astable multivibrator*

As seen from the diagram, the output switches between two logic levels with the time intervals ($t_1$and t_2) at each level. These two time intervals are determined by the R and C values.

The time interval t_1 is given by the equation,

$$t_1 = 0.7\, R_B C \qquad ...(i)$$

and the time interval,

$$t_2 = 0.7\, (R_A + R_B)\, C \qquad (ii)$$

∴ The time period, of the output waveform

$$T = t_1 + t_2$$

Substituting the values of t_1 and t_2 from equations (*i*) and (*ii*), the time period,

$$\text{or} \qquad T = 0.7\, R_B C + 0.7\,(R_A + R_B)\, C$$

$$= 0.7\,(R_A + 2\,R_B)\, C$$

The frequency of the output waveform,

$$f = \frac{1}{T} = \frac{1}{0.7\,(R_A + 2\,R_B)\,C}$$

$$= \frac{1.44}{(R_A + 2\,R_B)\,C}$$

Further, the duty cycle of the output waveform,

$$\text{duty cycle} = \frac{t_2}{T} \times 100\ \%$$

It may be noted from equations (i) & (ii) that t_1 and t_2 cannot be equal unless R_A is made zero. This cannot be done without producing excess current through the device. This means, it is impossible to produce an output waveform with a perfect 50% duty cycle. However, it is possible to get very close to 50% duty cycle (i.e. $t_1 \approx t_2$) by selecting,

$$R_B \gg R_A$$

It may be carefully noted that the value of R_A must be greater than 1k Ω and the value of C must be greater than 500 pF.

Example 5-25. *Fig.* 5-110 *shows the circuit of a* 555 *timer used as an astable multivibrator.*

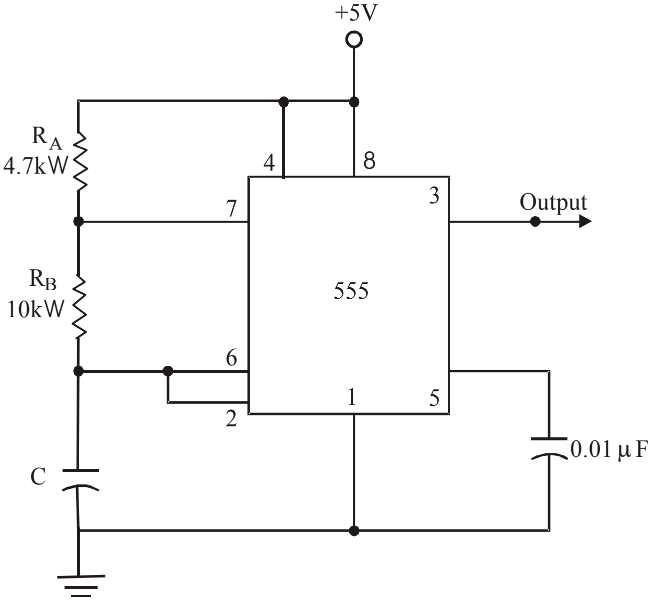


Fig. 5-110.

Determine the values t_1, t_2, frequency and duty cycle of the output waveform.

Solution. Given : $R_A = 4.7\,k\,\Omega = 47 \times 10^3\,\Omega$, $R_B = 10\,k\,\Omega = 10 \times 10^3\,\Omega$ and $C = 680\,pF = 680 \times 10^{-12}\,F.$

We know that the time interval,

$$t_1 = 0.7\ \mathrm{R_B\ C} \times 0.7 \times (10 \times 10^3) \times (680 \times 10^{-12})$$

$$= 4.76 \times 10^{-6}\ s$$

$$= 4.76\ \mu s \quad \textbf{Ans.}$$

and the time interval,

$$t_2 = 0.7\ (R_A + R_B)\ C$$

$$= 0.7 \times (\,(4.7 \times 10^3) + (10 \times 10^3)) \times (680 \times 10^{-12}\,)$$

$$= 6.997 \times 10^{-6}\,s$$

$$\approx 6.997\ \mu s \quad \textbf{Ans.}$$

The time period,

$$T = t_1 + t_2 = 4.76 + 6.997 = 11.757\ \mu s \quad \textbf{\textit{Ans.}}$$

and the frequency,

$$f = \frac{1}{T} = \frac{1}{11.757 \times 10^{-6}} = 8.5 \times 10^4$$

$$= 85\ kHz \qquad \textbf{Ans.}$$

The duty cycle,

$$= \frac{t_2}{T} \times 100\% = \frac{6.997}{11.757} = 100\%$$

$$= 59.5\ \% \quad \textbf{Ans.}$$

SUMMARY

In this chapter you have learned that :

1. A flip-flop is a logic circuit which has a memory chracteristic. Its Q and $\overline{\mathrm{Q}}$ outputs will go to a new state in response to an input pulse and will remain in that new state after the input pulse is removed.
2. A NAND and a NOR latch are flip-flops that respond to logic levels on their SET and CLEAR inputs.
3. Clearing (or resetting) a flip-flop means that its output is Q = 0 $\left(\overline{\mathrm{Q}}=1\right)$ state. Setting a flip-flop means that its output is Q = 1 $\left(\overline{\mathrm{Q}}=0\right)$ state.
4. Clocked flip-flops have a clock input (CLK) that is edge triggered. The edge triggered flip-flop means that it triggers on the rising edge or on the falling edge of the CLK input.
5. Clocked (or Edge triggered) flip-flops can be triggered to a new state by the active edge of the clock input according to the state of the synchronous control inputs (S, C or J, K or D).
6. Most clocked flip-flops also have asynchronous inputs that can set or clear the output independent of the clock input.
7. A one-shot (or monoshot) is a logic circuit that can be triggered from its normal resting state (Q = 0) to its quasi-stable state (Q = 1). It remains triggered in quasi-stable state for a time interval proportional to an RC time constant.

8. As table multivibrator is a logic circuit that has no stable states and is used as an oscillator to generate timing waveforms in digital system.
9. Logic circuits that have a schmitt-trigger type of input will respond reliably to slow changing signals and will produce outputs with clean sharp edges.

GLOSSARY

Astable multivibrator. A digital circuit that oscillates between two unstable states.

Asynchronous counter. A type of binary counter in which flip-flop output serves as the clock input signal for the next flip-flop in the chain.

Asynchronous inputs. The inputs of a flip-flop that can affect its operation independent of the synchrounous inputs (J, K or D) and clock inputs.

Clocked flip-flop. The flip-flops that have clock input.

Clocked D flip-flop. A type of clocked flip-flop in which D (or data) input is the synchronous inputs.

Clocked J-K flip-flop. A type of clocked flip-flop in which inputs J and K are the synchrounous inputs.

Edge detector circuit. A circuit that produces a narrow pulse that occurs coincident with the active edge of clock input pulse.

Edge triggered. A method in which a clocked flip-flop is activated.

Freequency division. The devision of input clock frequency by an integer number. Frequency division is achieved by using flip-flop circuits.

MOD number. An integar number of different states that a counter (consists of flip-flops) can squeence though. It's also called counter's frequency division ration.

Non-retriggerable one shot. A type of one-shot that will not respond to a trigger input signal while in its quasi-stable state.

Pulse steering circuit. A Logic circuit that can be used to select the destination of an input pulse depending upon the logic levels present at is circuit inputs.

Retriggereable one shot. A type of one-shot that will respond to a trigger input signal while in its quassi-stable state.

Schmitt-trigger. A digital circuit that accepts a slow changing input signal and produces a rapid, oscillation-free transition at the output.

Timer. A digital circuit that can produce an output signal whose frequency depends on external components R and C.

Synchronous control inputs. The input signals that in the presence of clock pulse determine the output state of a flip-flop.

DESCRIPTIVE QUESTIONS

1. List the two types of latches. Explain briefly, the operation of each latch using the block symbol and truth table.
2. Define the followings : (*a*) set-up time and (*b*) Hold-time.
3. What is meant by the following two terms :
   (*a*) edge triggered (*b*) level triggered
4. How does a J-K flip-flop differ from an S-C flip-flop in its basic operation?
5. Explain why a J-K flip-flop can be used as an S-C flip-flop but an S-C flip-flop cannot be used as a J-K flip-flop.
6. Realize a J-K flip-flop using S-C (i.e. S-R) flip-flops. Explain the operation of J-K flip-flop using its block diagram and truth table.
7. Explain briefly, how a J-K flip-flop can be used for parallel data transfer.
8. Describe how a D latch operates differently from an edge triggered D flip-flop.

9. Explain how, the operation of asynchronous input in a J-K flip-flop differ from that of a synchronous input.

10. Draw the circuit of a J-K flip-flop. Describe its working.

(*Gauhati University, 2007, RGTU., Feb., 2010*)

11. What is master slave flip-flop? Give logic diagram of JK master slave flip-flop using NAND gates. Explain the working.

(*Nagpur University, 2008*)

12. Convert the following

(*i*) D Flip flop to J-K flip-flop

(*ii*) SR flip to T flip-flop

(*Nagpur University, 2008*)

13. Explain different methods of Triggering of flip-flop. (*Nagpur University, 2008*)

14. Draw the diagram for J-K Master slave flip flop using NAND gates and explain its operation.

(*Nagpur University, 2008, GTU., Dec., 2011*)

15. Design a J-K flip flop using S-R flip-flop.

(*Nagpur University, 2008*)

16. What are the advantages of Master Slave flip-flop.

(*Mahatma Gandhi University, Jan. 2007*)

17. Explain the process of converting a J-K flip flop into S-R flip-flop.

(*Mahatma Gandhi University, Jan., 2007*)

18. Draw the circuit diagrams of clocked J-K, D and T flip-flops using fundamental logic gates and explain their working using the respective truth tables.

(*Mahatma Gandhi University, Dec. 2007*)

19. How does a Master Slave Flip Flop eliminate the race around condition ?

(*Mahatma Gandhi University, Nov. 2005*)

20. Draw a Master-Slave J-K flip flop using NAND gates only with the present, clear input facility.

(*Mahatma Gandhi University, Nov. 2005*)

21. Show how using a D-type, flip flop how a J-K flip-flop can be designed.

(*Mahatma Gandhi University, Nov. 2005*)

22. Write the truth table of D-type flip-flop and explain its applications.

(*Mahatma Gandhi University, May 2005*)

23. What is race round condition? Explain how it eliminated using JK master slave flip-flop.

(*VTU, Jul./Aug. 2005, RGTU., June 2009, 2011, GBTU., 2010-11*)

24. Explain the different types of flip flops along with their truth table. Also explain the race-round condition in a flip flop.

(*VTU, Jan./Feb. 2006*)

25. With a neat logic diagram and timing waveform describe the operation of a master-slave J-K flip-flop.

(*VTU, Jul. 2006*)

26. Draw the circuit of a J-K flip flop using NAND gates building blocks. Verify that J-K flip-flop satisfy the difference equation: $Q_{m+1} = J_n \overline{Q}_n + \overline{K}_n Q_n$.

(*VTU, Dec./Jan. 2008*)

27. Show how J-K flip flop can be connected as

(*i*) D-flip-flop

(*ii*) T-Flip-flop

28. Discuss what is race-around in J-K flip flop and describe

(*i*) Master-slave J-K flip-flop

(*ii*) Edge trigged flip-flop

(*VTU, Dec./Jan. 2008*)

29. Give the truth table after each flip-flop type

(*i*) J-K

(*ii*) D

(*iii*) T

(*PTU, May 2008*)

30. Draw the logic symbols for T and R-S flip-flops. Explain the function of each type of flip-flop. (*PTU, May 2008*)

31. Draw a logic symbol for a D-flip flop and compare with R-S flip-flop.

(*PTU, Dec. 2008*)

32. Draw a master-slave J-K flip flop system. Explain the operation stage that the race-around condition is eliminated.

(*PTU, Dec. 2008*)

33. What is the difference between level and edge triggering? Explain the working of master slave J-K flip-flop.

(*PTU, May, 2009*)

34. What is difference between latch and flip-flop?

(*PTU, Dec. 2009*)

35. Perform the following conversions

(*i*) S-R flip flop to J-K flip-flop.

(*ii*) T flip to D flip-flop

(*PTU, Dec. 2009*)

36. What is the drawback of S-R flip-flop? How is this minimize?

(*Anna University, May/Jan. 2006*)

37. How does J-K flip flop differ from an S-R flip-flop in its basic operation?

(*Anna University, Nov./Dec. 2006*)

38. Draw the logic diagram for a master slave J-K flip-flop and explain.

(*Anna University, Nov./Dec. 2006*)

39. How does a J-K flip flop differ from the S-R flip-flop in its basic operation?

(*Anna University, May/Jun. 2006*)

40. Draw the clocked S-R flip-flop and explain with truth table.

(*Anna University, May/Jun, 2006*)

41. Give the characteristics expression of a J-K flip-flop.

(*Anna University, Nov./Dec. 2007*)

42. Draw the logic symbol and truth table of a D flip-flop.

(*Anna University, Apr./May 2008, RGTU., Dec., 2010*)

43. Explain the working principle of a master slave J-K FF.

(*Gauhati University, 2006*)

44. Why S = R = 1 is not permitted in the S-R FF.

(*Gauhati University, 2006*)

45. What is meant by 'excitation table' ? Write down the excitation table SR, JK, T and D FF.

(*Gauhati University, 2006*)

46. Explain the difference among a truth table, a state table, a characteristics table of flip-flops.

(*GBTU/MTU, 2005-06*)

47. Is the master slave option limited to J-K flip-flops only explain?

(*GBTU/MTU, 2004-05*)

48. Convert the D flip-flop to T flip-flop.

(*GBTU/MTU, 2004-05*)

49. Draw the circuit for converting S-R flip-flop into D flip-flop. Also design a one input and one output sequential circuit using D flip-flop for the state diagram shown in Fig......

(*GBTU/MTU, 2009-10*)

50. Design a 4- bit binary synchonous counter with D flip-flop. Also briefly describe the universal shift register.

(*GBTU/MTU, 2009-10*)

51. Explain the mechanism that a flip-flop holds a one bit of information. Describe the operation and working of the following flip-flops.

(*a*) S-R (*b*) J-K
(*c*) Mater Slave (*d*) T
(*e*) D

(*GBTU/MTU, 2006-07*)

52. Design a R-S flip-flop using T flip-flop.

(*GBTU/MTU, 2006-07*)

53. Draw gate diagram of J-K flip-flop write truth table.

(*Nagpur University, 2004*)

54. Convert D to T type of flip-flop.

(*Nagpur University, 2004*)

55. Convert

(*i*) S-R to J-K flip-flop

(*ii*) T to S-R flip-flop

(*Nagpur University, 2004*)

56. What is the function of preset and clear in a flip-flop?

(*Nagpur University, 2004*)

57. Explain the operation of D flip-flop with suitable logic diagram and truth table.

(*Nagpur University, 2004*)

58. Given and explain excitation tables of D and J-K flip-flops.

(*Nagpur University, 2004*)

59. Explain the race round condition in case of J-K flip-flop. How it can be eliminated?

(*Nagpur University, 2004, GTU, May 2011*)

60. Convert a J-K flip-flop into a T type flip-flop.

(*Nagpur University, 2004*)

61. Difference between latch and Flip-flop.

(*Nagpur University, 2004*)

62. Explain difference method of triggering of a flip-flop.

(*Nagpur University, 2004*)

63. What is flip-flop? Explain the operation of S-R and J-K flip-flop along with their truth tale. Give their realization using NAND gates.

(*Gauhati University, 2003*)

64. Define the following terms as applied to Flip-Flop
(*i*) Hold time
(*ii*) Propagation delay
(*iii*) Maximum clock frequency

(*Gauhati University, 2003*)

65. Explain the working of Master-Slave J-K flip-flop.

(*Gujarat Technological University, Dec. 2009*)

66. Convert S-R flip-flop to T and J-K flip-flop.

(*Gujarat Technological University, Dec. 2009*)

67. Draw the S-C flip-flop. Give its working. Give the truth table. What is race around condition ? For which combination of the inputs this is observed?

(*RGPV Bhopal, June 2008*)

68. Describe the following in brief, Excitation table and next state equation of FFs

(*GBTU/MTU, 2006-07*)

69. Define propagation delay.

(*Nagpur University, 2004*)

70. Give and explain excitation tables of D and J-K flip-flops.

(*Nagpur University, 2004, WBUT., 2011*)

71. Name the fastest logic family and draw its basic gate structure. Describe the circuit operation briefly.

(*Gauhati University, 2003*)

72. Describe how a retriggerable one-shot operates differently from a non-retriggerable one-shot.

73. How is the output pulse width set in most IC one-shots?

74. Explain the difference between an astable multivibrator and a monostable multivibrator.

75. Sketch the functional block diagram of a 555 IC timer. Briefly explain the function of each component.

76. What is multivibrator ? Explain monostable and bistable multivibrator.

(*RGTU., Dec., 2010*)

77. Define flip-flop. Give their types and explain. The flip-flop in details.

(*RGTU., Dec., 2010*)

78. Explain the working of Monostable multivibrator using 555 times.

(*RGTU., June, 2009*)

79. Explain the operation of monostable multivibrator with the help waveforms.

(*RGTU., June., 2011*)

80. Explain the working of R-S flip-flop.

(*RGTU., March-April, 2010*)

81. With the help of circuit diagram and timing diagram explain the working of monostable multivibrator.

(*RGTU., Dec., 2009*)

82. With the help of circuit diagram and timing diagram explain the working of a stable Multivibrator.

(*RGTU., June., 2010*)

83. Explain the working of 555 times as bistable multivibrator with the help of circuit diagram and waveforms.

(*RGTU., Dec., 2010*)

84. Describe the R-S flip-flop using NAND gates with circuit diagram truth table and waveforms.

(*RGTU., June, 2011*)

85. Convert a 7 flip-flop into a J-K flip-flop.

(*WVUT., 2011*)

TUTORIAL PROBLEMS

1. The waveforms of Fig 5-111 (a) are applied to the inputs of a NAND gate latch shown in Fig.5-111 (b). Assume that initially $Q = 0$, and determine the waveform at Q–output.

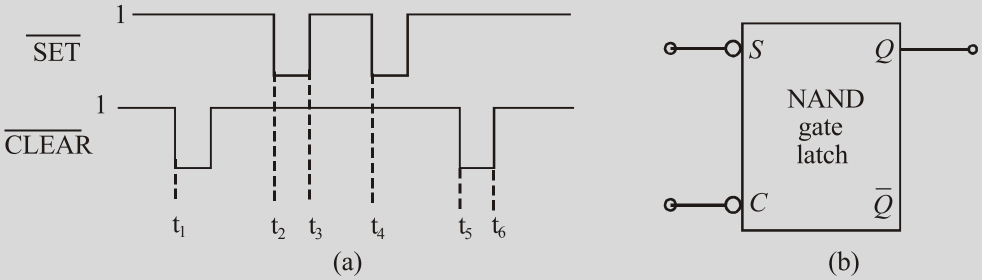


Fig. 5-111.

2. The waveforms of Fig. 5-112 (a) are applied to the inputs of a NOR gate latch shown in Fig. 5-112 (b). Assume that initially $Q = 0$ and determine the waveform at Q output.

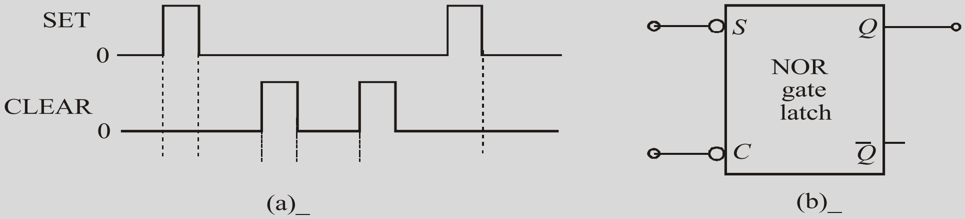


Fig. 5-112.

3. Apply the J-K and CLK waveforms to a flip-flop shown in Fig. 5-113. Assume that $Q = 0$ initially. Sketch the output waveform.

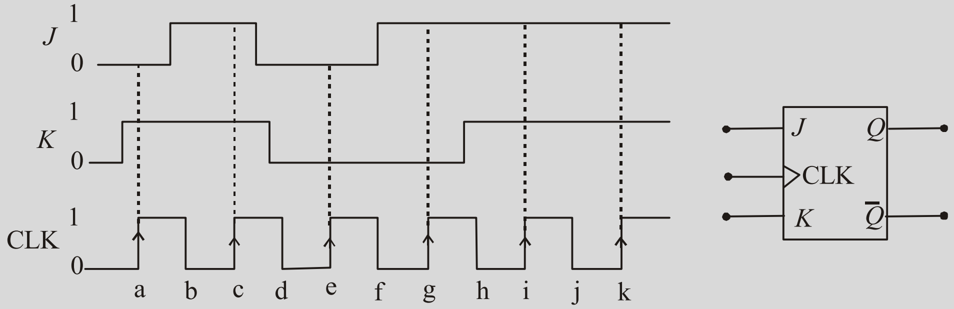


Fig. 5-113.

4. Determine the output waveform for a positive-edge triggered D flip-flop for the J, K and CLK waveforms as shown in Fig. 5-114.

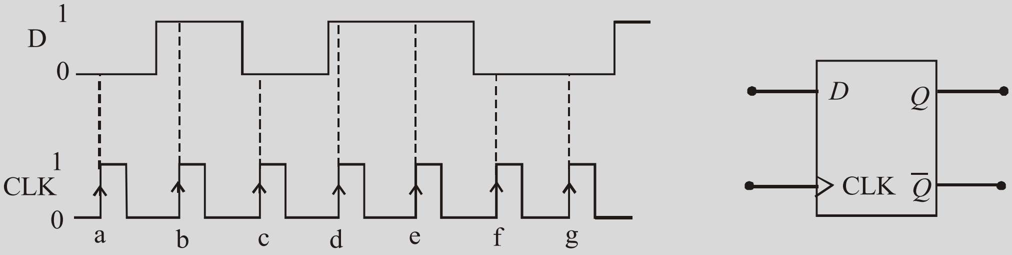


Fig. 5-114.

5. Determine the Q waveform for a D latch with ENABLE (EN) and DATA (D) input waveforms shown in Fig. 5-115. Assume Q = 0 initially.

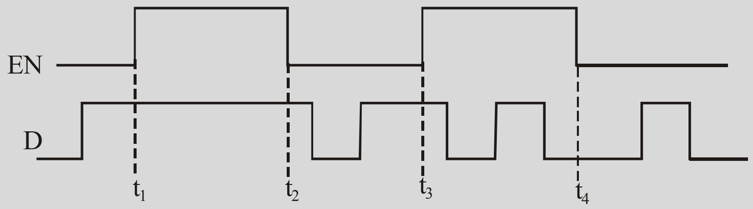
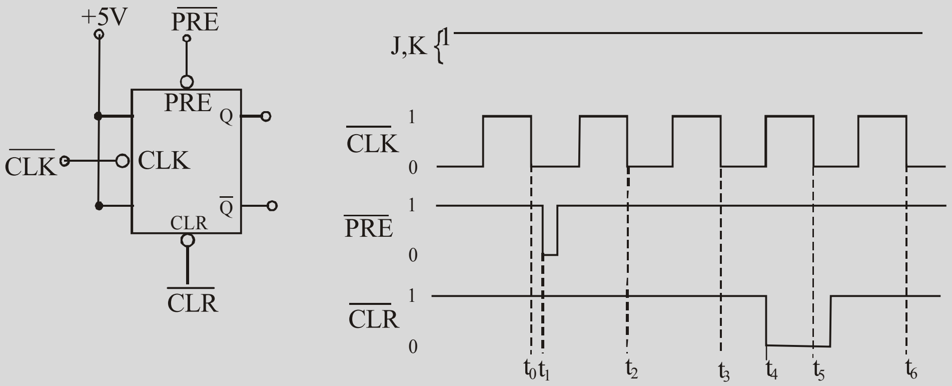


Fig. 5-115.

6. Fig. 5-116 (a) shows the symbol for a J-K flip-flop that responds to a NGT (i.e. negative going trigger) on its clock input and has asynchronous inputs. Sketch the output waveform in response to the $\overline{CLK}$, $\overline{PRE}$ and $\overline{CLR}$ waveforms as shown in Fig. 5-116 (b)



(a) (b)

Fig. 5-116.

7. Determine the Q output for a negative edge triggered J-K flip-flop for the input waveforms shown in Fig. 5-117. Assume $t_H = 0$ and that Q = 0 initially.

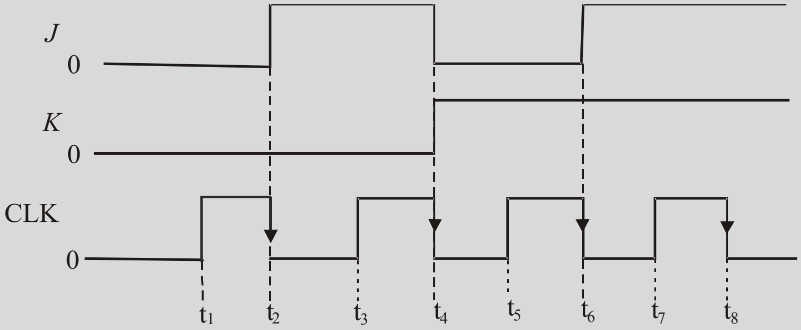


Fig. 5-117.

8. A 20-kHz clock signal is applied to a J-K flip-flop with J = K = 1. Sketch the output waveform and determine its frequency.

(**Ans.** 10 kHz)

9. In a J-K flip-flop, J = K = 1. A 1 MHz square wave is applied at its clock input. It has a propagation delay of 50 ns. Draw the input square wave and the output waveform expected at Q. Show the propagation delay time.

10. A D flip-flop has following datasheet information :

Set up-time = 5 ns
Hold time = 10 ns
Propagation time = 15 ns

How far ahead of the triggering clock edge must the data be applied.

11. Draw the block diagram of a 4-bit shift register using D flip-flops. If initially all the flip-flop outputs are in zero state, prepare the truth table when the input sequence is 1, 1, 0, 1, 0. Draw the above shift register using J-K flip-flops only.

12. Consider a 4-bit shift register using J-K flip-flops. Assume that initially, Q_0 =1, Q_0 = 1, Q_2 = 1 and Q_3 = 1. Sketch the output of each flip-flop if an input sequence 101101 is applied to D_0 synchronously with the clock (Q_0 is LSB).

13. Calculate the frequency and the duty cycle of the 555 astable multivibrator output for C = 0.001 μF, R_A = 2.2 kΩ and R_B = 100 kΩ .

(**Ans** 50.5%)

MULTIPLE CHOICE QUESTIONS

1. If a NAND latch has a 1 on the SET input and a 0 on the CLEAR input, then the SET input goes to 0, the latch will be:

(*a*) HIGH (*b*) LOW (*c*) Invalid (*d*) None of these

2. The invalid state of a NAND latch occurs when

(*a*) *S = 1, C = 0* (*b*) *S = 0, C = 1* (*c*) *S = 1, C = 1* (*d*) *S = 0, C = 0*

3. The invalid state of a NOR latch occurs when

(*a*) *S = 1, C = 0* (*b*) *S = 0, C = 1* (*c*) *S = 1, C = 1* (*d*) *S = 0, C = 0*

4. Like a latch, the flip-flop belongs to a category of logic circuits known as

(*a*) monostable multivibrators (*b*) astable multivibrators
(*c*) bistable multivibrators (*d*) none of these

5. Which of the following flip-flops is used as a latch?

(*a*) J-K flip-flop (*b*) S-C flip-flop
(*c*) D flip-flop (*d*) T flip-flop

6. A flip-flop which can have an uncertain output state is :

(*a*) J-K flip-flop (*b*) S-C flip-flop
(*c*) D flip-flop (*d*) T flip-flop

7. The purpose of a clock input to a flip-flop is to

(*a*) clear the device
(*b*) set the device
(*c*) always cause the output to change the states
(*d*) cause the output to assume a state dependent on the controlling (S - C, J-K or D) inputs.

8. A feature that distinguishes the J-K flip-flop from an S-C flip-flop is the

(*a*) toggle condition (*b*) preset input
(*c*) type of clock (*d*) clear input.

9. A J-K flip-flop is in the toggle condition when

(*a*) $J = 1, K = 0$ (*b*) $J = 1, K = 1$

(*c*) $J = 0, K = 0$ (*d*) $J = 0, K = 1$

10. A J-K flip-flop with J = 1 and K = 1 has a 10 kHz clock input. The Q – output is,

(*a*) constantly LOW (*b*) constantly HIGH

(*c*) a 5 kHz square wave (*d*) a 10 kHz square wave

11. For an edge-triggered D flip-flop,

(*a*) a change in the state of the flip-flop can occur only at a clock pulse edge

(*b*) the state that flip-flop goes to depends on the D input

(*c*) the output follows the input at each clock pulse

(*d*) all of these answers

12. The flip-flop shown in Fig. 118 logically behaves as

(*a*) a D flip-flop (*b*) a J-K flip-flop (*c*) a T flip-flop (*d*) an R-S flip-flop

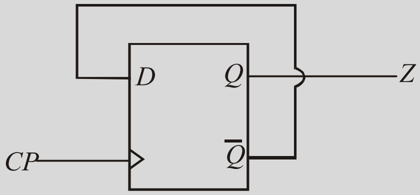


Fig. 5-118.

(*Grad. I.E.T.E. June, 1997*)

13. A J-K flip-flop is a device to

(*a*) divide the frequency by 2

(*b*) divide the frequency by 4

(*c*) generate waveform of same frequency as that of the input

(*d*) cannot be used for frequency division.

14. A 1 μs pulse can be converted into a 1 ms pulse by using

(*a*) a monostable multivibrator (*b*) an astable multivibrator

(*c*) a bistable multivibrator (*d*) a J-K flip-flop

(*U.P.S.C. Engg. Services, 1999*)

15. The following is not a sequential circuit

(*a*) J-K flip-flop (*b*) counter (*c*) full-adder (*d*) shift register

(*Dip. I.E.T.E., Dec. 1996*)

16. A one – shot is a type of

(*a*) monostable multivibrator (*b*) astable multivibrator

(*c*) bistable multivibrator (*d*) timer

17. An astable multivibrator

(*a*) requires a periodic trigger input (*b*) has one stable states

(*c*) is an oscillator (*d*) produces a non-periodic pulse output.

18. A 1 msec pulse can be converted to a 10 msec pulse by using:

(*a*) an astable multivibrator (*b*) a monostable multivibrator

(*c*) a bistable multivibrator (*d*) a J-K flip-flop

(*U.P.S.C. Engg. Services 1990*)

19. A retriggerable one shot is one which

(*a*) can be triggered only once

(*b*) has two quassi-stable states

(*c*) cannot be triggered until full pulse has been outputted.

(*d*) is capable of being triggered while the output is being generated.

20. The output of the circuit shown in Fig. 5-119 will be

(*a*) delayed pulses (*b*) squarewaves

(*c*) trianglular waves (*d*) trapezoidal waves.

(*U.P.S.C Engg. Services 1997*)

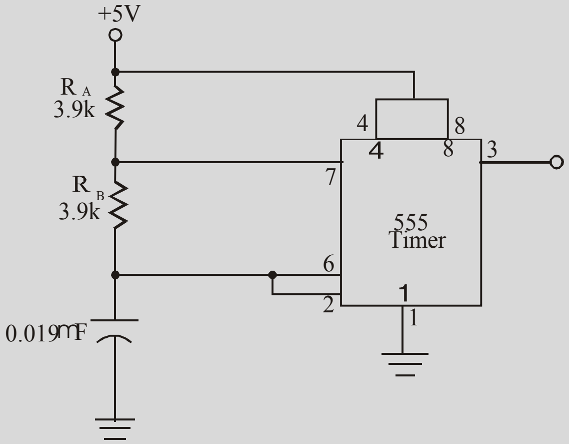


Fig. 5-119.

21. The input waveform V_i and the output waveform V_o of a Schmitt NAND are shown in Fig 5-120. The duty cycle of the output waveform is

(*a*) 100 % (*b*) 85.5 %

(*c*) 72.2 % (*d*) 25 %

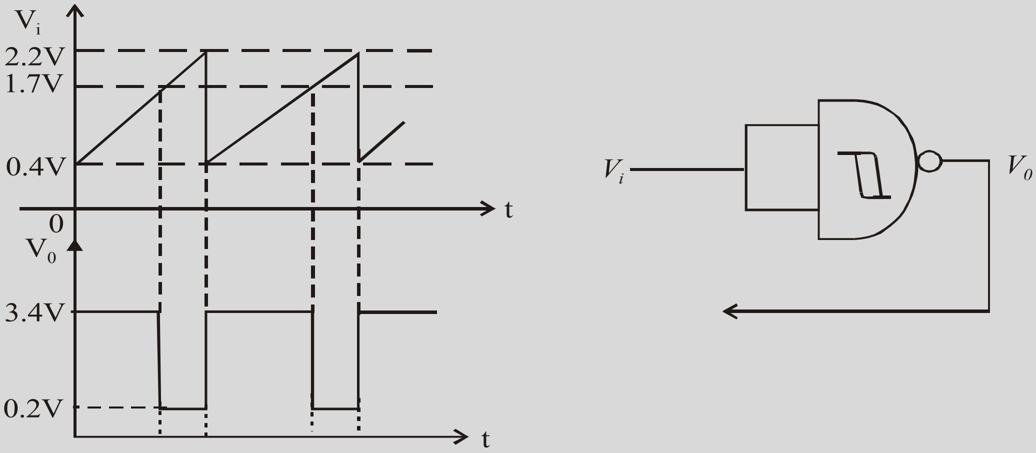


Fig. 5-120.

(*U.P.S.C Engg. Services. 1999*)

22. Which of the following circuit is used for production of delays?

(*a*) astable multivibrator (*b*) bistable multivibrator

(*c*) monostable multivibrator (*d*) schmitt-trigger

23. A Schmitt-trigger can be used as a

(*a*) comparator only

(*b*) square-wave generator only

(*c*) flip-flop only

(*d*) comparator, square wave generator or flip-flop.

24. A 555 IC timer can be used to operate as

(*a*) as monostable multivibrator (*b*) an astable multivibrator

(*c*) a voltage controlled oscillator (*d*) all of the above

ANSWERS

1. (*a*)	**2.** (*d*)	**3.** (*c*)	**4.** (*c*)	**5.** (*b*)	**6.** (*b*)
7. (*d*)	**8.** (*a*)	**9.** (*b*)	**10.** (*c*)	**11.** (*d*)	**12.** (*c*)
13. (*a*)	**14.** (*a*)	**15.** (*c*)	**16.** (*a*)	**17.** (*c*)	**18.** (*b*)
19. (*d*)	**20.** (*b*)	**21.** (*c*)	**22.** (*c*)	**23.** (*a*)	**24.** (*d*)

6

DIGITAL ARITHMETIC : OPERATIONS AND CIRCUITS

OUTLINE

Objectives

Upon completion of this chapter, you should be able to

- do binary addition, subtraction, multiplication and division on the given two binary numbers.
- add and subtract the given two hexadecimal numbers
- manipulate the signed binary numbers using the 2's complement system.
- understand the usage of full-adders in the design of parallel binary adders.
- explain the different types of code converter.
- understand the magnitude comparator.

6-1. Introduction

Many digital systems such as computers, video game machines, PDAs, recent mobile phones have a capability to do a variety of arithmetic operations.

In order to understand the functioning of such a digital system, it is important to know the various methods of performing arithmetic operations and the circuits that can do that task. We will begin our study by learning first, how the various arithmetic operations are performed on binary numbers using "pencil and paper". Then we will study the actual integrated circuits that perform these operations in digital systems.

6-2. Binary Addition

The addition of two binary numbers is performed in exactly the same manner as the addition of decimal numbers. Let us review the decimal addition. Consider the addition of two decimal numbers 264_{10} and 173_{10}.

$$\begin{array}{llccc} & & \text{MSD} & & \text{LSD} \\ \text{Carry} & \rightarrow & 1 & & \\ & & 2 & 6 & 4 \\ & + & 1 & 7 & 3 \\ \hline & & 4 & 3 & 7 \end{array}$$

Recall that for decimal addition we start from the least-significant digit side. Thus the sum of digits 4 and 3 produce 7. Next the digits in the second position are added. This produces13. So we write 3 in the second position and a carry of 1 in the third position. Finally the digits in the third position are added along with the carry. This produces a digit 4 in the third position.

We can follow the same general steps for binary addition. Unlike decimal addition, there are only four cases that can occur in binary addition. These four cases are:

$$\begin{aligned} 0+0 &= 0 \\ 1+0 &= 1 \\ 1+1 &= 10 = 0 + \text{carry of 1 into next position.} \\ 1+1+1 &= 11 = 1 + \text{carry of 1 into next position.} \end{aligned}$$

The last case occurs when the two bits in a certain position are 1 and there is a carry from the previous position.

Consider for example the addition of the two binary numbers at a time (decimal equivalents are shown in parenthesis)

$$\begin{array}{lrl} & (a) & \\ \text{Carry} \rightarrow 1 & & \\ & 1\,1 & (3) \\ + & 1\,1 & (3) \\ \hline & 1\,1\,0 & (6) \end{array} \qquad \begin{array}{lrl} & (b) & \\ & & \\ & 1\,0\,0 & (4) \\ + & 1\,0 & (2) \\ \hline & 1\,1\,0 & (+6) \end{array} \qquad \begin{array}{rl} (c) & \\ & \\ 1\,1\,0 & (6) \\ 1\,0\,0 & (4) \\ \hline 1\,0\,1\,0 & (10) \end{array}$$

As a matter of fact, it is not necessary to consider the addition of more than two binary numbers at a time. It is because of the fact that in all digital systems the circuitry that actually performs the addition can handle only two numbers at a time. Whenever we need to add more than two numbers, first the two numbers are added together and then their sum is added to the third number and so on. It is not a serious drawback since digital computers can typically perform an addition operation in few microseconds or less.

It will be interesting to know that addition is the most important arithmetic operation in digital systems. The operations of subtraction, multiplication and division are carried out using addition as their basic operation.

6-3. 1's and 2's Complements of Binary Numbers

The 1's complement and 2's complement of a binary number are important because they allow us to represent the negative numbers. The method of 2's complement system (discussed later in Art. 6-7) is commonly used to handle negative numbers. Let us now study how the 1's complement and 2's complement of a binary number is obtained.

Obtaining 1's complement of a binary number. The 1's complement of a binary number is obtained by simply changing (or complementing) all 1s to 0s and all 0s to 1s. For example, the 1's complement of 10110010 is obtained as follows:

MSB							LSB	
1	0	1	1	0	0	1	0	— Binary number
↓	↓	↓	↓	↓	↓	↓	↓	
0	1	0	0	1	1	0	1	— 1's complement

∴ The 1's complement of 10110010 = 01001101

Obtaining 2's Complement of a binary number. The 2's complement of a binary number is obtained by adding 1 to the LSB (least-significant bit) of the 1's complement. For example, 2's complement of 10110010 is obtained as follows:

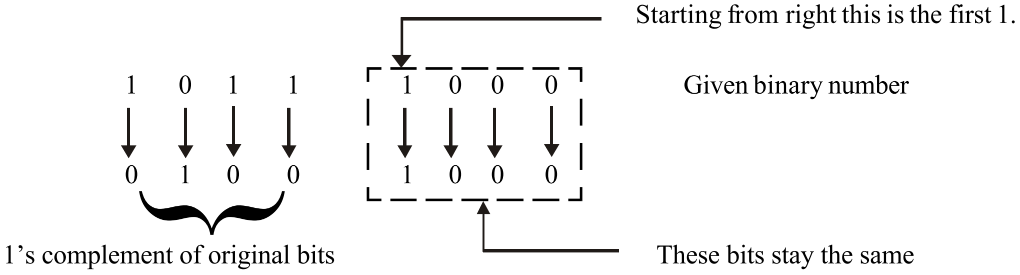
	MSB							LSB	
	1	0	1	1	0	0	1	0	— Binary number
	↓	↓	↓	↓	↓	↓	↓	↓	
	0	1	0	0	1	1	0	1	— 1's complement
+								1	
	0	1	0	0	1	1	1	0	— 2's complement

∴ The 2's complement of 10110010 is 01001110

An alternative method of obtaining 2's complement of a binary number is as follows:

1. Start at the right with the LSB (least-significant bit) and write the bits as they are up to and including the first 1.
2. Take 1's complement of the remaining bits.

Let us illustrate the procedure by finding the 2's complement of 10111000.


Starting from right this is the first 1.
1 0 1 1
1 0 0 0
Given binary number
0 1 0 0
1 0 0 0
1's complement of original bits
These bits stay the same

∴ 2's complement of 10111000 is 01001000.

Example 6-1. *Determine the* 2'*s complement of the following* 8-*bit binary numbers*:

(*a*) 00010110 (*b*) 11111100 (*c*) 10010001

Solution.

(*a*) **2's complement of 00010110**

$$\begin{array}{ccccccccc|l} & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & \text{— Given binary number}\\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & \text{— 1's complement}\\ + & & & & & & & & 1 & \\ \hline & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & \text{— 2's complement.}\end{array}$$

$\therefore$ The 2's complement of 00010110 is 11101010 **Ans.**

(*b*) **2's complement of 11111100**

$$\begin{array}{ccccccccc|l} & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & \text{— Given binary number}\\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \text{— 1's complement}\\ + & & & & & & & & 1 & \\ \hline & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \text{— 2's complement}\end{array}$$

$\therefore$ The 2's complement of 11111100 is 00000100 **Ans.**

(*c*) **2's complement of 10010001**

$$\begin{array}{ccccccccc|l} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \text{— Given binary number}\\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & \text{— 1's complement}\\ + & & & & & & & & 1 & \\ \hline & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & \text{— 2's complement}\end{array}$$

$\therefore$ The 2's complement of 10010001 is 01101111 **Ans.**

Example 6-2. *Obtain 2's complement of* $(10111011)_2$.

(*Gujarat Technological University., Dec. 2009*)

Solution. Given : The binary number 10111011.

$$\begin{array}{ccccccccc|l} & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & \text{— Given binary number}\\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & \text{— 1's complements}\\ + & & & & & & & & 1 & \\ \hline & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & \text{— 2's complement}\end{array}$$

$\therefore$ The 2's complement of 10111011 is 01000101 **Ans.**

Example 6-3. *2's complement of 1101010.*

(*Jamia University, 2007*)

Solution. Given : Binary number 1101010.

$$\begin{array}{ccccccccl}
 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & \text{— Given binary number} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & \text{— 1's complement} \\
+ & & & & & & & 1 & \\
\hline
 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & \text{— 2's complement}
\end{array}$$

∴ The 2's complement of 1101010 is 1010110 **Ans.**

Example 6-4. *Find 2's complement of the following numbers.*

(*a*) *01001110* (*b*) *01100100*

(*PTU., May 2008*)

Solution.

(*a*) **2's complement of 01001110**

$$\begin{array}{cccccccccl}
 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & \text{— Given binary number} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & \text{— 1's complements} \\
+ & & & & & & & & 1 & \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \text{— 2's complement}
\end{array}$$

∴ The 2's complement of 01001110 is 10110010 **Ans.**

(*b*) **2's complement of 01100100**

$$\begin{array}{cccccccccl}
 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & \text{— Given binary number} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \text{— 1's complements} \\
+ & & & & & & & & 1 & \\
\hline
 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \text{— 2's complement}
\end{array}$$

∴ The 2's complement of 01001110 is 10110010 **Ans.**

6-4. Representing Signed Numbers

A signed number is a number which consists of both sign and magnitude information. The sign indicates whether the number is positive or negative and the magnitude is the value of the number.

There are three methods of representing signed numbers. These are:

1. The sign magnitude system
2. The 1's complement system
3. The 2's complement system.

All these methods are discussed one by one in the following pages.

6-5. Sign-Magnitude System

In this system, the left-most bit of a binary number is the sign-bit. The sign-bit is 0 for positive and a 1 for negative binary numbers. The magnitude bits are in true (or uncomplemented) binary for both positive and negative numbers.

For example, the decimal number +13 is expressed as a binary number in a sign- magnitude system as

$$+13 = \underset{\text{Sign bit}}{0}\ \underbrace{1101}_{\text{Magnitude bits}}$$

The decimal number –13 is expressed as

$$-13 = \underset{\text{Sign bit}}{1}\ \underbrace{1101}_{\text{Magnitude bits}}$$

Notice the only difference between + 13 and – 13 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

Example 6-5. *Express the decimal number – 25 as a binary number using sign-magnitude system.*

Solution. Given the decimal number = – 25

$$-25 = \underset{\text{Sign bit}}{1}\ \underbrace{11001}_{\text{Magnitude bits}}$$

Notice that the sign bit is 1 because the given decimal number is negative.

∴ $-25_{10} = 111001_2$ **Ans.**

6-6. 1's Complement System

The 1's complement system for representing the signed numbers works as explained below:

If a number is positive— the sign bit is 0 and the magnitude of a binary number is represented in its true binary form.

If a number is negative— the sign bit is 1 and the magnitude of a binary number is represented as 1's complement of a corresponding positive number.

For example consider the decimal number +9. Its magnitude is represented as 1001, i.e., +9 =1001. When we attach a sign bit of 0, the complete signed number becomes 01001. However, if we consider a decimal number -9_{10}, it must be represented in 1's complement form.

$$\begin{array}{rcccccl} +9 = & 1 & 0 & 0 & 1 & & \\ & \downarrow & \downarrow & \downarrow & \downarrow & & \\ & 0 & 1 & 1 & 0 & & \text{— 1's complement} \end{array}$$

When we attach a sign bit of 1, the complete signed number becomes,

$$-9 = \underset{\text{Sign bit}}{1}\ 0110$$

6-7. 2's Complement System

The 2's complement system for representing the signed numbers works as explained below:

If a number is positive— the sign bit is 0 and the magnitude of a binary number is represented in its true binary form.

If a number is negative— the sign bit is 1 and the magnitude of a binary number is represented as 2's complement of a corresponding positive number.

For example consider the decimal number +19. Its magnitude is represented as 10011. When we attach a sign bit of 0, the complete signed number becomes 010011. However, if we consider – 19_{10}, it must be represented in 2's complement form.

$$\begin{array}{llccccccl}
19 & = & 1 & 0 & 0 & 1 & 1 & & \text{— Binary equivalent of } 19_{10} \\
 & & 0 & 1 & 1 & 0 & 0 & & \text{— 1's complement} \\
 & + & & & & & 1 & & \\
\hline
 & & 0 & 1 & 1 & 0 & 1 & & \text{— 2's complement}
\end{array}$$

When we attach a sign bit of 1, the complete signed number becomes,

$$-19_{10} = \underset{\text{Sign bit}}{\underset{\uparrow}{1}}\ 01101_2$$

6-8. Special Case in 2's Complement System

We have already discussed the 2's complement system in the last article. Now we will study some special cases in this system of representing signed binary numbers.

Whenever a signed binary number has a 1 in the sign bit and all 0s for the magnitude bits, its decimal equivalent is -2^N, where N is the number of bits in the magnitude. For example,

$$1000 = -2^3 = -8$$

$$10000 = -2^4 = -16$$

$$100000 = -2^5 = -32$$

and so on.

Thus we can state that the complete range of the values that can be represented in the 2's complement system having N magnitude bits is.

$$-2^N \text{ to } +(2^N-1)$$

In other words, there are 2^{N+1} different values including zero.

Fig. 6-1 lists all the signed numbers that can be represented in four bits using the 2's complement system (note that there are three magnitude bits and one sign bit). The sequence starts at $-2^N = -2^3 = -8 = 1000$ and proceed upwards to $+(2^N-1) = +2^3-1 = +7 = 0111$.

The 2's complement system is the most widely used method of representing signed binary numbers and performing arithmetic operations in computer systems. Most computer systems are based on 8-bit or 16-bit numbers. In a 8-bit system, the total number of different combinations of bits is 2^8 (decimals 256). On the other hand in a 16-bit system, the number is (2^{16} equals 65, 536).

In a 8-bit system, we have seven bits for magnitude and one bit for representing the sign. So the range of positive numbers in an 8-bit system is 0000 0000 to 01111111 (i.e. 0 to 127). On the other hand, the range of negative numbers is $1111\ 1111_2$ to 1000 0000 (i.e. – 1 to – 128).

Decimal Value	***Signed binary in 2's complement***
$+7 = 2^3 - 1$	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
– 1	1111
– 2	1110
– 3	1101
– 4	1100
– 5	1011
– 6	1010
– 7	1001
$-8 = -2^3$	1000

Fig. 6-1.

Example 6-6. *Determine the range of signed decimal values that can be represented in* 12 *bits* (*including the sign bit*).

Solution. We know that a 12-bit signed number will have one sign-bit and eleven bits to represent magnitude.

The largest negative value is,

$$1000\ \ 0000\ \ 0000 = -2^{11} = -2048_{10}$$

and the largest positive value is,

$$0111\ \ 1111\ \ 1111 = +2^{11} - 1 = +2047$$

Thus the complete range is – 2048 to + 2047; this is a total of 4096 different values, including zero. Alternatively, since there are eleven magnitude bits (N = 11), then there are $2^{N+1} = 2^{12} = 4096$ different values. **Ans.**

Example 6-7. *Determine the largest negative decimal value that can be represented by a two-byte number* ?

Solution. We know that a byte is a group of eight bits. So a two-byte number is a sixteen – bit number. Now since MSB (i.e. most significant bit) is used to represent a sign, then there are fifteen bits for magnitude.

∴ The largest negative value in a sixteen-bit number is,

$$= -2^{15} = -32,768 \quad \textbf{Ans.}$$

Example 6-8. *List an* 8-*bit representation of numbers by starting with positive number* + 7 to – 8, *using* 2'*s complement*.

Solution. Given the decimal numbers from + 7 to – 8.

Decimal Number	*Representation using 2's complement*
+7	0000 0111
+6	0000 0110
+5	0000 0101
+4	0000 0100
+3	0000 0011
+2	0000 0010
+1	0000 0001
0	0000 0000
– 1	1111 1111
– 2	1111 1110
– 3	1111 1101
– 4	1111 1100
– 5	1111 1011
– 6	1111 1010
– 7	1111 1001
– 8	1111 1000

Fig. 6-2.

We know that for all positive decimal numbers starting from +7 to 0 will have a sign bit as zero and the magnitude part as in the true binary form. Therefore the representation of these numbers is as shown in Fig. 6-2.

We also know that for all negative decimal numbers starting from – 1 to – 8 will have have a sign bit of '1' and the magnitude put in the 2's complement form.

Following the steps explained earlier we can obtain 2's complement of the magnitude bit of all the negative number. Attaching a sign bit-1 with the magnitude part, the number will become as shown in Fig. 6-2.

Example 6-9. *Express the decimal number* –39 *as an* 8-*bit number in sign- magnitude*, 1'*s complement and* 2'*s complement systems*.

Solution. Given the decimal number = – 39

Representation of –39 in sign- magnitude system.

First of all, let us write the 8-bit binary number for + 39. This can be written as:

$$+39 = 00100111$$

In the sign magnitude system, – 39 is produced by changing the sign-bit (i.e. MSB) to a 1 and leaving the magnitude bits unchanged. The number is.

$$\underset{\text{Sign-bit}}{\underset{\uparrow}{1}}\ \underbrace{0100111}_{\text{Magnitude}} = -39$$

$\therefore \quad -39 = 10100111$ **Ans.**

Representation of –39 in 1's complement system.

In 1's complement system, – 39 is produced by taking 1's complement of + 39. Thus,

$$\begin{array}{llcccccccccl} +39 & = & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & & \text{— Binary number}\\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \\ & & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & & \text{— 1's complement} \end{array}$$

$\therefore$ – 39 = 11011000 **Ans.**

Representation of – 39 in 2's complement system.

In 2's complement system, – 39 is produced by taking 2's complement of +39. Thus.

$$\begin{array}{llccccccccl} +39 & = & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & \text{— Binary number}\\ & & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & \text{— 1's complement}\\ & + & & & & & & & & 1 & \\ \hline & & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \text{— 2's complement} \end{array}$$

$\therefore$ – 39 = 11011001 **Ans.**

Example 6-10. *Using 2's complement system show the binary representation of* – 43.875_{10} *and* – 0.712_{10} (*Nagpur University Oct/ Nov. 1997*)

Solution. Given : the numbers – 43.875_{10} and – 0.712_{10}

Binary representation of – 43.875

Let us use 2's complement system to represent the given number into its equivalent binary.

$$\begin{array}{cccccccccccl} 0 & 1 & 0 & 1 & 0 & 1 & 1 & . & 1 & 1 & 1 & \text{— Binary equivalent of + } 43.875_{10}\\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & . & 0 & 0 & 0 & \text{— Complement each bit to form 1's complement}\\ & & & & & & & & & & +1 & \text{— add 1 to form 2's complement}\\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 & . & 0 & 0 & 1 & \text{— 2's complement of 43.875} \end{array}$$

Thus the binary representation of – 43.875 in 2's complement system is 1010100.001 **Ans.**

Binary representation of – 0.712.

Again use 2's complement system to represent the given number into its equivalent binary.

$$\begin{array}{cccccccccccl} 0 & . & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & & \text{— Binary equivalent of + } 0.712_{10}\\ \downarrow & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \\ 1 & & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & & \text{— Complement each bit to form 1's complement}\\ & & & & & & & & & +1 & & \text{— add 1 to form 2's complement}\\ \hline 1 & . & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & & \text{— 2's complement of + } 0.712_{10} \end{array}$$

Thus the binary representation of – 0.712_{10} in 2's complement system is 1.01001010_{10} **Ans.**

6-9. Evaluation of Signed Numbers

We have already discussed in Art 6-5, 6-6 and 6-7 the three different systems for representing signed numbers. Now we will study the evaluation of signed numbers, i.e. the reverse process. This means given the signed number expressed in any one of three systems: sign-magnitude, 1's complement or 2's complement system, how to obtain the decimal value of the signed number. This is discussed below separately for each signed number separately.

6-10. Evaluation of Numbers Represented in Sign-magnitude System

Positive and negative numbers in the sign-magnitude system are evaluated by summing the position values (or weights) in all the *magnitude* bit positions where there are 1s and ignoring those position values where there are 0s. The sign is determined by examination of the sign bit.

Consider for example the signed number 10010101 represented in sign magnitude system. The seven magnitude bits and their position values are as follows:

$$\begin{array}{cccccccl}
0 & 0 & 1 & 0 & 1 & 0 & 1 & \text{— Magnitude bits of signed number} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{— Position values of the bits} \\
\cancel{2^6} & \cancel{2^5} & 2^4 & \cancel{2^3} & 2^2 & \cancel{2^1} & 2^0 & \text{— Ignore position, values which correspond to 0's}
\end{array}$$

$2^4 + 2^2 + 2^1$ — Sum the position, values which correspond to 1's.

$= 16 + 4 + 1$

$= 21$

Since the sign bit is 1, therefore the decimal value is -21_{10}.

Example 6-11. *Determine the decimal value of the sign-magnitude number* 01110111.

Solution. Given the number represented in sign magnitude system = 01110111. The decimal value of this number can be obtained as shown below:

$$\begin{array}{cccccccl}
1 & 1 & 1 & 0 & 1 & 1 & 1 & \text{— Magnitude bits of the signed number} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{— Position value of the bits} \\
2^6 & 2^5 & 2^4 & \cancel{2^3} & 2^2 & 2^1 & 2^0 & \text{— Ignore position values, which correspond to 0's}
\end{array}$$

$2^6 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0$ — Sum the position values which correspond to 1s.

$= 64 + 32 + 16 + 4 + 2 + 1$

$= 119$

Since the sign bit is 0, therefore the decimal value is +119 **Ans.**

6-11. Evaluation of a Number Represented in 1's Complement System

Positive numbers in the 1's complement system are evaluated by summing the position values (or weights) in all bit positions where there are 1s and ignoring those positions where there are 0s. Negative numbers are evaluated by assigning a *negative value* to the position value of the sign bit, summing all the position values where there are 1s and adding 1 to the result.

Consider for example, the evaluation of the signed number 00010111_2. The decimal value of this number is determined as shown below:

$$\begin{array}{ccccccccl}
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & \text{— 8-bits of the 1's complement number} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
-2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{— Position value of the bits} \\
\uparrow & & & & & & & &
\end{array}$$

Note that the sign bit is
assigned the –ve value.

$-\cancel{2^7}\ \cancel{2^6}\ \cancel{2^5}\ 2^4\ \cancel{2^3}\ 2^2\ 2^1\ 2^0$ — Ignore position values which correspond to 0s

$2^4+2^2+2^1+2^0$ — Sum the position values which correspond to 1s

$= 16+4+2+1$

$= 23$

∴ The decimal value of the signed number 00010111(represented in 1's complement system) is +23.

Example 6-12. *Determine the decimal value of the* 1'*s complement number* 11101011.

Solution. Given, the 1's complement number = 11101011

In order to determine its decimal value, we write the number and the position value under each bit. Recall that the sign bit is to be assigned a negative position value. Sum all the position values where there are 1s and ignore those positions where there are 0s. If the sum is negative add 1 to the result.

$$\begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ -2^7 & 2^6 & 2^5 & \cancel{2^4} & 2^3 & \cancel{2^2} & 2^1 & 2^0 \end{array}$$

$= -\ 2^7+2^6+2^5+2^3+2^1+2^0$

$= -128+64+32+8+2+1$

$= -\ 21$

Since the number is negative, we add 1 to the result, therefore the final number,

$$= -21+1 = -20_{10}\ \textbf{Ans.}$$

6-12. Evaluation of a Number Represented in 2's Complement System

Positive and negative numbers in 2's complement system are evaluated by summing the position values (or weights) in all bit positions where there are 1s and ignoring those positions where there are 0s. The position value of the sign bit in the negative number is given a negative value.

Consider for example, the evaluation of 2's complement number 01010110_2. The decimal value of this number is determined as shown below:

$$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \uparrow & & & & & & & \end{array}$$

— 8-bits of 2's complement number.

— Position value of the bits

Note that the sign bit is assigned the –ve value.

$-\cancel{2^7}\ 2^6\ \cancel{2^5}\ 2^4\ \cancel{2^3}\ 2^2\ 2^1\ \cancel{2^0}$ — Ignore position values which correspond to 0s.

$2^6+2^4+2^2+2^1$ — Sum the position values which correspond to 1s

$= 64+16+4+2$

$= +\ 86$

∴ The decimal value of the 2's complement number is + 86.

Now let us consider another signed number 11101000_2 represented in 1's complement system. The decimal value of this number is determined as shown below:

$$\begin{array}{ccccccccl} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & \text{— 8-bits of the signed number.} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{— Position value of the bits.} \\ \uparrow & & & & & & & & \end{array}$$

Note that the sign bit
is assigned the –ve value.

$$\begin{array}{ll} -2^7\ \ 2^6\ \ 2^5\ \ \cancel{2^4}\ \ 2^3\ \ \cancel{2^2}\ \ \cancel{2^1}\ \ \cancel{2^0} & \text{— Ignore position values which correspond to 0s.} \\ -2^7 + 2^6 + 2^5 + 2^3 & \text{— Sum the position values which correspond to 1s.} \\ = -128 + 64 + 32 + 8 & \\ = -24 & \end{array}$$

Since the sum is negative, so we add 1 to the result. Therefore the final number is,

$$-24 + 1 = 23$$

Thus the decimal value of the signed number 11101000 (represented in 1s complement system) is – 23.

To summarize if the sign bit is positive sum the position values in all the magnitude bit positions where there are 1s and ignore those positions where there are 0s. However if the sign bit is negative, then first complement all the magnitude bits, then sum all the magnitude bit positions where there are 1s and ignore those positions where there are 0s.

Now let us consider another 2's complement number 10101010. The decimal value of this number is determined as shown below.

$$\begin{array}{ccccccccl} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \text{— 2's complement number.} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{— Position value of the bits} \\ \uparrow & & & & & & & & \end{array}$$

Note that the sign-bit
is assigned a –ve value.

$$\begin{array}{ll} -2^7\ \cancel{2^6}\ \ 2^5\ \ \cancel{2^4}\ \ 2^3\ \ \cancel{2^2}\ \ 2^1\ \ \cancel{2^0} & \text{— Ignore position values which correspond to 0's} \\ = -128 + 32 + 8 + 2 & \text{— Sum the position values which correspond to 1's} \\ = -86 & \end{array}$$

∴ The decimal value of the 2's complement number 10101010 is – 86.

To summarize, if the sign-bit is positive, sum the position values in all the magnitude bit positions where there are 1s and ignore those positions where there are 0s. However if the sign-bit is negative, then first complement all the magnitude bits, then add one to the least-significant bit. Then sum all the magnitude bit positions where there are 1s and ignore those positions where there are 0s.

Example 6-13. *Determine the decimal value of the* 2'*s complement number* 111010111.

Solution. Given the 2's complement number = 111010111.

$$\begin{array}{cccccccccl} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & \text{— 2's complement number} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ -2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{— Position value of the bit.} \\ \uparrow & & & & & & & & & \end{array}$$

Note that the sign bit
is assigned -ve value

$-2^8 \quad 2^7 \quad 2^6 \quad \cancel{2^5} \quad 2^4 \quad \cancel{2^3} \quad 2^2 \quad 2^1 \quad 2^0$ — Ignore position values which correspond to 0s.

$-2^8 + 2^7 + 2^6 + 2^4 + 2^2 + 2^1 + 2^0$ — Sum the position values which correspond to 1s.

$= -256 + 128 + 64 + 16 + 4 + 2 + 1$

$= -41$

∴ The decimal value of the 2's complement number 111010111_2 is -41_{10} **Ans.**

Example 6-14. *Determine the range of unsigned decimal numbers that can be represented in a byte.*

Solution. We know that a byte is a group of eight bits. Since we are interested in unsigned decimal numbers here, therefore there is no sign bit. This means we can use all the eight bits to represent a magnitude of a number. Therefore the values of unsigned decimal numbers range from.

$$00000000_2 = 0_{10} \text{ to } 11111111_2 = 255_{10}$$

This is a total of 256 different numbers which we could have predicted, since $2^8 = 256$ **Ans.**

6-13. Addition of Two Binary Numbers

We have already discussed the three different methods of representing binary numbers namely (1) signed system (2) 1's complement system and (3) 2's complement system. Now let us study the addition of two binary numbers. We will use 2's complement system for representing binary numbers as this is the system used in all digital systems to do arithmetic operations.

In order to illustrate the addition operation in 2's complement system, we will consider the following five different types of cases :

1. Both the binary numbers are positive
2. One of the numbers is positive and the other one is negative. It will be assumed that negative number is smaller than the positive number.
3. Again one of the numbers is positive and the other one is negative. But in this case the negative number is larger than the positive number.
4. Both the numbers are negative.
5. Both the numbers are equal and opposite.

All these five different cases for addition are discussed one by one in the following pages.

6-14. Addition of Two Positive Numbers

Consider the addition of two positive numbers say +7 and +5. The addition operation for these two numbers is as shown below:

$$\begin{array}{rcl} +7 \rightarrow & 0 & 0111 \\ +5 \rightarrow & 0 & 0101 \\ \hline & \boxed{0} & 1100 \quad (\text{Sum} = +12) \\ & \uparrow & \\ \text{Sign bits} \longrightarrow & & \end{array}$$

Notice that the sign bits of both the numbers are 0 and the sum of sign bit is 0. This indicates that sum is positive. Also notice that both numbers have same number of bits. This must always be done while performing arithmetic operations in 2's complement system.

6-15. Addition of a Positive Number and a Small Negative Number

Consider the addition of two binary numbers where one of the numbers is positive and the other one is a negative number. It is also under consideration that negative number is a smaller number. Let us suppose we have the two numbers as +7 and –5. Remember that the number – 5 will be in its 2's complement form. Thus +5 (00101) must be converted to – 5(11011). The addition operation for this situation is as shown below.

$$\begin{array}{rcll} & & & \text{Sign bits} \\ +7 \rightarrow & 0 & 0111 & \\ -5 \rightarrow & 1 & 1011 & \\ \hline \cancel{1} & \boxed{0} & 0010 & \end{array}$$

This carry is disregarded; the result is 0010 (= +2).

Notice that the sign bit of negative number is 1. Further the sign bits also participate in the addition process. As a matter of fact, a carry is generated in the last position of addition. This carry is always disregarded, so that the final sum is 0010 which is equal to +2.

6-16. Addition of a Positive Number and a Large Negative Number

Consider the addition of two numbers where one of the numbers is positive and the other one is negative-the negative number is larger than the positive number. Let us suppose we have two numbers –7 and +5. Remember that number –7 will be in its 2's complement form. Thus +7 (00111) must be converted to – 7 (i.e. 11001). The addition operation of this situation is as shown below:

$$\begin{array}{rcll} & & & \text{Sign bits} \\ -7 \rightarrow & 1 & 1001 & \\ +5 & 0 & 0101 & \\ \hline & 1 & 1110 & (\text{Sum} = -2) \end{array}$$

Negative sign bit

Notice that the sum here has a sign bit of 1, indicating a negative number. Since the sum is negative, it is in the 2's complement form.

To find the true magnitude of the sum, we must take 2's complement of the result, 11110

$$\begin{array}{rccccc l} & 1 & 1 & 1 & 1 & 0 & \text{— Result} \\ & 0 & 0 & 0 & 0 & 1 & \text{— 1's complement} \\ + & & & & & 1 & \\ \hline & 0 & 0 & 0 & 1 & 0 & \text{— 2's complement} \end{array}$$

Thus 2's complement of the result (11110) is 00010 = +2. Hence the result 11110 represents –2.

6-17. Addition of Two Negative Numbers

Let us now study the addition of two negative numbers. In order to illustrate the procedure, consider the two numbers as –7 and – 5. Remember that both numbers –7 and –5 will be in their 2's complement form. Thus +7 (00111) must be converted to –7 (i.e., 11001). Similarly, +5 (00101) must be converted to – 5 (i.e. 11011). The addition operation of –7 and –5 is as shown below:

Sign bits

$$\begin{array}{rcc} -7 \rightarrow & 1 & 1001 \\ -5 \rightarrow & 1 & 1011 \\ \hline 1 & 1 & 0100 \end{array}$$

This carry is disregarded; the result is 10100 (sum = –12).

Notice that the final result is negative and in 2's complement form with a sign bit of 1. Taking 2's complement of result (10100), we get

$$\begin{array}{cccccccl} & 1 & 0 & 1 & 0 & 0 & & \text{— result} \\ & 0 & 1 & 0 & 1 & 1 & & \text{— 1's complement of result} \\ + & & & & & 1 & & \\ \hline & 0 & 1 & 1 & 0 & 0 & & \text{— 2's complement of result} \end{array}$$

Thus we find that 2's complement of result is 01100 = +12. Therefore the sum of two negative numbers – 7 and – 5 is – 12.

6-18. Addition of Two Equal and Opposite Numbers

Consider the addition of two equal and opposite numbers say + 7 and – 7. The addition of these two numbers is as shown below.

$$\begin{array}{rcc} +7 \rightarrow & 0 & 0111 \\ -7 \rightarrow & 1 & 1001 \\ \hline \cancel{1} & 0 & 0000 \end{array}$$

Sign bits

disregard this carry; the result is 00000 (Sum = + 0)

Notice that -7 is in 2's complement form. We find that the result is +0 as expected.

Example 6-15. *Perform the following addition operation in the* 2'*s complement system. Use eight* (*including the sign bit*) *for each number. Check your results by converting binary result back to decimal.*

(*a*) 32_{10} *and* 18_{10} (*b*) $+21_{10}$ *and* -13_{10}

(*c*) -28_{10} *and* $+38_{10}$ (*d*) -36_{10} *and* -15_{10}

(*e*) $+17_{10}$ *and* -17_{10}

Solution.

(*a*) **Addition of 32_{10} and 18_{10}.** We know that in this case both the numbers are positive. Therefore the addition is carried out as follows:

Sign bits

$$\begin{array}{rccl} +32_{10} \rightarrow & 0 & 0100000 & \\ +18_{10} \rightarrow & 0 & 0010010 & \\ \hline 50_{10} & 0 & 0110010 & (\text{Sum} = +50) \end{array}$$

Notice that the sign bits of both numbers are 0 and the sum of sign bit is 0. Therefore sum is positive. Thus using 2's complement system we find that,

$$32_{10} + 18_{10} = 50_{10} \textbf{ Ans.}$$

(*b*) **Addition of +21$_{10}$ and –13$_{10}$.** We know that in order to add – 13$_{10}$ to + 21$_{10}$, we will have to find 2's complement of –13$_{10}$. Thus

Sign bits

$$\begin{array}{lrl} +21 \rightarrow & 0 & 0010101 \\ -13 \rightarrow & 1 & 1110011 \\ \hline +8 & \cancel{1}\ 0 & 0001000 \\ \hline \end{array}$$

This carry is disregarded; the result is 00001000 (= 8$_{10}$).

Thus using 2's complement we find that 21$_{10}$ –13$_{10}$ is +8$_{10}$ **Ans.**

(*c*) **Addition of – 28$_{10}$ and + 38$_{10}$.** In order to add –28$_{10}$ to +38$_{10}$, we must represent – 28$_{10}$ in its 2's complement form. Thus

Sign bits

$$\begin{array}{lrl} -28_{10} \rightarrow & 1 & 11100100 \\ +38_{10} \rightarrow & 0 & 00100110 \\ \hline +10_{10} & \cancel{1}\ 0 & 0001010 \\ \hline \end{array}$$

This carry is disregarded; the result is 00001010 (= 10$_{10}$).

Thus using 2's complement, we find that – 28$_{10}$ + 38$_{10}$ is +10$_{10}$ **Ans.**

(*d*) **Addition of – 36$_{10}$ and – 15$_{10}$.** In order to add –36$_{10}$ and –15$_{10}$, we must represent these numbers to their 2's complement form. Thus

Sign bits

$$\begin{array}{lrll} -36 \rightarrow & 1 & 1100100 & \\ -15 \rightarrow & 1 & 1110001 & \\ \hline -51 & \cancel{1}\ 1 & 1001101 & \text{(Sum} = -51) \\ \hline \end{array}$$

This carry is disregarded; the result is 11001101 (= – 51$_{10}$).

Notice that the find result is negative and in 2's complement form with a sign bit of 1. Taking 2's complement of result (11001101), we get

$$\begin{array}{rccccccccl} & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & \text{— result} \\ & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \text{— 1's complement of result} \\ + & & & & & & & & 1 & \\ \hline & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \text{— 2's complement of result} \end{array}$$

Thus we find 2's complement of result is 00110011 = – 51$_{10}$ **Ans.**

(*e*) **Addition of + 17$_{10}$ and – 17$_{10}$.** We know that this is a case of two equal and opposite numbers. The addition of these two numbers is as shown below:

Sign bits

$$\begin{array}{lrl} +17 \rightarrow & 0\ \ 0 & 010001 \\ -17 \rightarrow & 1 & 1101111 \\ \hline & \cancel{1}\ 0 & 0000000 \\ \hline \end{array}$$

disregard this carry; the result is 00000000 (= 0$_{10}$).

Thus using 2's complement system we find that the result is + 0 as expected.

6-19. Arithmetic Subtraction

Subtraction is a special case of addition. When subtracting one binary number (called subtrahend), from another binary number (called the minuend), we can use the following two step procedure:

1. Negate (i.e. find 2's complement of) the subtrahend- This will change the subtrahend to its equivalent value of opposite sign.
2. Add this to the minuend- The result of this addition will represent the difference between the subtrahend and the minuend.

It may be carefully noted that both numbers have the same number of bits in their representations. This is valid in all 2's complement arithmetic operations.

In order to illustrate the subtraction, let us consider the situation where +5 is to be subtracted from +7. The subtraction is shown as below.

+7 → 00111 (minuend)

+5 → 00101 (subtrahend)

If we take 2's complement of the subtrahend, we get 11011 which represents –5. Now add this to the minuend.

Sign bits

+ 7 → 0 0111 (minuend)

– 5 → 1 1011 (2's complement of subtrahend)

~~1~~ 0 0010

disregard, so the result is 00010 = + 2

It is evident from the above discussion that when the subtrahend is changed to its 2's complement, it actually becomes –5, so we are adding –5 and +7. As a matter of fact, this is the same as subtracting +5 from +7. This is actually the same situation as discussed in Art 6-15. Thus we find that any subtraction operation actually becomes one of addition when the 2's complement system is used. This feature of 2's complement system has made it the most widely used methods available. This is possible because it allows addition and subtraction to be performed by the same circuitry.

Let us consider another example of subtraction. This time we will illustrate the procedure of subtracting + 7 from – 5.

Sign bits

– 5 → 1 1011 (minuend)

– 7 → 1 1001 (2's complement of + 7)

~~1~~ 1 0100

disregard; so the result is 10100 = (Equal to – 12)

Notice that – 5 is in 2's complement form. Also note that we have negated +7 to perform subtraction. Since the result (10100) is negative, it has to converted to its 2's complement, which is 01100 (i.e.+12). Therefore the subtraction of + 7 from – 5 is obviously – 12.

The student is strongly recommended to verify the results of the above procedure for the following subtractions:

(*a*) +7 – (– 5) (*b*) – 7 – (+5)

(*c*) – 7 – (– 5) (*d*) + 5 – (– 5)

You must remember always that when the result has a sign bit of 1, it is negative and in 2's complement form.

Example 6-16. *Subtract* 78_{10}*from* 97_{10} *by using* 2'*s complement arithmetic*

(*Nagpur Univ. Oct/Nov. 1997*)

Solution. Given the two decimal numbers 78_{10} and 97_{10}.

In order to subtract 78 from 97, we will have to find its 2's complement. Then add this number to the binary equivalent of 97. Thus

$$\begin{array}{lcccccccccl}
 & \downarrow & & & & & & & & \text{Sign bit} & \\
+78 \rightarrow & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & & \text{— binary equivalent of 78} \\
 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & & \text{— 1's complement of 78} \\
+ & & & & & & & & 1 & & \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & & \text{— 2's complement of 78}
\end{array}$$

Now,

$$\begin{array}{llcccccccccl}
 & & \downarrow & & & & & & & & \text{Sign bits} & \\
97 \rightarrow & & \boxed{0} & 1 & 1 & 0 & 0 & 0 & 0 & 1 & & \text{— binary equivalent of 97} \\
-78 & & \boxed{1} & 0 & 1 & 1 & 0 & 0 & 1 & 0 & & \text{— 2's complement of 78} \\
\overline{19} & + & & & & & & & & & & \\
\hline
 & \cancel{1} & \boxed{0} & 0 & 0 & 1 & 0 & 0 & 1 & 1 & &
\end{array}$$

disregard; so the result is 10011.
This is equivalent to 19_{10}.

$\therefore$ $97_{10} - 78_{10} = 19_{10}$ **Ans.**

Example 6-17 *Using* 2'*s complement method, perform the subtraction* $101_2 - 111_2$.

(*Nagpur University, Oct/Nov. 1997*)

Solution. Given the two binary numbers 101_2 and 111_2. In order to subtract 111_2 from 101_2, we will have to find the 2's complement of 101_2 and then add this to the binary 101_2. Thus 2's complement of 111_2 is 001_2.

$$\therefore \qquad \begin{array}{lccl}
 & \downarrow & & \text{Sign bits} \\
 & \boxed{0} & 101 & \\
+ & \boxed{1} & 001 & \\
\hline
 & 1 & 110 &
\end{array}$$

Since the sign bit of the result is 1, the result is in 2's complement . Thus taking 2's complement of 1110, we get 0010. Therefore the result is -0010_2 or -2_{10} **Ans.**

Example 6-18. *Perform 7-15 using 2's complementary method.*

(*Gauhati University, 2007*)

Solution. Given : The two decimal number 7_{10} and 15_{10}. In order to subtract 7 from 15, we will have to find its 2's complement. Then add this number to binary equivalent of 15. Thus

$$\begin{array}{llccll}
 & & \downarrow & & \text{Sign bit} & \\
7 \rightarrow & & \boxed{0} & 0111 & & \text{— binary equivalent of 7} \\
-15 \rightarrow & + & \boxed{1} & 0001 & & \text{— 2's complement of 15} \\
\hline
-8 & & \boxed{1} & 0001000 & & \\
\hline
 & & 1 & 1000 & \text{This is equivalent to} - 810 &
\end{array}$$

$7_{10} - 15_{10} = -8_{10}$ **Ans.**

Example 6-19. *Perform the subtraction by 2's complement method: 01100 – 00011*

(*Assam Engineering College, 2006*)

Solution. Given the two binary number 01100 – 00011

We have to find the 2's complement of 00011. Then add this number to the binary number 011000.

$$\begin{array}{ccccc l}
0 & 0 & 0 & 1 & 1 & \text{— binary number} \\
1 & 1 & 1 & 0 & 0 & \text{— 1s complement} \\
 & & & & 1 & \\
\hline
1 & 1 & 1 & 0 & 1 & \text{— 2s complement}
\end{array}$$

Now,

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

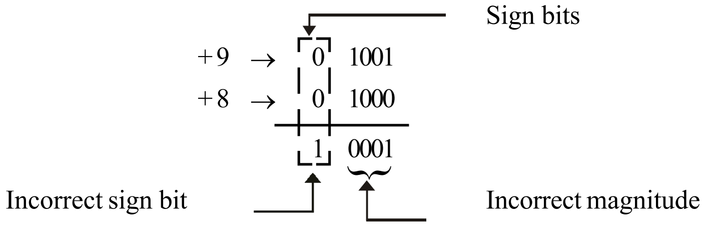
Carry

Discard the carry bit.

1001 **Ans.**

6-20. Arithmetic Overflow

We have already discussed in previous articles, the addition and subtraction of two binary numbers using 2's complement system. In each of the examples used to illustrate the addition or subtraction, the numbers consisted of one sign bit and four magnitude bits. The results also consisted of a sign bit and four magnitude bits. Any carry into the sixth bit was disregarded. In all of the cases we have considered, the magnitude of the answer was small enough to fit into four bits. Let us look at the addition of +9 and +8.



It is evident from the above that the answer has a negative sign bit, which is obviously incorrect because we are adding two positive numbers. Notice that the result of+9 and +8 should be +17, and the magnitude 17 requires more than four bits. Because of this reason, the magnitude overflows into the sign bit position. The overflow condition can occur only when two positive or two negative numbers are being added. Further, the overflow condition produces an incorrect result. The overflow can be detected by checking the sign bit of the result is the same an the sign bits of the number being added.

It may be noted that overflow can occur while performing subtraction if minuend and subtrahend are of different signs. For example, if we are subtracting –8 from +9, the –8 is negated to become +8 and is added to + 9 just as shown above. This causes an overflow and produces an erroneous negative result since the magnitude is too large

It will be interesting to know that all computers have a special circuit to detect any overflow condition whenever two numbers are added or subtracted.

6-21. Arithmetic Multiplication

As a matter of fact, the multiplication of binary numbers is done in the same manner as the multiplication of decimal numbers. But the process is much simpler because we are multiplying by 0 or 1 and no other digits. Let us illustrate the multiplication by a following example.

$$
\begin{array}{cccccccccl}
 & & & & 1 & 0 & 1 & 1 & \leftarrow & \text{multiplicand} = 11_{10} \\
 & & & & 1 & 1 & 0 & 1 & \leftarrow & \text{multiplier} = 13_{10} \\
\cline{2-8}
 & & & & 1 & 0 & 1 & 1 & & \\
 & & & 0 & 0 & 0 & 0 & & & \text{Partial products} \\
 & & 1 & 0 & 1 & 1 & & & & \\
 & 1 & 0 & 1 & 1 & & & & & \\
\cline{2-8}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \leftarrow & \text{Final product} = 143_{10}
\end{array}
$$

As seen, the multiplicand and the multiplier are in true binary form and no sign bit is used. Notice that the steps used in multiplication process are exactly the same as used in decimal multiplication.

1. First, we examine the least-significant bit (LSB) of the multiplier. In our case it is 1. This 1 when multiplied by the multiplicand produces 1011. This is written as the first partial product.
2. Next the second bit of the multiplier is examined. Since it is 0, when multiplied by the multiplicand, it produces 0000. This is written as the second partial product. Notice that it is shifted one place to the left relative to the first partial product.
3. Next the third bit of the multiplier is examined. Since it is 1 and so 1011 is written as the third partial product. Again note that this third partial product is shifted one place to the left relative to the second one.
4. Next, the fourth bit is 1 and so the last partial product is 1011 shifted again one position to the left.
5. Finally the four partial products are then summed to produce the final product.

It may be carefully noted that most digital circuits can add only two binary numbers at a time. Because of this reason, the partial products formed during multiplication cannot be all added together at the same time. Instead, they are added together two at a time. That is the first partial product is added to the second, their sum is added to the third and so on. The process of addition is shown below.

$$
\begin{array}{lcccccccccl}
\text{Add} & & & & 1 & 0 & 1 & 1 & & \leftarrow & \text{first partial product} \\
 & & & & 0 & 0 & 0 & 0 & & \leftarrow & \text{second partial product shifted left} \\
\cline{2-9}
\text{Add} & & & 0 & 1 & 0 & 1 & 1 & & \leftarrow & \text{sum of first two partial products} \\
 & & & 1 & 0 & 1 & 1 & & & \leftarrow & \text{third partial product shifted left} \\
\cline{2-9}
\text{Add} & & 1 & 1 & 0 & 1 & 1 & 1 & & \leftarrow & \text{sum of first three partial products} \\
 & & 1 & 0 & 1 & 1 & & & & \leftarrow & \text{fourth partial product shifted left} \\
\cline{2-9}
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \leftarrow & \text{sum of four partial products.}
\end{array}
$$

This equals final total product

6-22. Multiplication in 2's Complement System

We have already discussed in last article that the multiplication of binary numbers is carried out in the same manner as for decimal numbers. This method is used by the computer (that use the 2's complement representation), provided that both the multiplicand and the multiplier are put in true binary form. If the two numbers to be multiplied are positive, they are already in true binary form and are multiplied as they are. The result product is also positive and is given a sign bit of 0.

On the other hand, when the two numbers are negative, they will be in 2's complement form. The 2's complement of each number is taken to convert it into a positive number and then the two numbers are multiplied . The product is kept as a positive number and is given a sign bit of 0. It may be noted that when one of the numbers is positive and the other is negative, the negative number is first converted to a positive magnitude by taking its 2's complement. The product will be in true –magnitude form. However the product must be negative, since the original numbers are of opposite sign. Thus the product is then changed to 2's complement form and is given a sign bit of 1

Example 6.20. *Perform the following operations:*

(*i*) *1 1 1 1 1 + 1 1 1 1* (*ii*) *1 1 0 0 1 – 1 0 0 0 1* (*iii*) *1 0 1 1 * 1 0 1*

(*Jamia University, 2007*)

Solution. (*i*) Given : 1 1 1 1 1 + 1 1 1 1

We know that,

$$\begin{array}{cccccccl} & & 1 & 1 & 1 & 1 & 1 & (31) \\ & + & & 1 & 1 & 1 & 1 & (15) \\ \hline & 1 & 0 & 1 & 1 & 1 & 0 & (46) \end{array}$$

∴ 11111 + 1111 = 101110 **Ans.**

(*ii*) Given 11001 – 10001

$$\begin{array}{rcccccccc} & & & 1 & 1 & 0 & 0 & 1 & \\ \text{1's complement of } 10001 - & & & + & 0 & 1 & 1 & 1 & 0 \\ \hline \text{Sum} & = & 1 & 0 & 0 & 1 & 1 & 1 & \\ \text{End-around carry} & & \multicolumn{4}{l}{\longrightarrow} & + & 1 & \\ \hline & & & 0 & 1 & 0 & 0 & 0 & \end{array}$$

0 1 0 0 0

drop the 0 we get

1 0 0 0

∴ 11001 – 10001 = 1000 **Ans.**

(*iii*) Given : 1011 * 101

$$\begin{array}{ccccccl} & & 1 & 0 & 1 & 1 & \leftarrow \text{multiplicand} = 11_{10} \\ \hline & & & 1 & 0 & 1 & \leftarrow \text{multiplier} = 5_{10} \\ & & 1 & 0 & 1 & 1 & \Bigg\} \text{ partial products} \\ & 0 & 0 & 0 & 0 & \times & \\ \hline 1 & 0 & 1 & 1 & \times & \times & \\ 1 & 1 & 0 & 1 & 1 & 1 & \leftarrow \text{Final product} = 55_{10} \end{array}$$

∴ 1011 × 101 = 110111 **Ans.**

6-23. Binary Division

As a matter of fact the process for dividing one binary number (called dividend) by another (called divisor) is the same as that which is followed by decimal numbers. The actual process(refered to as long division) is simpler in binary because when we are checking to see how many times the divisor goes into the dividend, there are only two possibilities, 0 and 1. Let us illustrate the division process by the following simple division examples,

$$\begin{array}{lr}
 & 0011 \\
 & 11\overline{\smash{)}\,1001} \\
 & \underline{011} \\
\text{First partial remainder} \longrightarrow & 0011 \\
 & \underline{11} \\
\text{Second partial remainder} \longrightarrow & 0
\end{array}$$

1001 → dividend

11 → divisor

0011 → quotient

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

The above example shows that we have 1001_2 divided by 11_2. This is equivalent to 9 ÷ 3 in decimal. Notice that the resulting quotient is $0011_2 = 3_{10}$.

In digital computers, the division is carried out by using repeated subtraction using 2's complement system. Following is the procedure used for division in most of the digital circuits.

1. Determine if the signs of the dividend and divisor are the same or different. This determines what the sign of the quotient will be. Intialize the quotient to 0.
2. Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient. If this partial remainder is positive go to step 3. However if the partial remainder is zero or negative, the division is complete.
3. Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive, repeat for the next partial remainder. If the result is zero or negative, the division is complete.

It is evident from the above procedure that we continue to subtract the divisor from the dividend and the partial remainders until there is a zero or negative result.

Example 6-21. *Divide the number* 221_{10} *by* 17_{10} *by converting the original decimal number to its 8-bit binary equivalent.*

Solution. Given the two decimal numbers 221 and 17. We know that the 8-bit binary equivalent of 221_{10} is,

$$221_{10} = 11011101_2$$

and that of 17_{10} is

$$17_{10} = 0001\ 0001_2$$

Let us now divide 11011101_2 by $0001\ 0001_2$ as shown below.

$$\begin{array}{lrl}
 & 1101 & \leftarrow \quad \text{quotient} \\
0001\ 0001 & \overline{\smash{)}\,1101\ 1101} & \leftarrow \quad \text{dividend} \\
\uparrow\ \text{divisor} & \underline{-\,1000\ 1} & \\
 & 101\ 01 & \\
 & \underline{-\,100\ 01} & \\
 & 1\ 0001 & \\
 & \underline{-\,1\ 0001} & \\
 & 0 &
\end{array}$$

From the division process shown above we find that, $11011101_2 \div 00010001_2 = 1101_2 = 13_{10}$. **Ans.**

Note. Check that $221_{10} \div 17_{10} = 13_{10}$ which is found to be the same as through the binary division.

6-24. BCD Addition

We have already discussed in chapter 2 that many computers and calculators use the BCD code to represent decimal numbers. Recall that this code takes each decimal digit and represents it by a four-bit code ranging from 0000 to 1001. The addition of decimal numbers that are in BCD form can be best understood by considering the two cases that can occur when two decimal digits are added.

1. **Sum equals 9 or less.** Let us consider the addition of 5 and 4 using BCD to represent each digit.

$$\begin{array}{rrl} 5 & 0101 & \leftarrow \text{BCD code for 5} \\ \underline{+\,4} & \underline{0100} & \leftarrow \text{BCD code for 4} \\ 9 & 1001 & \leftarrow \text{BCD code for 9} \end{array}$$

As seen the addition is carried out in normal binary addition and the sum is 1001 which is the BCD code for 9. Consider another example,

$$\begin{array}{rrl} 54 & 0101\ 0100 & \leftarrow \text{BCD code for 54} \\ \underline{+\,33} & \underline{0011\ 0011} & \leftarrow \text{BCD code for 33} \\ 87 & 1000\ 0111 & \leftarrow \text{BCD code for 87} \end{array}$$

It may be noted that the four-bit codes for 4 and 3 are added in binary to produce 0111 which is a BCD code for 7. Similarly, if we add the second decimal digit positions, it produces 1000 which is BCD code for 8. The total is 1000 0111 which is BCD code for 87.

It may be carefully noted that in none of the examples discussed above, the sums of the pairs of decimal digits exceeded 9. Therefore no decimal digits were produced. For these cases, the BCD addition process is straight forward. In fact the addition process is the same as binary addition.

2. **Sum Greater than 9.** Let us consider the addition of 7 and 5 using BCD to represent each digit.

$$\begin{array}{rrl} 7 & 0111 & \leftarrow \text{BCD code for 7} \\ \underline{+\,5} & \underline{0101} & \leftarrow \text{BCD code for 5} \\ +\,12 & 1100 & \leftarrow \text{Invalid BCD code} \end{array}$$

It may be carefully noted that the sum 1100 does not exist in the BCD code. As a matter of fact, it is one of the six forbidden or invalid four-bit code groups. This has occurred because of the sum of two digits exceed 9. Whenever this occurs, the sum must be corrected by the addition of six (0110) to take into account the skipping of the six invalid code groups. Thus,

$$\begin{array}{rrl} & 0111 & \leftarrow \text{BCD code for 7} \\ & \underline{+\,0101} & \leftarrow \text{BCD code for 5} \\ & 1100 & \leftarrow \text{Invalid BCD code} \\ & \underline{+\,0110} & \leftarrow \text{Add 6 for correction} \\ \underbrace{0001}_{1} & \underbrace{0010}_{2} & \leftarrow \text{BCD code for 12.} \end{array}$$

It is evident from the illustration above that if 0110 is added to the invalid BCD code, it produces a correct BCD result. It may be noted that with the addition of 0110, a carry is produced into the second decimal position. The addition must be performed whenever the sum of the two decimal digits is greater than 9.

Example 6-22. *Add the following decimal numbers after converting each to its BCD code.*

(*a*) 74 + 23

(*b*) 147 + 380

Solution. (*a*) Given the decimal numbers 74 and 23.

$$\begin{array}{rccl}
74 & 0111 & 0100 & \leftarrow \text{BCD code for 74} \\
\underline{+23} & \underline{0010} & \underline{0011} & \leftarrow \text{BCD code for 23} \\
97 & 1001 & 0111 & \leftarrow \text{BCD code for 97}
\end{array}$$

(*b*) Given the decimal numbers 147 and 380.

$$\begin{array}{rcccl}
147 & 0001 & 0100 & 0111 & \leftarrow \text{BCD code for 147} \\
380 & 0011 & 1000 & 0000 & \leftarrow \text{BCD code for 380} \\
+\underline{\quad\;\;} & \underline{\quad\;\;} & \underline{\quad\;\;} & \underline{\quad\;\;} & \\
527 & 0100 & 1100 & 0111 & \leftarrow \text{Invalid code in 2nd digit} \\
 & 1 \leftarrow & +\,0110 & & \leftarrow \text{Add 6 to convert} \\
\hline
 & \underbrace{0101}_{\substack{\uparrow \\ 5}} & \underbrace{0010}_{\substack{\uparrow \\ 2}} & \underbrace{0111}_{\substack{\uparrow \\ 7}} & \leftarrow \text{Convert BCD sum}
\end{array}$$

Notice that the addition of the four-bit codes for 4 and 8 digits results in invalid sum. This is converted by adding 0110. This generates a carry of 1, which is carried over to be added to the BCD sum of the third-position digits.

Example 6-23. *Perform BCD addition*

(256)$_{10}$ + *(683)*$_{10}$ (*Nagpur University, 2004*)

Solution. Given : The decimal number 147 and 380

$$\begin{array}{rcccl}
147 & 0001 & 0100 & 0111 & \leftarrow \text{BCD code for 147} \\
683 & 0110 & 1000 & 0011 & \leftarrow \text{BCD code for 380} \\
+\underline{\quad\;\;} & \underline{\quad\;\;} & \underline{\quad\;\;} & \underline{\quad\;\;} & \\
830 & 0111 & 1100 & 1010 & \leftarrow \text{Invalid code in 1st digit} \\
 & & 1 \;\; + & 0110 & \leftarrow \text{Add 6 to convert} \\
\hline
 & 0111 & 1101 & 0000 & \leftarrow \text{Invalid code in 2nd digit} \\
 & 1 \;\; + & 0110 & & \leftarrow \text{Add 6 to convert} \\
\hline
 & \underbrace{1000}_{8} & \underbrace{0011}_{3} & \underbrace{0000}_{0} &
\end{array}$$

$\therefore$ $(256)_{10} + (683)_{10}$ = 10000011000 BCD **Ans.**

6-25. Hexadecimal Arithmetic

We have already discussed hexadecimal numbers in chapter 2. Recall that it is a method of representing groups of four-bits as a single digit. Hexadecimal numbers are used extensively in computer programming using machine-language and expressing addresses of computer memories. When working in these areas, you will encounter situations where hexadecimal numbers must be added or subtracted.

6-26. Hexadecimal Addition

Addition can be done directly with hexadecimal numbers by remembering that the hexadecimal digits 0 through 9 are equivalent to decimal digits 0 through 9 and that hexadecimal digits A through F are equivalent to decimal numbers 10 through 15. When adding two hexadecimal numbers use the following procedure.

1. Add the two hexadecimal digits in decimal, mentally inserting the decimal equivalent for those digits larger than 9.
2. If the sum is 15 or less, it can be directly expressed as a hexadecimal digit.
3. If the sum is greater than or equal to 16, subtract 16 and carry a 1 to the next digit position.

Example 6-24. *Find the sum of the following points of hexadecimal numbers*:

(*a*) 4F *and* 2D (*b*) 91 B *and* 6F2

(*c*) FFF + 0FF (*d*) A7C5 and 2DA8

Solution.

(*a*)

$$\begin{array}{rcc} & 4 & F \\ + & 2 & D \\ \hline & 7 & C \end{array}$$

Notice that F + D = 15 + 13 = 28, which is 12 with a carry of 1, i.e (28 – 16 =12), 12 is written down as C. Next, 4 + 2 + Carry = 7.

∴ 4F + 2D = 7C **Ans.**

(*b*)

$$\begin{array}{cccc} & 9 & 1 & B \\ + & 6 & F & 2 \\ \hline 1 & 0 & 0 & D \end{array}$$

Notice that B + 2 = 11 + 2 = 13 which is equivalent to hexadecimal number D. Further 1 + F = 1 + 15 = 16, which is 0 with a carry of 1, i.e (16 – 16 = 0). Next 9 + 6 + carry = 16, which is equal to 0 with a carry of 1.

∴ 91B + 6F2 = 100D **Ans.**

(*c*)

$$\begin{array}{cccc} & F & F & F \\ + & 0 & F & F \\ \hline 1 & 0 & E & E \end{array}$$

Notice that F + F = 15 +15 =30 ,which is 14 with a carry of 1, i.e (30 – 16 = 14), 14 is written as E. Next F + F = 30 again E with a carry of 1. Finally F+ carry = 15+1=16 which in hexadecimal is equal to 0 with a carry of 1.

∴ FFF + 0FF = 10EE **Ans.**

(*d*)

$$\begin{array}{ccccc} & A & 7 & C & 5 \\ + & 2 & D & A & 8 \\ \hline & D & 5 & 6 & D \end{array}$$

Notice that 5 + 8 = 13 which is equivalent to hexadecimal D. Next the second position digits C + A = 12 + 10 = 22 which is equal to 6 with a carry of 1, i.e (22 – 16 = 6). Similarly, the third position digits 7 + D + carry = 7 + 13 + 1 = 21 which is equal to 5 with a carry of 1, i.e (21 – 16 = 5). Finally the fourth position digits, A + 2 + carry = 10 + 2 + 1 = 13 which is equivalent to hexadecimal D.

∴ A7C5 + 2DA8= D56D **Ans.**

Example 6-25. *Add the following hexadecimal numbers* $2A3.06_{16}$ *and* $001B.20_{16.}$

(*Nagpur University, Oct/Nov. 1997*)

Solution. Given : The two hexadecimal numbers 2A3.06 and 001B.20.

Since both the numbers are positive, we can add them straightaway. Thus

$$\begin{array}{ccccccc} & & 2 & A & 3 & . & 0 & 6 \\ + & 0 & 0 & 1 & B & . & 2 & 0 \\ \hline & 0 & 2 & B & E & . & 2 & 6 \end{array}$$

$\therefore$ $2A3_{16} + 001B.20_{16} = 02BE.26_{16}$ **Ans.**

6-27. Hexadecimal Subtraction

Since a hexadecimal number can be used to represent a binary number, it can also be used to represent the 2's complement of a binary number. For example the hexadecimal representation of $D6_{16}$ is 11010110. The 2's complement of this number is 00101010 which in decimal is equivalent to $2A_{16}$.

We know that 2's complement allows us to subtract by adding binary numbers. We can also use this method for hexadecimal subtraction.

Consider for example, the subtraction of 21 from 75 as shown below.

	21_{16}		← Hexadecimal number
0010		0001	← Binary number
1101		1111	← Take 2's complement
	DF		← Convert back to Hex

Now add DF_{16} to 75,Thus,

$$\begin{array}{ccc} & 7 & 5 \\ + & D & F \\ \hline \cancel{1} & 5 & 4 \end{array} \longrightarrow \text{Drop carry as in 2's complement addition.}$$

$\therefore$ The difference $75 - 21 = 54_{16}$

Example 6-26. *Perform the following subtractions on the pairs of hexadecimal numbers.*

(*a*) D7 – A8 (*b*) A05C – 24CA

Solution.

(*a*)

A	8	← Hexadecimal number
1010	1000	← Binary number
0101	1000	← Take 2's complement
5	8	← Convert back to Hexadecimal

Now, add 58_{16} to D7. Thus,

$$\begin{array}{ccc} & D & 7 \\ + & 5 & 8 \\ \hline \cancel{1} & 2 & F \end{array} \leftarrow \text{Drop carry as in 2's complement addition.}$$

Note that 7 + 8 = 15 which is hexadecimal F. Similarly D + 5 = 12.Dropping carry we get 2.

∴ The difference D7 – A8 = $2F_{16}$ **Ans.**

(*b*) 2 4 C A ← Hexadecimal number

0010 0100 1100 1010 ← Binary number

1101 1011 0011 0110 ← Take 2's complement

D B 3 6 ← Convert back to Hexadecimal

Now add DB 36_{16} to $A05C_{16}$. Thus,

$$\begin{array}{cccccc} & & A & 0 & 5 & C \\ & + & D & B & 3 & 6 \\ \hline & \cancel{1} & 7 & B & 9 & 2 \end{array}$$

Drop carry as in 2's complement addition.

Note that adding digits at the first position C + 6 = 2 with a carry of 1.Next adding digits at the second position along with a carry, 5 + 3 + carry = 9. Next adding digits at the third-position, 0 + B = B. Finally adding digits at the fourth position, A + D = 17 = 7 + carry. Dropping carry as in 2's complement addition we get 7B92.

∴ The difference, $A05C_{16}$ – 24 CA_{16} = $7B92_{16}$ **Ans.**

6-28. Code Converter

The availability of large different types of codes for the same discrete elements of information result in the use of different codes by different digital systems. Sometimes, it may happen that the use of output of one number system as the input to another number system. Hence it is necessary to design a conversion circuit and insert it between the two systems.

A code converter is a circuit, which accepts the input information in one binary code, converts it and produces an output into another binary code *i.e.*, which makes the two systems compatible, even through each use a different binary code.

To convert from binary code *x* to binary code *y*, the input lines must supply the bit of combination of elements as specified by code *x* and the output lines must generate the corresponding bit combination of code *y*. A combination circuit performs this transformation by means of logic gates. The general block diagram of a code converter is shown in Fig. 6-3.

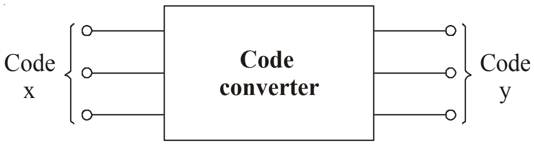


Fig. 6-3.

There are a wide variety of binary codes used in digital systems. Some of these codes are BCD, EX-3, Gray and so on. We will discuss some of the code converter one by one in the following pages.

6-29. Binary-to-BCD Converter

The truth table for binary-to-BCD converter is shown in Table 6-1. To draw the logic diagram we have to simplify the each output functions B_4, B_3, B_2, B_1 and B_0 by using Karnaugh map. We can map these values directly on the four-variable Karnaugh map.

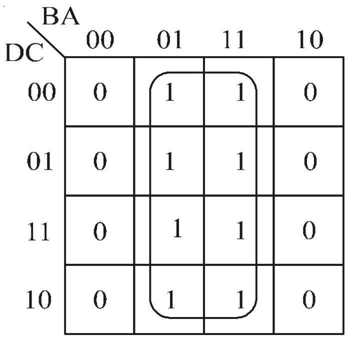
Table 6-1. Binary-to-BCD Code

	Binary code	*BCD code*				
Decimal value	*DCBA*	B_4	B_3	B_2	B_1	B_0
0	0000	0	0	0	0	0
1	0001	0	0	0	0	1
2	0010	0	0	0	1	0
3	0011	0	0	0	1	1
4	0100	0	0	1	0	0
5	0101	0	0	1	0	1
6	0110	0	0	1	1	0
7	0111	0	0	1	1	1
8	1000	0	1	0	0	0
9	1001	0	1	0	0	1
10	1010	1	0	0	0	0
11	1011	1	0	0	0	1
12	1100	1	0	0	1	0
13	1101	1	0	0	1	1
14	1110	1	0	1	0	0
15	1111	1	0	1	0	1

Karnaugh map for B_0

The function B_0 indicates that is output is 1 corresponding to the term indicated 1, 3, 5, 7, 9, 11, 13 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-4. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-4. The output function,

$$B_0 = A$$



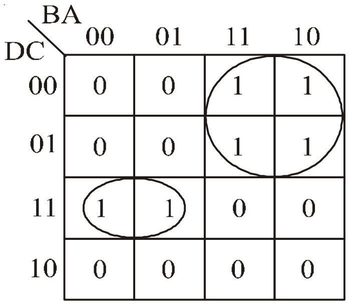
DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

Fig. 6-4.

Karnaugh map for B_1

The function B_1 indicates that is output is 1 corresponding to the term indicated 2, 5, 6, 7, 12 and 13. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-5. In order to simplify the Boolean expression represented o the Karnaugh map, group the 1s as shown in the Fig. 6-5. The output function.

$$B_1 = \overline{D}A + DC\overline{B}$$



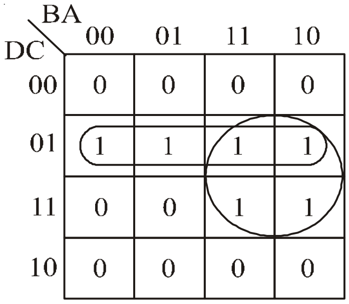
DC \ BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

Fig. 6-5.

Karnaugh map for B_2

The function B_2 indicates that is output is 1 corresponding to the term indicated 4, 5, 6, 7, 14 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-6. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-6. The output function,

$$B_2 = \overline{D}C + CB$$



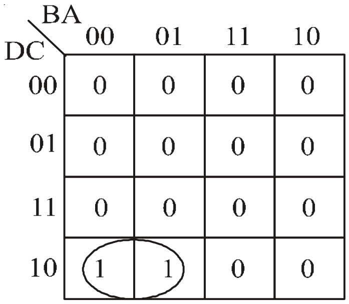
DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

Fig. 6-6.

Karnaugh map for B_3

The function B_2 indicates that is output is 1 corresponding to the term indicated 8 and 9. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-7. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-7. The output function,

$$B_3 = \overline{D}C\overline{B}$$



DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

Fig. 6-7.

Karnaugh map for B_4

The function B_4 indicates that is output is 1 corresponding to the term indicated 10, 11, 12, 13, 14 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-8. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-8. The output function,

$$B_4 = DC + DB = D\,(D + B)$$

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Fig. 6-8.

The logic diagram for the code converter of Binary-to-BCD Converter is obtained by the implementation of the output function B_4, B_3, B_2, B_1 *and* B_0 as a logic circuit. It is shown in Fig. 6-9 it requires four variables A, B, C and D.

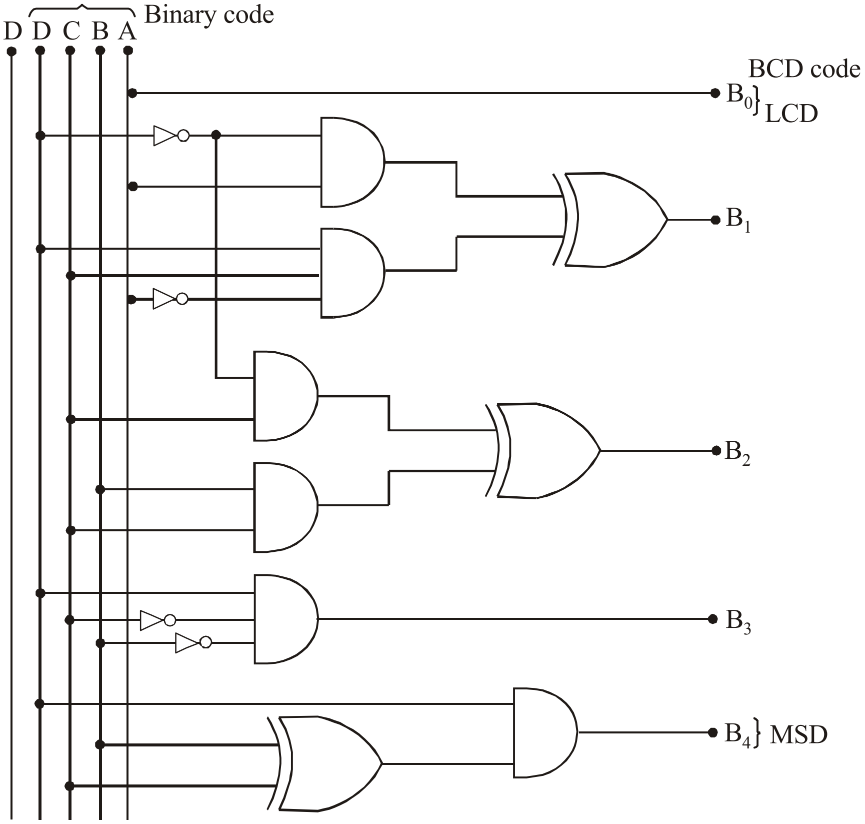


Fig. 6-9. Binary-to-BCD Converter

6-30. BCD-to-Binary Converter

For converting any BCD number into its equivalent binary number, we needs five inputs and five outputs as shown in Fig. 6-10.

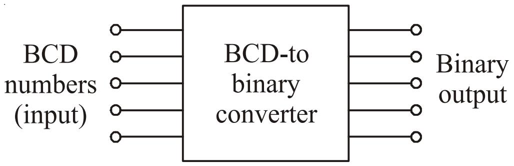


Fig. 6-10. Block diagram of BCD-to-binary converter

The truth table for BCD-to-Binary converter is shown in Table 6-2. To draw the logic diagram we have to simplify the each output functions *A, B, C, D and E* by using Karnaugh map. We can map these values directly on the five-variable Karnaugh map.

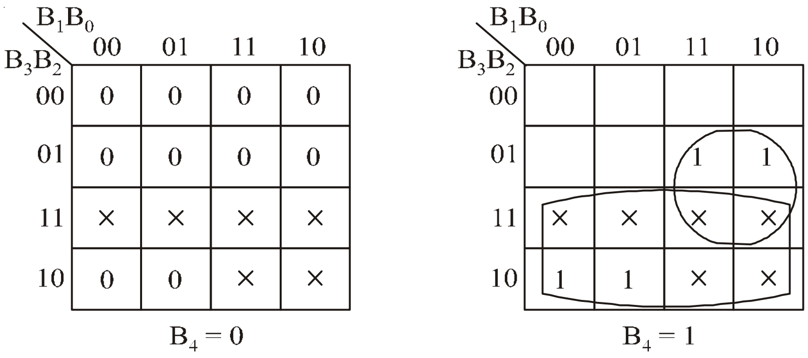
Table 6-2. BCD-to-Binary Code

Decimal number	***Inputs***					***Outputs***				
	B_4	B_3	B_2	B_1	B_0	E	D	C	B	A
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	1	0	0	0	0	0	1	0	1	0
11	1	0	0	0	1	0	1	0	1	1
12	1	0	0	1	0	0	1	1	0	0
13	1	0	0	1	1	0	1	1	0	1
14	1	0	1	0	0	0	1	1	1	0
15	1	0	1	0	1	0	1	1	1	1
16	1	0	1	1	0	1	0	0	0	0
17	1	0	1	1	1	1	0	0	0	1
18	1	1	0	0	0	1	0	0	1	0
19	1	1	0	0	1	1	0	0	1	1

Karnaugh map for *E*

The five-variable Karnaugh map for the function E is shown in Fig. 6-11. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for E,

$$E = B_4B_3 + B_4B_2B_0$$



B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	×	×	×	×
10	0	0	×	×

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00				
01			1	1
11	×	×	×	×
10	1	1	×	×

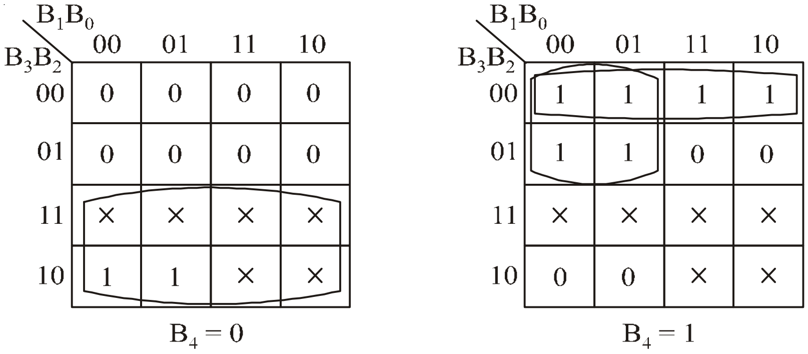
$B_4 = 1$

Fig. 6-11.

Karnaugh map for *D*

The five-variable Karnaugh map for the function D is shown in Fig. 6-12. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for D,

$$D = B_3\overline{B_4} + \overline{B_3}\,\overline{B_2}B_4 + B_4\overline{B_3}B_1$$



B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	×	×	×	×
10	1	1	×	×

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	×	×	×	×
10	0	0	×	×

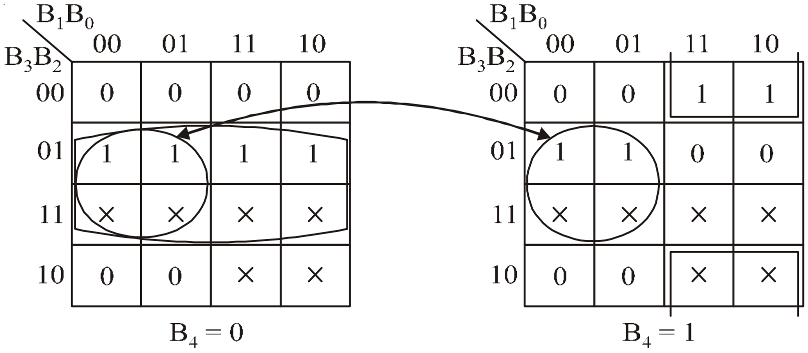
$B_4 = 1$

Fig. 6-12.

Karnaugh map for *C*

The five-variable Karnaugh map for the function C is shown in Fig. 6-13. To simplify the Boolean expression reprsented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for C,

$$C = \overline{B_4}B_2 + B_2\overline{B_1} + B_4\overline{B_2}B_1$$



B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	×	×	×	×
10	0	0	×	×

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	×	×	×	×
10	0	0	×	×

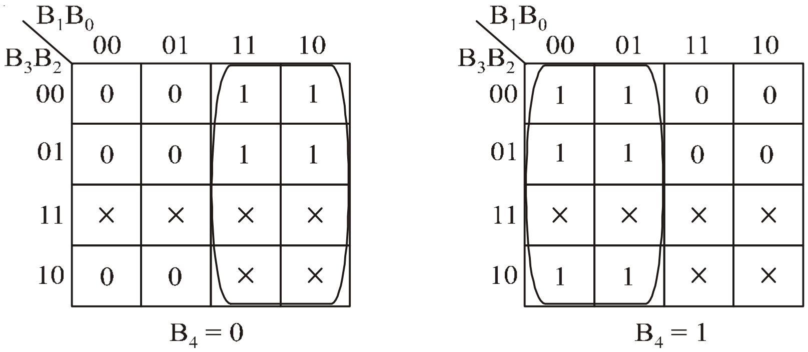
$B_4 = 1$

Fig. 6-13.

Karnaugh map for *B*

The five-variable Karnaugh map for the function B is shown in Fig. 6-14. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for B,

$$B = B_1 \oplus B_4$$



B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	×	×	×	×
10	0	0	×	×

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	×	×	×	×
10	1	1	×	×

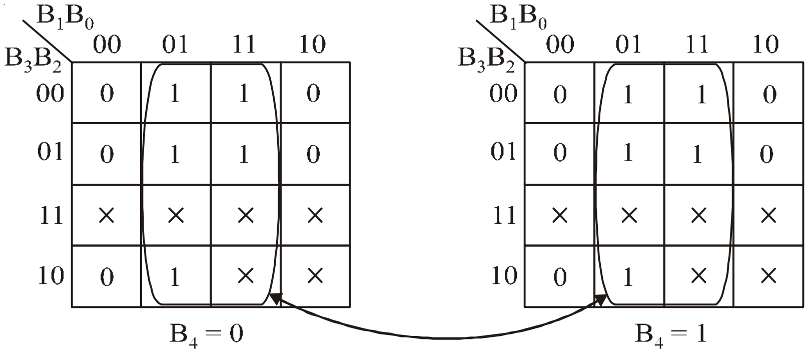
$B_4 = 1$

Fig. 6-14.

Karnaugh map for *A*

The five-variable Karnaugh map for the function A is shown in Fig. 6-15. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for A,

$$A = B_0$$



B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	×	×	×	×
10	0	1	×	×

$B_4 = 0$

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	×	×	×	×
10	0	1	×	×

$B_4 = 1$

Fig. 6-15.

The logic diagram for the code converter of BCD-to-Binary Converter is obtained by the implementation of the output function *E*, *D*, *C*, *B and A* as a logic circuit. It is shown in Fig. 6-16 it requires five variables B_4, B_3, B_2, B_1*and* B_0.

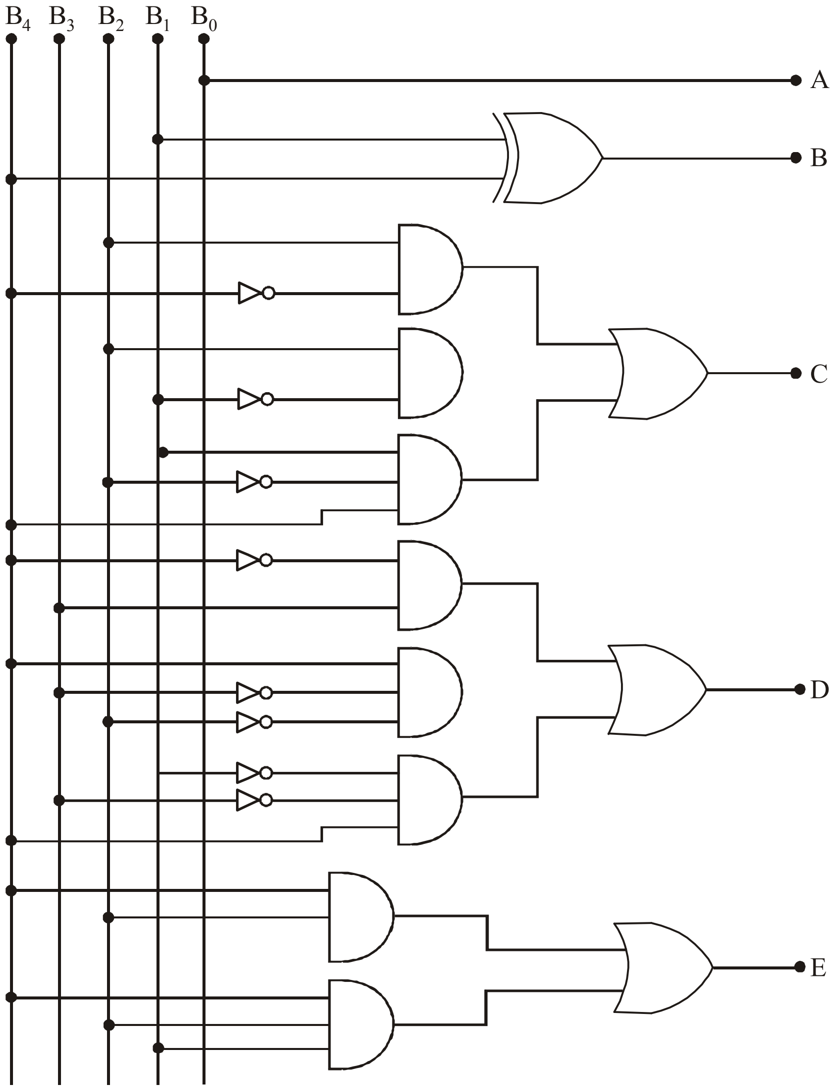


Fig. 6-16. BCD-to-Binary Converter.

6-31. Binary-to-Gray Code Converter

The main advantage of gray code is that, only one bit in the numerical representation changes between the successive numbers. Table 6-3 shows the truth table for Binary-to-Gray code conversion. The conversion circuit must contain four-inputs and four-outputs as shown in Fig. 6-17.

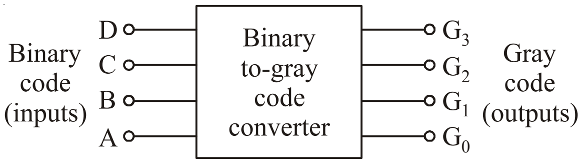


Fig. 6-17. Block Diagram for Binary-to-Gray code converter

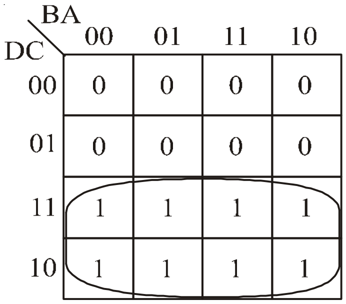
Table 6-3. Binary-to-Gray Code

Decimal	*Binary code*	*Gray code*			
	DCBA	G_3	G_2	G_1	G_0
0	0000	0	0	0	0
1	0001	0	0	0	1
2	0010	0	0	1	1
3	0011	0	0	1	0
4	0100	0	1	1	0
5	0101	0	1	1	1
6	0110	0	1	0	1
7	0111	0	1	0	0
8	1000	1	1	0	0
9	1001	1	1	0	1
10	1010	1	1	1	1
11	1011	1	1	1	0
12	1100	1	0	1	0
13	1101	1	0	1	1
14	1110	1	0	0	1
15	1111	1	0	0	0

Karnaugh map for G_3

The function G_3 indicates that is output is 1 corresponding to the term indicated 8, 9, 10, 11, 12, 13, 14 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-18. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-18. Thus output function,

$$G_3 = D$$



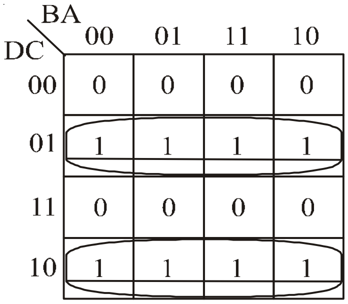
DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Fig. 6-18.

Karnaugh map for G_2

The function G_2 indicates that is output is 1 corresponding to the term indicated 4, 5, 6, 7, 8, 9, 10 and 11. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-19. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-19. Thus output function,

$$G_2 = \overline{D}C + D\overline{C} = D \oplus C$$



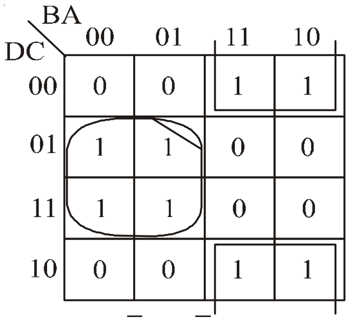
DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Fig. 6-19.

Karnaugh map for G_1

The function G_1 indicates that is output is 1 corresponding to the term indicated 2, 3, 4, 5, 10, 11, 12 and 13. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-20. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-20. Thus output function,

$$G_1 = C\overline{B} + \overline{C}B = C \oplus B$$



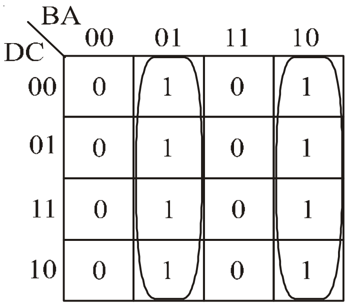
DC \ BA	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

Fig. 6-20.

Karnaugh map for G_0

The function G_0 indicates that is output is 1 corresponding to the term indicated 1, 2, 5, 6, 9, 10, 13 and 14. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-21. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-21. Thus output function,

$$G_0 = \overline{B}A + A\overline{B} = B \oplus A$$



DC \ BA	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

Fig. 6-21.

The logic diagram for the code converter of Binary-to-Gray Converter is obtained by the implementation of the output function G_3, G_2, G_1 and G_0 as a logic circuit. It is shown in Fig. 6-22 it requires four variable D, C, B *and* A.

$$\begin{aligned} G_3 &= D \\ G_2 &= \overline{D}C + D\overline{C} = D \oplus C \\ G_1 &= C\overline{B} + \overline{C}B = C \oplus B \\ G_0 &= \overline{B}A + A\overline{B} = B \oplus A \end{aligned}$$

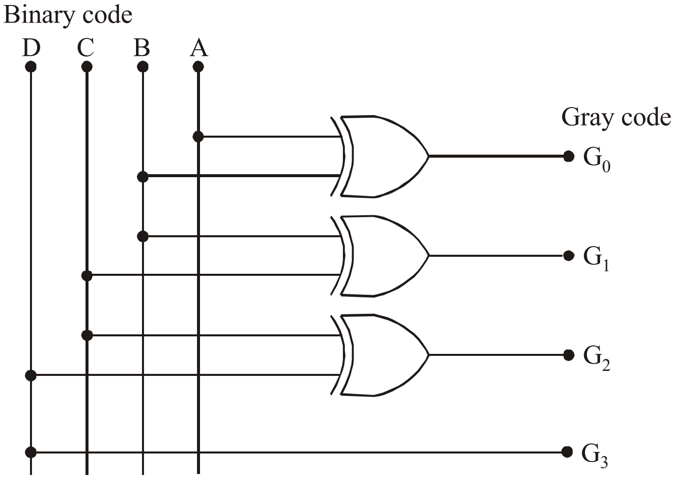


Fig. 6-22.

6-32. Gray-to-Binary Code Converter

Table 6-4 shows the truth table for Gray-to-Binary code conversion. To draw the logic diagram we have to simplify the each output functions A, B, C, *and* D by using Karnaugh map. We can map these values directly on the four-variable Karnaugh map.

Table 6-4. Gray-to-Binary Code

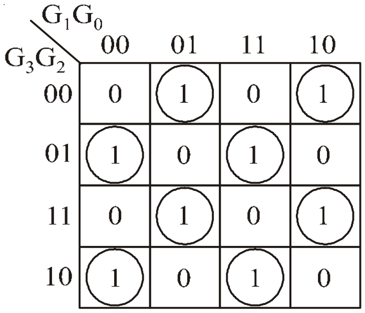
Decimal	***Gray code***				***Binary code***
	G_3	G_2	G_1	G_0	***DCBA***
0	0	0	0	0	0000
1	0	0	0	1	0001
3	0	0	1	1	0010
2	0	0	1	0	0011
6	0	1	1	0	0100
7	0	1	1	1	0101
5	0	1	0	1	0110
4	0	1	0	0	0111
12	1	1	0	0	1000
13	1	1	0	1	1001
15	1	1	1	1	1010
14	1	1	1	0	1011
10	1	0	1	0	1100
11	1	0	1	1	1101
9	1	0	0	1	1110
8	1	0	0	0	1111

Karnaugh map for *A*

The function *A* indicates that is output is 1 corresponding to the term indicated 1, 2, 4, 7, 8, 11, 13 and 14. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-23. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-23. Thus output function,

$$\begin{aligned} A = {} & \overline{G}_3\overline{G}_2\overline{G}_1G_0 + \overline{G}_3\overline{G}_2G_1\overline{G}_0 + \overline{G}_3\,G_2\overline{G}_1\overline{G}_0 + \overline{G}_3\,G_2G_1G_0 + G_3G_2\overline{G}_1G_0 \\ & + G_3G_2G_1\overline{G}_0 + G_3\overline{G}_2\overline{G}_1\overline{G}_0 + G_3\overline{G}_2G_1G_0 \end{aligned}$$

$$A = (G_0 \oplus G_1) \oplus (G_2 \oplus G_3)$$



G_3G_2 \ G_1G_0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

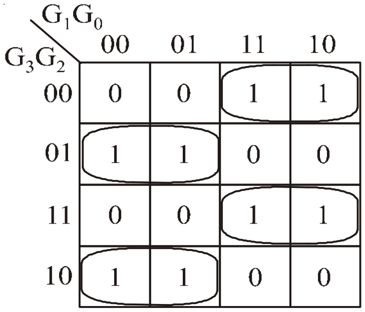
Fig. 6-23.

Karnaugh map for *B*

The function *B* indicates that is output is 1 corresponding to the term indicated 2, 3, 4, 5, 8, 9, 14 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-24. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-24. Thus output function,

$$B = \overline{G}_3\overline{G}_2G_1 + \overline{G}_3\,G_2\overline{G}_1 + G_3\,G_2\,G_1 + G_3\overline{G}_2\overline{G}_1$$

$$B = G_1 \oplus G_2 \oplus G_3$$



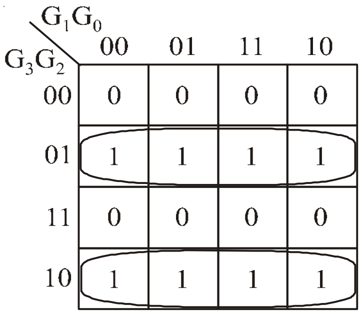
G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

Fig. 6-24.

Karnaugh map for *C*

The function *C* indicates that is output is 1 corresponding to the term indicated 4, 5, 6, 7, 8, 9, 10 and 11. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-25. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-25. Thus output function,

$$C = \overline{G}_3G_2 + G_3\overline{G}_2 = G_2 \oplus G_3$$



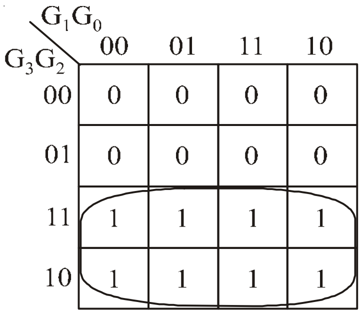
G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Fig. 6-25.

Karnaugh map for *D*

The function *D* indicates that is output is 1 corresponding to the term indicated 8, 9, 10, 11, 12, 13, 14 and 15. We can map these values directly on to the four-variable Karnaugh map as shown in Fig. 6-26. In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in the Fig. 6-26. Thus output function,

$$D = G_3$$



G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Fig. 6-26.

The logic diagram for the code converter of Gray-to-Binary Converter is obtained by the implementation of the output function *D*, *C*, *B and A* as a logic circuit. It is shown in Fig. 6-27 it requires four variable G_3, G_2, G_1 *and* G_0.

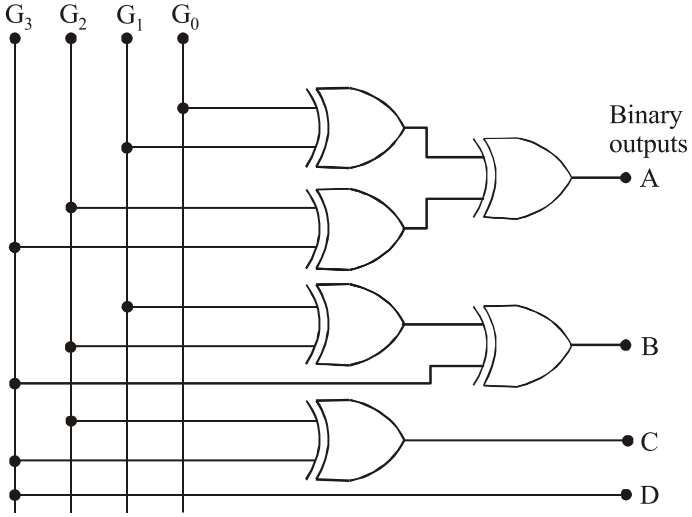


Fig. 6-27.

6-33. BCD-to-Excess-3 Code Converter

The Excess-3 code can be obtained from the BCD code by adding 3 to each coded number. The code converter contains four inputs and four outputs as shown in Fig. 6-28.

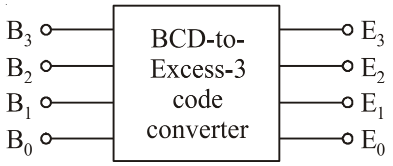


Fig. 6-28.

The truth table for BCD-to-Excess-3 code is shown in Table 6-5.

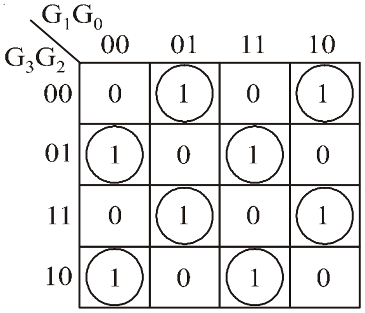
Table 6-5. BCD-to-Excess-3 code

Decimal number	***BCD number code***				***Excess-3 code***			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Karnaugh map for E_3

The four-variable Karnaugh map for the function E_3 is shown in Fig. 6-29. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for E_3,

$$E_3 = B_3 + B_2 B_0 + B_2 B_1$$



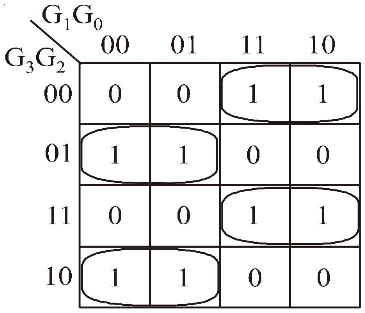
$G_3G_2 \backslash G_1G_0$	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Fig. 6-29.

Karnaugh map for E_2

The four-variable Karnaugh map for the function E_2 is shown in Fig. 6-30. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for E_2,

$$E_2 = \bar{B}_2 B_0 + \bar{B}_2 B_1 + B_2 \bar{B}_1 B_0$$



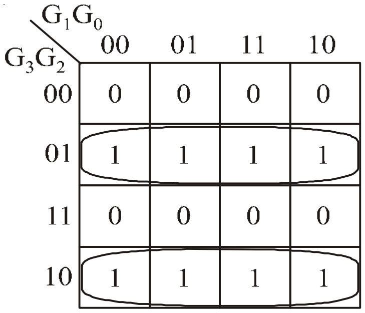
G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

Fig. 6-30.

Karnaugh map for E_1

The four-variable Karnaugh map for the function E_1 is shown in Fig. 6-31. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for E_1,

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



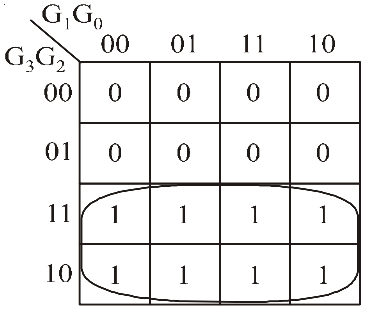
G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Fig. 6-31.

Karnaugh map for E_0

The four-variable Karnaugh map for the function E_0 is shown in Fig. 6-32. To simplify the Boolean expression represented on the Karnaugh map, group the 1s (Xs can be treated as 1s). Thus the output function for E_0,

$$E_0 = \bar{B}_0$$



G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Fig. 6-32.

The logic diagram for the code converter of BCD-to-Excess-3 code converter, it's obtained by the implementation of the output function E_3, E_2, E_1 *and* E_0 as a logic circuit. It is shown in Fig. 6-33 it requires four variables B_3, B_2, B_1 *and* B_0.

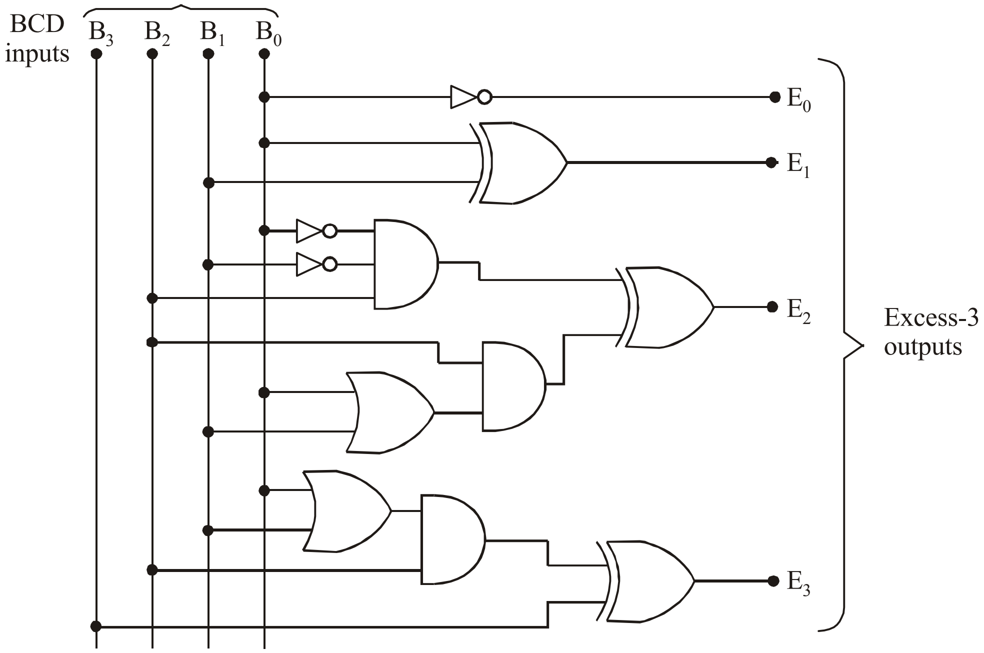


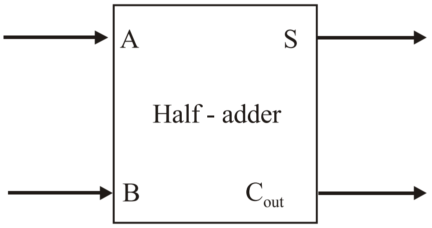
Fig. 6-33.

6-34. Arithmetic Circuits

As a matter of fact, all the arithmetic operations and procedures discussed in the previous articles can be implemented using adders formed from basic logic gates. For a large number of digits, we can use medium-scale integration (MSI) circuits. The MSI circuits have several adders within a single integrated package.

6-35. The Half Adder

Figure 6-34 (a). shows the logic symbol of a half-adder. As seen from this figure, we find that the half-adder accepts two binary digits on its inputs and produces two digits on its outputs: a sum bit (S) and a carry bit (C_{out}). Fig 6-34 (b) shows the truth table for the half-adder.



(*a*) logic symbol

Inputs		***Outputs***	
A	***B***	***S***	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(*b*) truth table

Fig. 6-34. A half-adder

The half- adder follows the basic rules for binary addition:

$$0+1 = 0$$
$$0+1 = 1$$
$$1+0 = 1$$
$$1+1 = 10 \text{ or } 0 \text{ with a carry of } 1$$

The Boolean expression for the sum output (S) can be expressed by the equation.

$$S = A\overline{B} + \overline{A}B$$
$$= A \oplus B \qquad ...(1)$$

and the Boolean expression for the carry output,

$$C_{ont} = AB \qquad ...(2)$$

The equations (1) and (2) can be implemented by a logic circuit shown in Fig 6-35 . As seen from this figure, the sum output (s) is obtained from an excluse-OR gate while the carry output (C_{out}) is the output of a two-input AND gate.

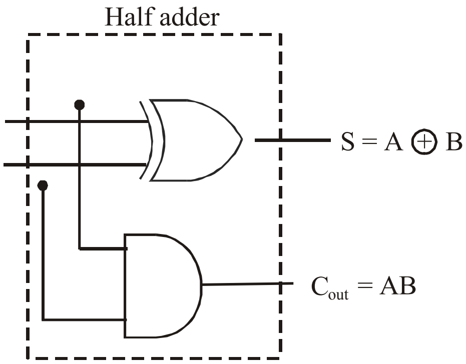
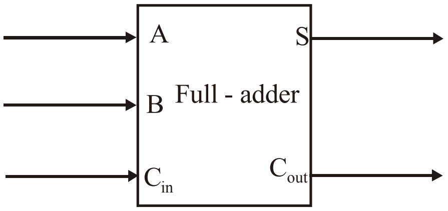


Fig. 6-35. Logic implementation of a half-adder

6-36. The Full – Adder

Fig 6-36 (a) shows the logic symbol of a full-adder. As seen from this figure, we find that the full-adder accepts three inputs-two input bits A and B and a carry input (C_{in}). It has two outputs: (1) a sum output (s) and a carry- output (C_{out}). Fig 6-36 (b) shows the truth table for the full-adder.



(*a*) logic symbol

Inputs			*Outputs*	
A	***B***	C_{in}	***S***	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(*b*) truth table

Fig. 6-36. Full-adder

The full-adder also follows the same basic rules for binary addition as the half adder, Thus,

$$
\begin{array}{lcll}
0+0+0 & = & 0 & \text{with carry} \quad 0 \\
0+0+1 & = & 1 & \text{with carry} \quad 0 \\
0+1+0 & = & 1 & \text{with carry} \quad 0 \\
0+1+1 & = & 0 & \text{with carry} \quad 1 \\
1+0+0 & = & 1 & \text{with carry} \quad 0 \\
1+0+1 & = & 0 & \text{with carry} \quad 1 \\
1+1+0 & = & 0 & \text{with carry} \quad 1 \\
1+1+1 & = & 1 & \text{with carry} \quad 1
\end{array}
$$

The Boolean expression for the sum output (S) can be obtained from the truth table (shown in Fig. 6-36 (b)) by summing the terms for which *S*=1.Thus, the sum,

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} \qquad \text{...(3)}$$

Let us now try to simplify this expression by factoring. Unfortunately none of the terms in the expression has two variables in common with any of the other terms. However, $\bar{A}$ can be factored from the first two terms and A can be factored from the last two terms. Thus,

$$S = \bar{A}\,(\bar{B}C_{in} + B\bar{C}_{\cdot in}) + A\,(\bar{B}\bar{C}_{in} + BC_{in}) \qquad \text{... (4)}$$

Notice that the first terms within parentheses in equation (4) can be recognized as the exclusive-OR combination of B and C_{in}, which can be written as $B \oplus C_{in}$. The second term within the parentheses in equation (4) can be recognized as the exclusive -NOR of B and C_{in}, which can be written as $\overline{B \oplus C_{in}}$. Thus the expression for S can be written as,

$$S = \bar{A}\,(B \oplus C_{in}) + A\,(\overline{B \oplus C_{in}}) \qquad \text{...(5)}$$

Let $X = B \oplus C_{in}$, then equation (5) can be rewritten as,

$$\begin{aligned} S &= \bar{A}.X + A.\bar{X} \\ &= A \oplus X \qquad \text{...(6)} \end{aligned}$$

Replacing X with $B + C_{in}$, equation (6) can be written as,

$$\begin{aligned} S &= A \oplus (B \oplus C_{in}) \qquad \text{...(7)} \\ &= A \oplus B \oplus C_{in} \end{aligned}$$

Next consider the carry output, (C_{out}) in the truth table of Fig 6-5 (b). The boolean expression for the carry output can be obtained from the truth table by summing the terms for which $C_{out} = 1$. Thus the carry output,

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

This expression can be simplified by factoring. In order to do so let us add the ABC_{in} term 3 times since it has common factors with each of the other terms. Hence the expression for carry output,

$$\begin{aligned} C_{out} &= \left(\bar{A}BC_{in} + ABC_{in}\right) + \left(A\bar{B}C_{in} + ABC_{in}\right) + \left(AB\bar{C}_{in} + ABC_{in}\right) \\ &= BC_{in}\,(\bar{A}+A) + AC_{in}\,(\bar{B}+B) + AB\,(\bar{C}_{in} + C_{in}) \end{aligned}$$

Since $\bar{A} + A = \bar{B} + B = \bar{C}_{in} + C_{in} = 1$, therefore the above expression can be simplified to,

$$C_{ont} = BC_{in} + AC_{in} + AB \qquad \text{... (8)}$$

The equations (7) and (8) can be implemented by using logic gates. From equation (7) we find that to implement to full-adder sum output function, two 2-input exclusive –OR gates can be used. The first exclusive-OR gate generates the term A ⊕ B, and the second has its inputs the output of the first exclusive –OR gate and the input carry as shown in the Fig 6-37. Similarly from equation (8) we find that to implement the full- adder carry output function, three 2-input AND gates followed by a 3-input OR gate can be used. The complete circuit of a full adder is shown in Fig 6-37.

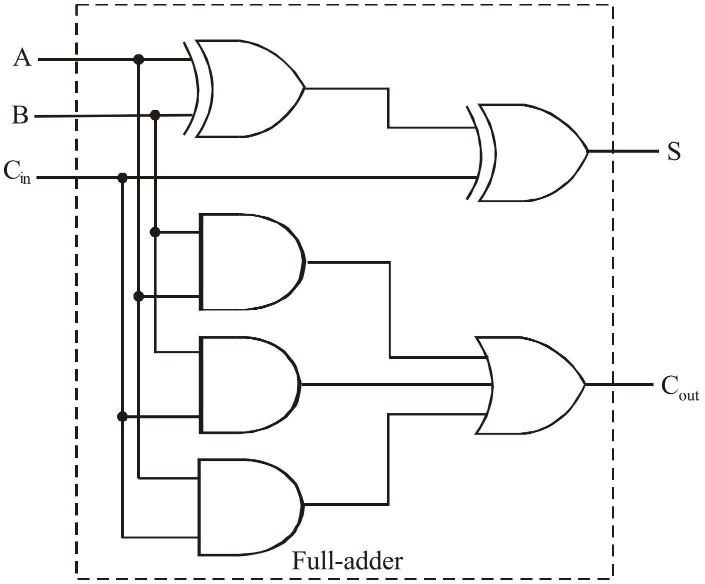


Fig 6-37. Complete logic circuitry for a full-adder.

Notes.

1. We simplified the expressions for S and C_{out} using algebraic methods. The K-map can also be used for this purpose. We leave it as an exercise for the students to try it out.
2. Several other implementations can be used to produce the same expressions for S and C_{out}.

6-37. Implementation of Full-adder using Two Half-adders

We have already discussed in the previous articles about the implementation of half-adder and a full-adder circuit using logic gates. Let us now study how a full adder can be implemented using half-adder circuit as a building block.

We know that the sum output of a half-adder with two-inputs (A and B) as shown in Fig 6-38 is given by the expression.

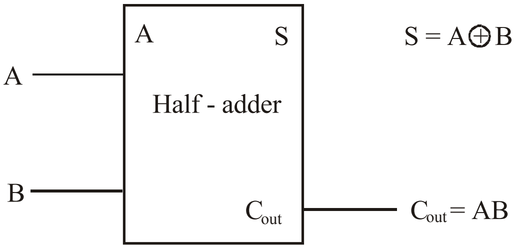


Fig. 6-38. Logic symbol of a Half-adder

$$S = A \oplus B$$

and the carry output,

$$C_{out} = AB$$

For a full-adder, the sum output,

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

and the carry output,

$$\begin{aligned} C_{out} &= \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C}in + ABC_{in} \\ &= (\overline{A}B + A\overline{B})\,C_{in} + AB\,(\overline{C}_{in} + C_{in}) \\ &= (A \oplus B)\,C_{in} + AB\,(1) \\ &= (A \oplus B)\,C_{in} + AB \end{aligned}$$

The above equation shows that the carry output function consists of two terms. The first term is ANDing of C_{in} and an exclusive-OR of A and B inputs. While the second term is an ANDing of two inputs A and B.

Now Fig 6-39 shows how the sum and the carry output functions of a full-adder can be obtained by using two half-adders.

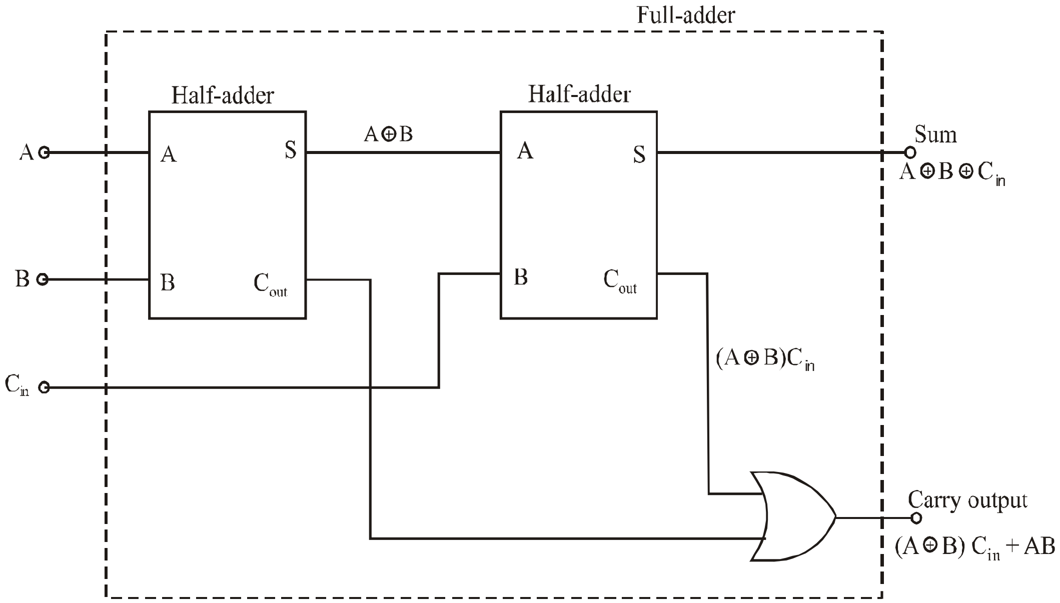


Fig 6-39. Implementation of a full-adder using two half-adders.

6-38. Parallel Binary Adders

As a matter of fact, a single full adder is capable of adding two 1-bit numbers and an input carry. If we need to add binary numbers with more than one bit, we require additional full-adders. It may be carefully noted that when one binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left. This process is illustrated below with a 2-bit numbers 11 and 01.

Carry bit from the right column

$$\begin{array}{rccc} & & 1 & \\ & & 1 & 1 \\ + & & 0 & 1 \\ \hline & 1 & 0 & 0 \end{array}$$

In this case, the carry bit from second column becomes a sum bit.

In order to add two binary numbers, a full adder is required for each bit in the number. Thus for two-bit numbers we need two full-adders. Similarly for the addition of four-bit numbers, we need four full-addres and so on.

Fig 6-40 shows a block diagram of a basic 2-bit parallel adder using two full- adders. Notice that the carry output of each full-adder is connected to the carry input of the next higher-order full adder. Further notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be connected to ground (0). This is because of the reason that there is no carry input to the least significant position.

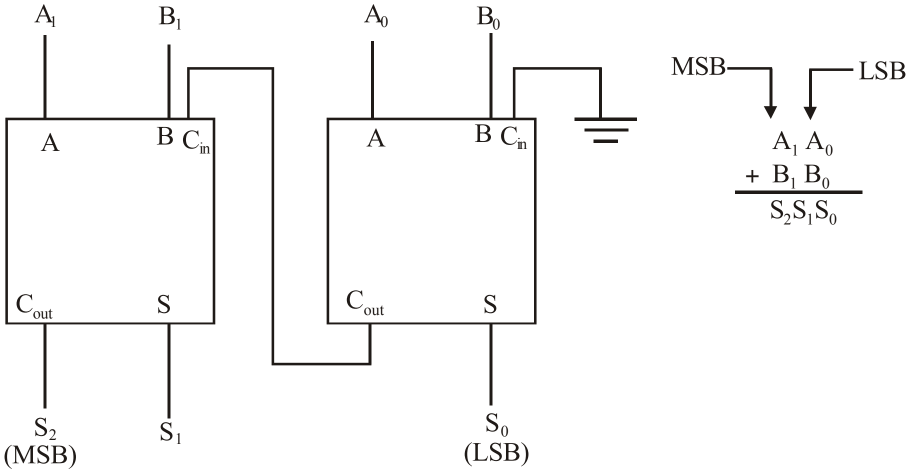


Fig. 6-40. Block diagram of a basic 2-bit parallel adder.

6-39. Four-bit Parallel Adder

Fig 6-41 shows the block diagram of a four-bit parallel adder. As seen, it consists of four full-adders connected together. Notice that the carry output of each adder is connected to the

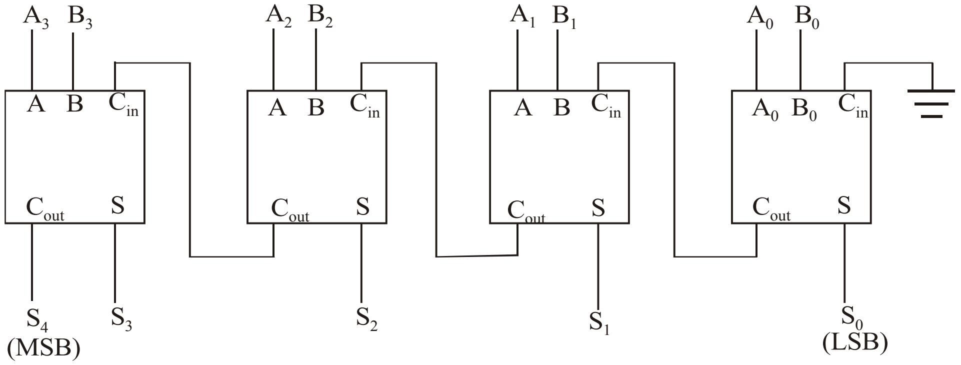


Fig. 6-41. Block diagram of a four-bit parallel adder.

carry input of next higher stage. These are called internal carries. Also notice that the least-significant bits (LSBs) of the two binary numbers (being added) go into the right-most full-adder whereas the higher order bits are applied as shown to the successive higher-order adders. The most-significant bits (MSBs) of the two numbers (being added) are applied to the left-most full-adder.

Fig – 6-42 shows the logic symbol for a 4-bit parallel adder. Notice that the input carry is labelled as C_0. and the output carry at C_4. This is done to keep in line with most manufacturer's datasheets.

In terms of the method used to handle carries in a parallel adder, there are two types :

1. The ripple carry adder and
2. The carry look-ahead adder

A ripple carry adder is one in which the carry output of each full-adder is connected to the carry input of the next higher order stage. The sum and the output carry of any stage cannot be produced until the input carry occurs. This process causes a time delay in the addition.

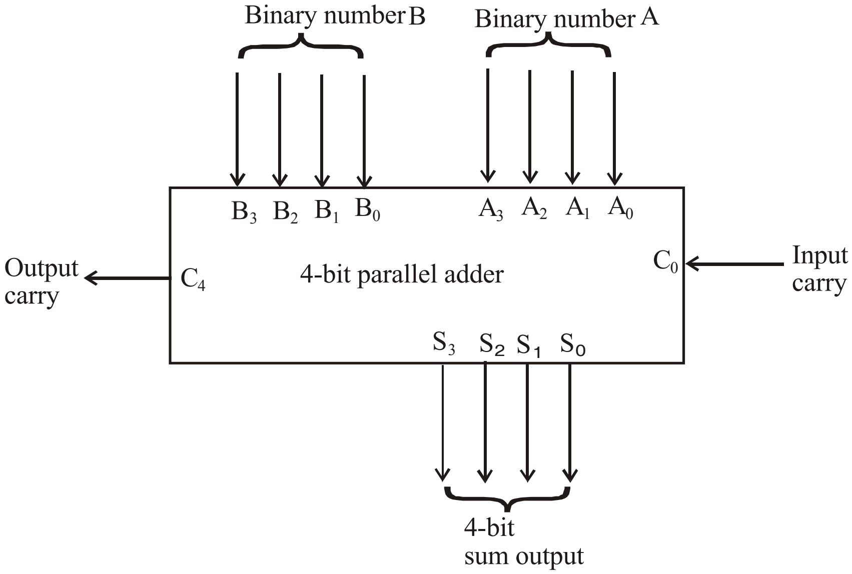


Fig. 6-42. Logic symbol of a 4-bit parallel adder.

The ripple carry delay can be eliminated by making use of a method called look-ahead carry addition. The look-ahead carry adder anticipates the output carry of each stage and based on the input bits of each stage, produces the output carry by either carry generation or carry propagation. Carry generated occurs when an output carry is produced (or generated) internally by the full- adder. A carry is generated only when both input bits are 1s. On the other hand, carry propagation occurs when the input carry is rippled to become the output carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s.

6-40. Carry Look-Ahead Adder

A carry look-ahead adder is a high speed adder which follows a special strategy for quick carry generation at each stage without waiting for the carries of the previous stages. Let us consider a full-adder circuit in block diagram from and its EX-OR realization shown in Fig. 6-43.

$$P_i = A_i \oplus Bi$$
$$G_i = A_i B_i$$

The output sum and carry can respectively expressed as

$$S_i = P_i \oplus C_i = A_i \oplus B_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a carry generate, and it produce a carry of 1 when both A_i and B_i are 1. P_i is called carry propagate, because it determines whether a carry into stage i will propagate into state $i + 1$.

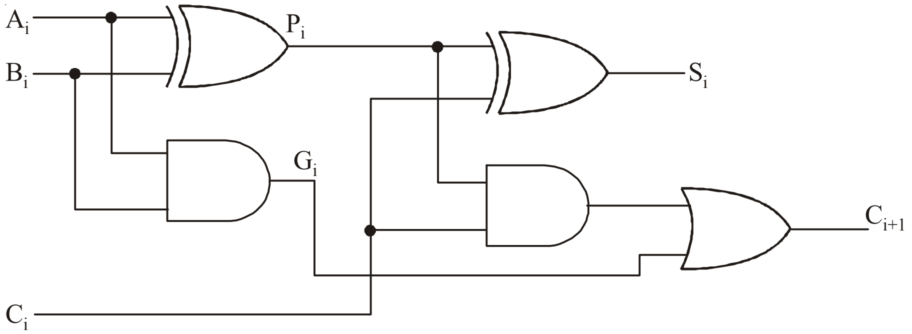


Fig. 6-43. Full Adder

Now we write Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations

$$C_0 = \text{input carry}$$

For stage 1:

$$C_1 = G_0 + P_0C_0$$

For stage 2:

$$C_2 = G_1 + P_1C_1$$

Subtitling the value of $C_1 = G_0 + P_0C_0$ we get

$$C_2 = G_1 + P_1\ (G_0 + P_0C_0)$$
$$C_2 = G_1 + (P_1G_0 + P_1P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

For stage 3:

$$C_3 = G_2 + P_2C_2$$

Subtitling the value of $C_2 = G_1 + P_1G_0 + P_1P_0C_0$ we get

$$C_3 = G_2 + P_2\ (G_1 + P_1G_0 + P_1P_0C_0)$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

The G variable can be generated directly from A and B inputs using AND gates, the P variable are obtained directly from A and B inputs using EX-OR gates. C_i is the carry input. The Boolean function for each output carry is expressed in sum-of-products form, each function can be implemented with one level of AND gates followed by an OR gate or by a two-level NAND gate.The Boolean functions of three stages are implements in the carry lookahead generator shown in Fig. 6-44. The circuit can in less time because for C_3, it does not wait for C_2 and C_1 to propagate.

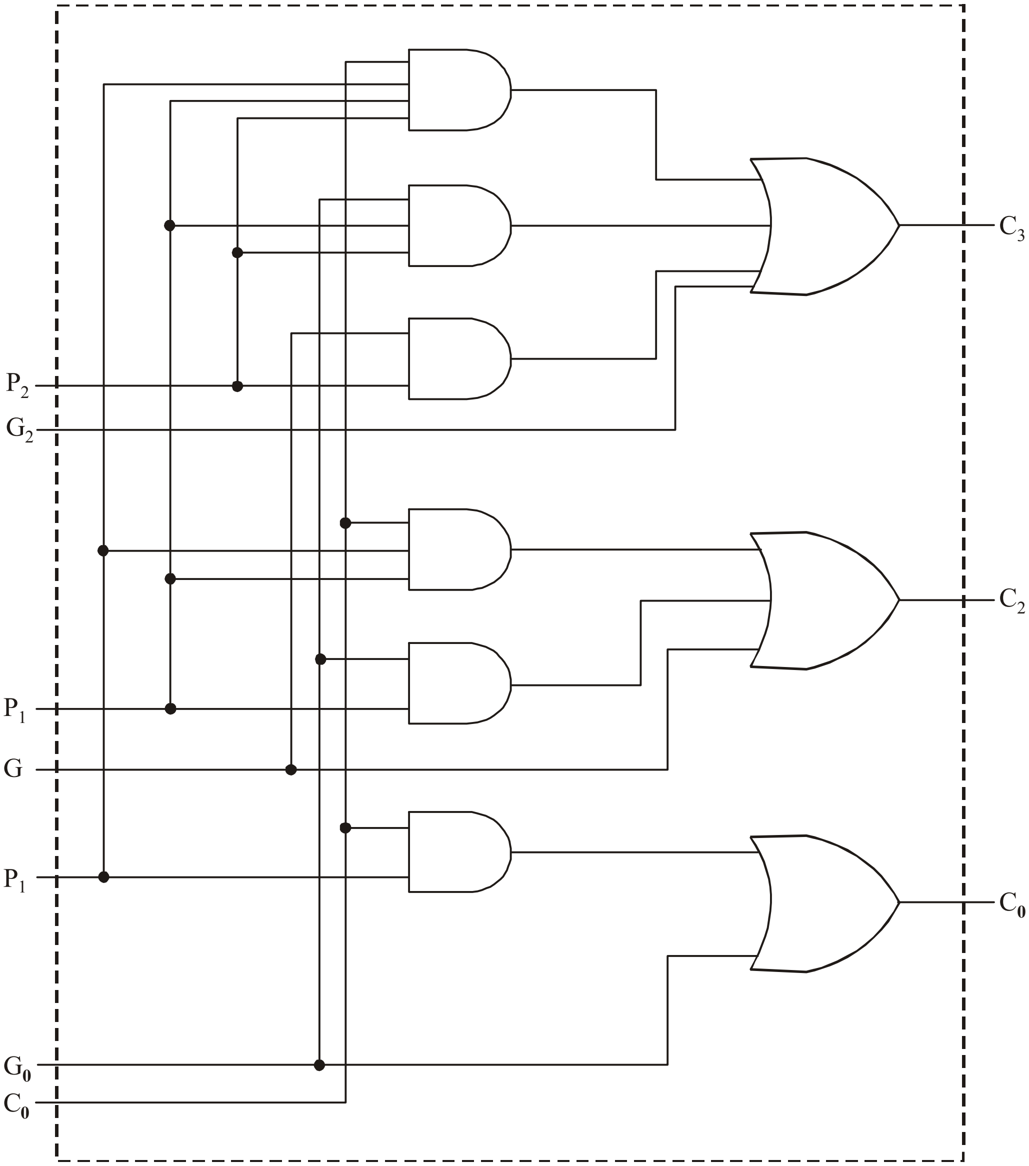


Fig. 6-44. Carry Lookhead Generator.

A 4-bit adder with a carry lookhead is shown in Fig. 6-45. Each output sum requires two EX-OR gates. The output of the first EX-OR gate generates P_i variable, and the AND gate generates the G_i variable. The carries are propagated through the carry lookhead generator and applied as inputs to the second EX-OR gate. All output carries are generated after delay through to levels of gates. The outputs S_1 through S_3 have equal propagation delay times. The two-level circuit for the output carry C_4 is not shown.

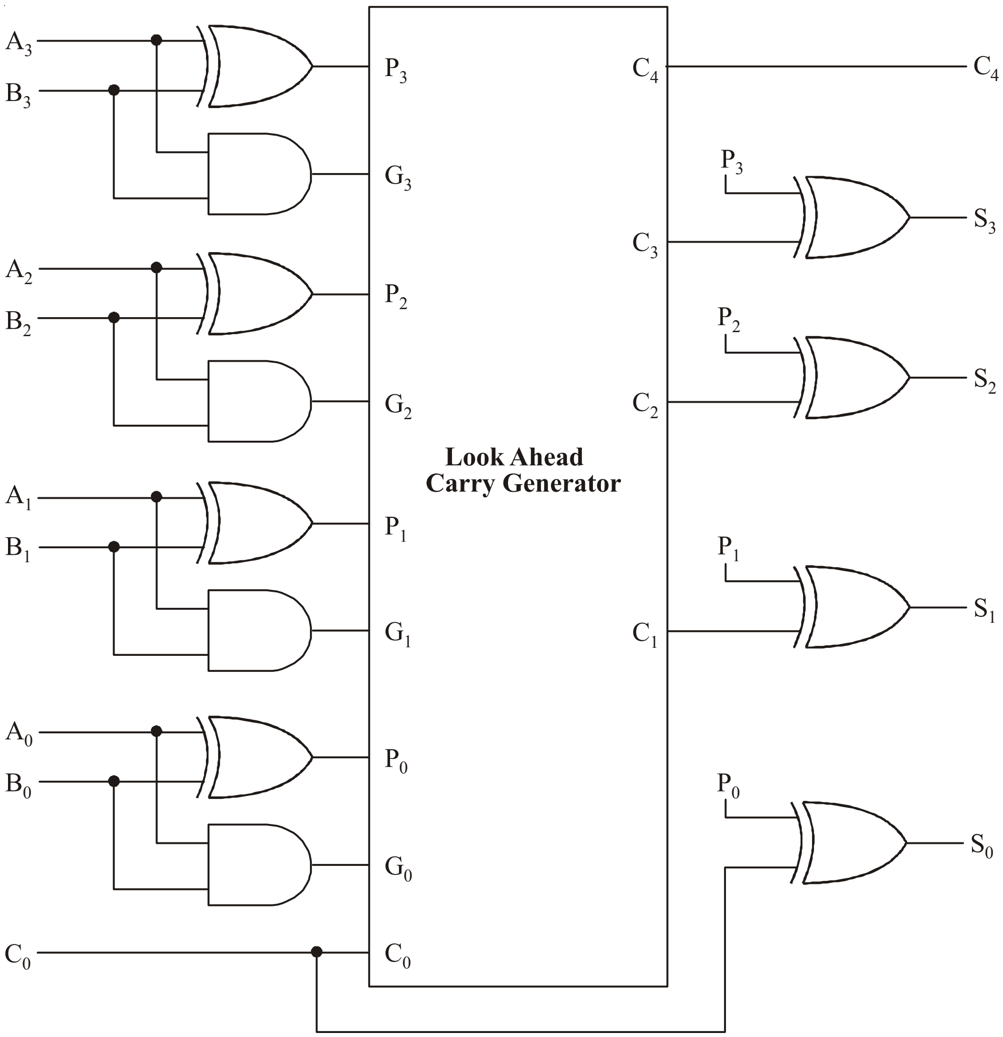


Fig. 6-45. 4-bit Adder Carry Lookhead.

6-41. Integrated Circuits Parallel Adders

There are several parallel adders that are available as ICs (integrated-circuits) in the market. Table 6-1 shows the most commonly used parallel adders. All these adders contain four interconnected full-adders and the look-ahead carry circuitry needed for high speed operation.

Table 6-1. IC parallel Adders

Device	*Family*	*Description*
7483A	TTL	4-bit binary full-adder
74LS83A	LSTTL	4-bit binary full-adder
74283	TTL	4-bit binary full-adder
74LS283	LSTTL	4-bit binary full-adder
74HC283	CMOS	4-bit binary full-adder

The 74283s are identical to 7483s except that they have V_{CC} and ground pins 16 and 8 respectively. In all new integrated-circuit devices, it has become a common standard to have the power and ground pins at the corners of the integrated circuit parallel adders. For pin configuration of 7483/74283, refer to Appendix C.

6-42. Cascading Parallel Adders

In order to accomplish addition of large binary numbers, two or more IC adders can be connected together (i.e. cascaded). Fig 6-46 shows two 74LS83A adders connected to add two 8-bit binary numbers, $A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$ and $B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0$.

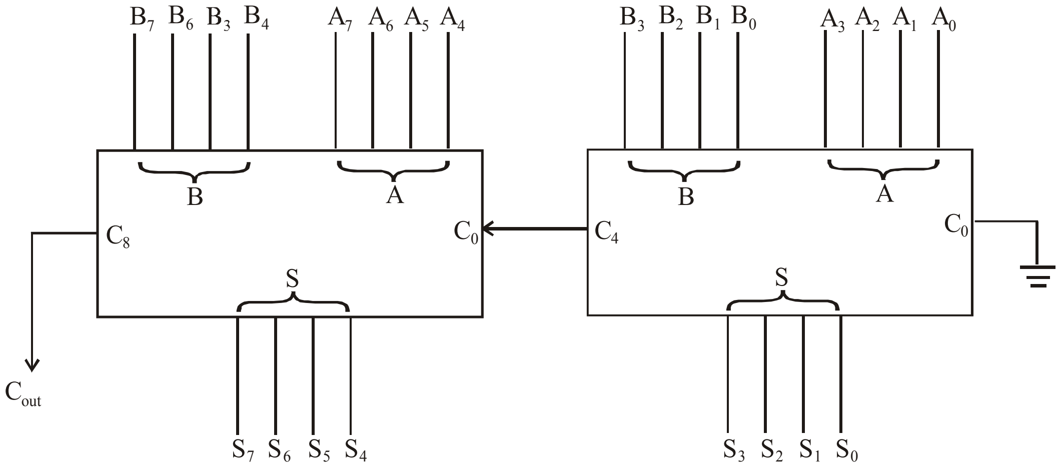
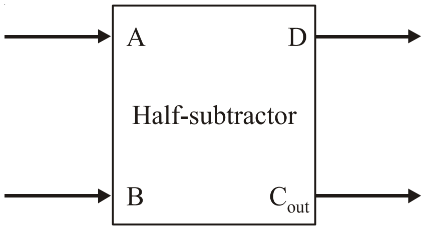


Fig. 6-46. Cascading of two 74LS283As to form an 8-bit parallel adder.

It may be noted from the figure shown above that the adder on the right adds the lower-order bits of the numbers. On the other hand, the adder on the left adds the higher-order bits plus the C_4 carry out of the lower-order adder. The eight sum outputs ($S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0$) are the resultant sum of the two 8-bit numbers. C_8 is the carry output (C_{out}) of the MSB position. The C_8 can be used as a carry input to the third adder stage if larger binary numbers are to be added.

6-43. The Half Subtractor

Fig. 6-47 (*a*) shows the logic symbol of a half-subtractor. As seen from this figure, we find that the half-subtractor accepts two binary digits on its inputs and produce two digits on it outputs : a difference bit (D) and a borrow bit (C_{out}). Fig. 6-47 (*b*) shows the truth table for the half-subtractor.



(*a*) logic symbol

Inputs		***Outputs***	
A	***B***	***D***	***C_{out}***
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

(*b*) truth table

Fig. 6-47. Half-subtractor.

The half-subtractor follows the basic rules for binary subtraction :

$$0-0 = 0$$
$$0-1 = 1 \quad \text{with a borrow of 1}$$
$$1-0 = 1$$
$$1-1 = 0$$

The Boolean expression for the difference bit (D) can be expressed by the equation.

$$D = A\overline{B} + \overline{A}B$$
$$= A \oplus B \qquad \text{...(1)}$$

and the Boolean expression for the borrow bit,

$$C_{\text{out}} = \overline{A}B$$

The equation (1) and (2) can be implemented by a logic circuit shown in Fig. 6-48. As seen from this figure, the difference output (D) is obtained from an exclusive-OR gate while the borrow bit (C_{out}) is the output of a two-input AND gate.

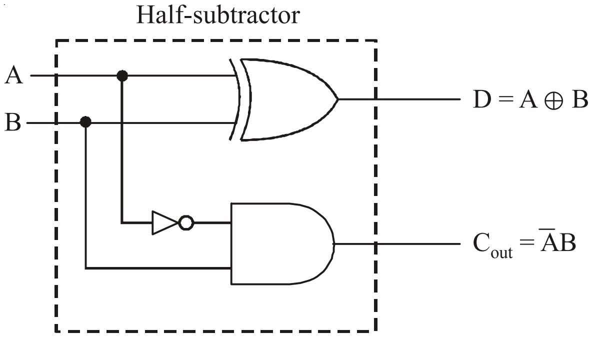
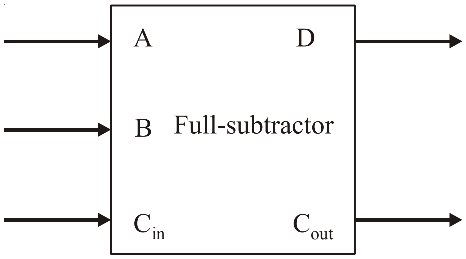


Fig. 6-48. Logic implementation of a half subtractor

6-44. The Full-Subtractor

Fig. 6-49 (*a*) shows the logic symbol of a full-subtractor. As seen from this figure, we find that the full-subtractor accepts three inputs. Two input bits *A* and *B* and a borrow bit (C_{in}). If has two outputs : (1) a difference output (D) and a borrow output (C_{out}). Fig. 6-49 (*b*) shows the truth table for the full-subtractor.



(*a*) logic symbol

Inputs			***Outputs***	
A	***B***	C_{in}	***D***	C_{out}
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

(*b*) truth table

Fig. 6-49. Full-Subtractor

The full-subtractor also follows the same basic rules for binary subtraction as the half-subtractor, thus,

$0 - 0 - 0 = 0$ with borrow 0
$0 - 0 - 1 = 1$ with borrow 1
$0 - 1 - 0 = 1$ with borrow 1
$0 - 1 - 1 = 0$ with borrow 1
$1 - 0 - 0 = 0$ with borrow 0
$1 - 1 - 0 = 0$ with borrow 0
$1 - 1 - 1 = 1$ with borrow 1

The Boolean expression for the difference output (D) can be obtained from the truth table (shows in Fig. 6-49 (b)) by summing the terms for which D = 1, Thus, the difference,

$$D = \overline{A}\,\overline{B}\,C_{\text{in}} + \overline{A}\,\overline{B}\,\overline{C}_{\text{in}} + A\overline{B}\,\overline{C}_{\text{in}} + ABC_{\text{in}} \quad \text{... (1)}$$

Let us now try to simplify the expression,

$$D = \overline{A}(\overline{B}\,C_{\text{in}} + B\overline{C}_{\text{in}}) + A(\overline{B}\,\overline{C}_{\text{in}} + BC_{\text{in}}\ ) \quad \text{... (2)}$$

Notice that the first terms within parentheses in equation (2) can be recognized as the exclusive OR combination of B and C_{in}, which can be written as $B \oplus C_{\text{in}}$. The second term within the parentheses in equation (2) can be recognized as the exclusive-NOR of B and C_{in}, which can be writhen as $\overline{B \oplus C_{\text{in}}}$. Thus the expression for D can be writhen as,

$$D = \overline{A}(B \oplus C_{\text{in}}) + A(\overline{B \oplus C_{\text{in}}}) \quad \text{... (3)}$$

Let $X = B \oplus C_{\text{in}}$, the equation (3) can be to rewrithen as,

$$D = \overline{A}\cdot X + A\overline{X}$$

$$= A \oplus X \quad \text{... (4)}$$

Replacing X with $B \oplus C_{\text{in}}$, equation (4) can be writhen as,

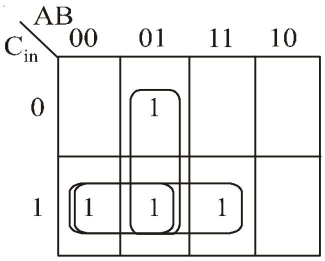
$$D = A \oplus (B \oplus C)$$

$$= A \oplus B \oplus C \quad \text{... (5)}$$

Next consider the carry output, (C_{out}) in the truth table of fig. the boolean expression for the carry output can be obtained from the truth table by summing the terms for which $C_{\text{out}} = 1$. Thus the carry output.

$$C_{\text{out}} = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + ABC \quad \text{... (6)}$$

Solving the equation (6) with K-map as shown in Fig. 6-50.



C_{in} \ AB	00	01	11	10
0		1		
1	1	1	1	

Fig. 6-50.

$$C_{\text{out}} = \overline{A}B + \overline{A}\,C_{\text{out}} + B_{\text{in}} \qquad \text{...(7)}$$

The equation (5) and (7) can be implemented by using logic gates. From equation (5) we find that to implement to full-subtractor, difference output function, two-2 input exclusive-OR gates can be used. Similars from equation (7) we find that to implement the full-subtractor borrow output function, three 2-input AND gates followed by a 3-input OR gate can be used. The complete circuit of a full-subtractor is shown in Fig. 6-51.

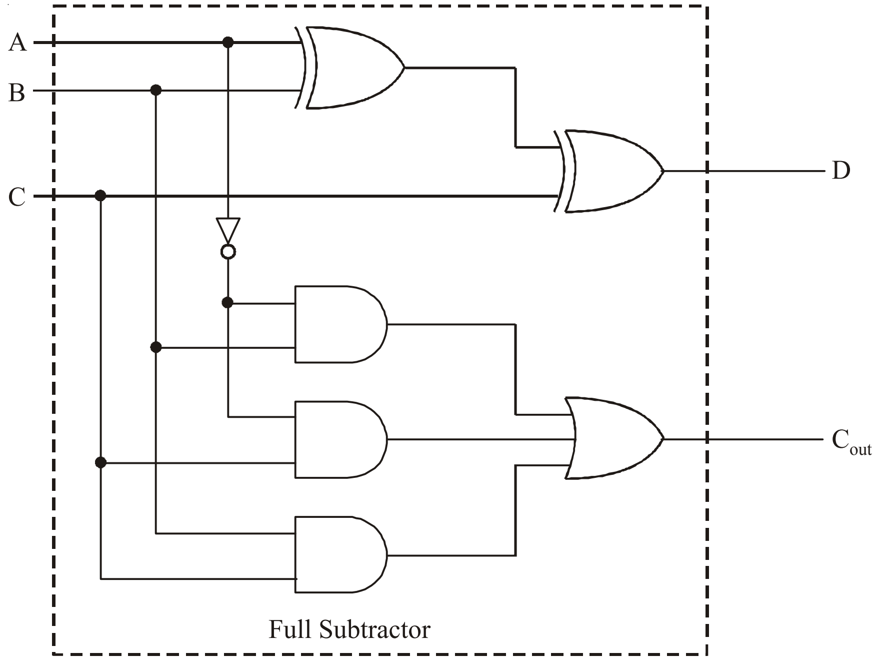


Fig. 6-51. Complete logic circuitry for a full-subtractor

6-45. 2's Complement Adder/Subractor Circuit

We have already mentioned earlier that arithmetic operations like subtraction, multiplication and division can be performed by adders and logic gates. Let us see how the parallel adder can be used to do addition and subtraction using 2's complement.

Recall that positive 2's complement numbers are exactly the same as regular true binary numbers. These can be added using regular binary addition. However, subtraction in 2's complement arithmetic is performed by converting the number to be subtracted to a negative number in the 2's complement and then using regular binary addition. Therefore once we have the numbers in 2's complement form, we can use a parallel adder to get the answer.

For example, to subtract 9 from 18, we would first convert 9 to a 2's complement number, *i.e.*,

$$\begin{array}{llcccccccc l}
+9 & \rightarrow & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \\
 & & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & \text{— 1's complement of 9} \\
 & & & & & & & & & +1 & \\
\hline
 & & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & \text{— 2's complement of 9}
\end{array}$$

The we odd 18 + (– 9), *i.e.*,

$$\begin{array}{llcccccccccl}
18 \rightarrow & & & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & \\
-9 & & & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & \text{— 2's complement of 9} \\
 & + & & & & & & & & & & \\
\hline
 & & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 &
\end{array}$$

disregard the carry. The result is $00001001_2 = 9_{10}$

Fig. 6-52 shows a circuit to implement a 2's complement adder/subtractor using 74283 adders. As seen from the circuit, 8-bit number on A inputs ($A_0 – A_7$) is brought directly into parallel adders. The other 8-bit binary number comes in on B_0 – B_7 lines. If the B number is to be subtracted, the complementing switch will be in position 'S'. this causes each bit in the B-number to be complemented (i.e it produces 1's complement of B number). At the same time, the lower-order parallel added receives $C_0 = 1$. This has the effect of adding a 1 to the already complemented B number, making it a 2's complement number. Notice that if the complementry switch is in position 'A', no complementation is done and the circuit performs normal addition.

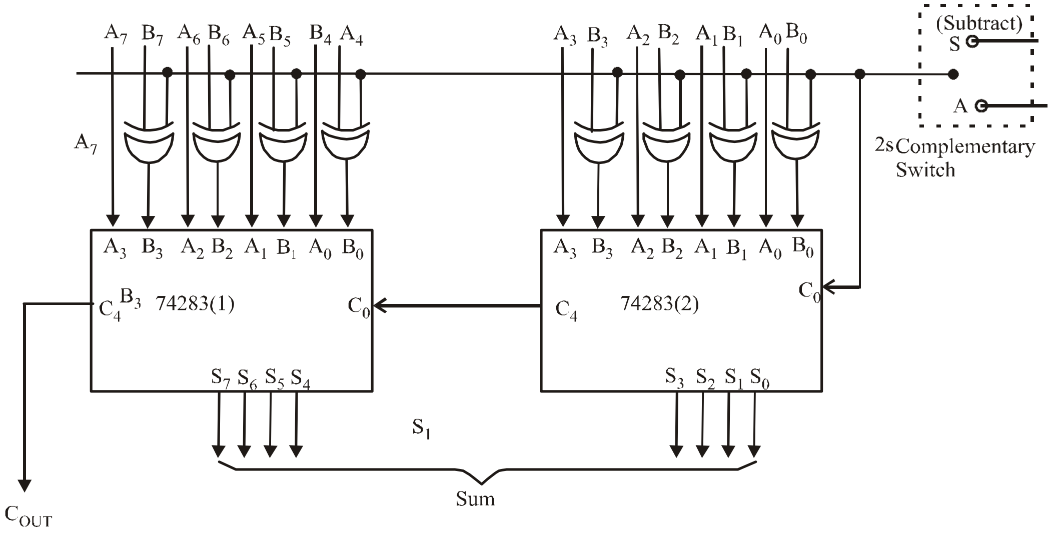


Fig. 6-52.

6-46. BCD Adder Circuit

BCD adders can also formed using the 4-bit parallel adders. However recall the problem in the BCD addition from Art 6-24 that when any group-of-four BCD sum exceeds 9, or when there is a carry-out, the number is invalid. It must be corrected by adding 6 to the invalid number to get the correct answer.

For example adding 7 and 5 using BCD addition produces an invalid result.

$$\begin{array}{rrl}
 & 0111 & \leftarrow \text{BCD code for 7} \\
 & 0101 & \leftarrow \text{BCD code for 5} \\
\cline{2-2}
 & 1100 & \leftarrow \text{Invalid BCD code} \\
 & +110 & \leftarrow \text{Add 6 for correction} \\
\cline{2-2}
\underbrace{0001}_{1} & \underbrace{0010}_{2} & \leftarrow \text{BCD code for 12}
\end{array}$$

Checking for a sum greater than 9 (or a carry out) can be done easily using logic gates. Then whenever an invalid sum occurs, it can be corrected by adding 6 (i.e 0110) via the connections as shown in Fig. 6-53.

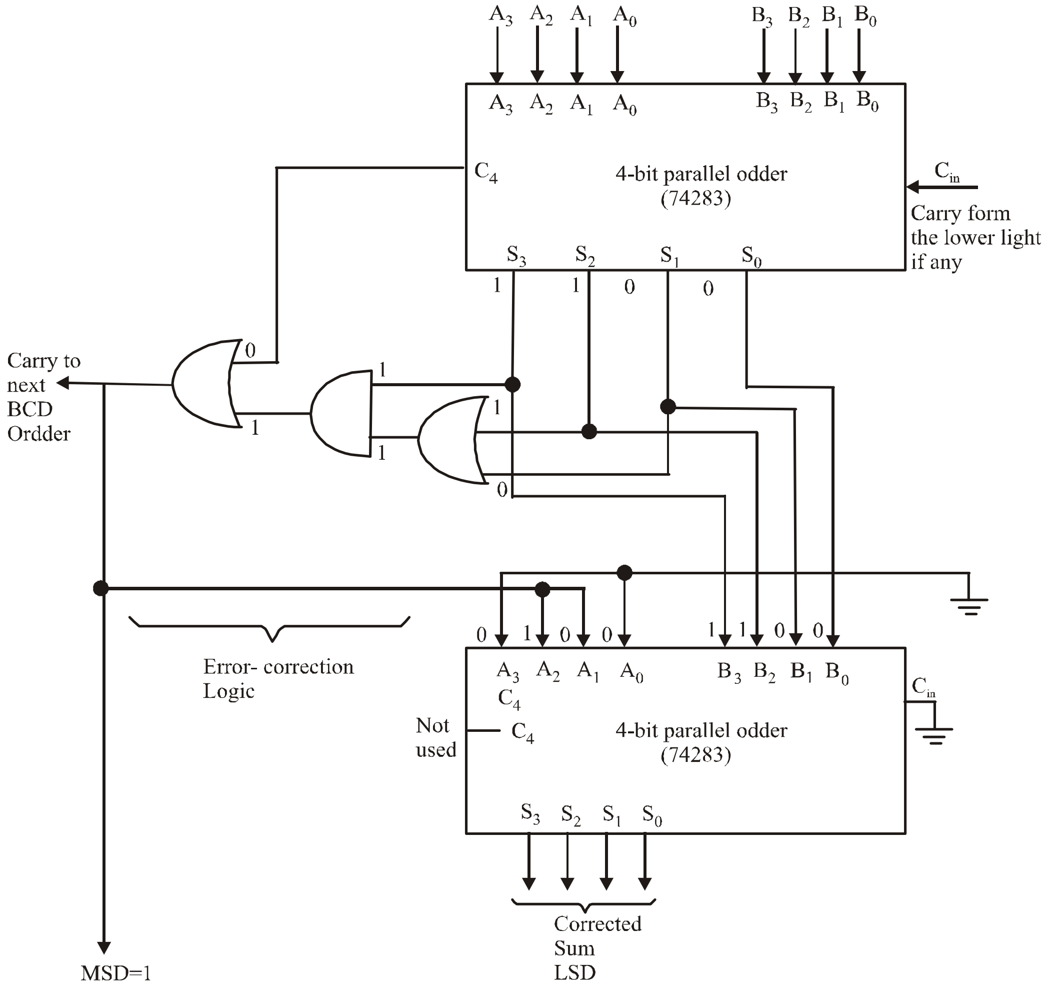


Fig. 6-53.

A 4-bit BCD adder is available in a single IC package. The IC74583 has an internal error correction logic circuitry to add two 4-bit numbers and produce a corrected 4-bit result with carry output.

6-47. Binary Subtractor Circuit

The subtraction of two number can be done by taking the 2's complement of one number and adding it to other number. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant fair of bits. The 1's complement can be implemented with inverters and a 1 can be added to the sum through the input carry.

For subtracting two number *A* and *B*, the circuit consists of an adder with inverters placed between each date input of *B*. The input carry C_0 must be equal to 1 when subtraction is performed. The operation thus become *A*, plus the 1's complement of *B*, plus 1. This is equal to *A* pluse the 2's complement of *B*.

Fig. 6-54. shows the block diagram of a four-bit parallel subtractor. As seen, it consists of four full-adder connected together. The carry output of each adder is connected to the carry input of next higher stage. There is the mode input M which controls the operation. When $M = 0$, the circuit work like a adder, and when $M = 1$ it become subtractor circuit. Each exclusive-OR gate receives input M and one of the inputs of B

when $M = 0$, then operation,

$$\begin{aligned} B \oplus 0 &= B\overline{0} + \overline{B}0 \\ &= B \end{aligned}$$

when $M = 1$, then operation,

$$\begin{aligned} B \oplus 1 &= B\overline{1} + \overline{B}1 \\ &= \overline{B} \end{aligned}$$

Thus we see that all input B are complement when $M = 1$. The carry 1 is added, through the input carry. The circuit performs the operation A plus the 2's complement of B.

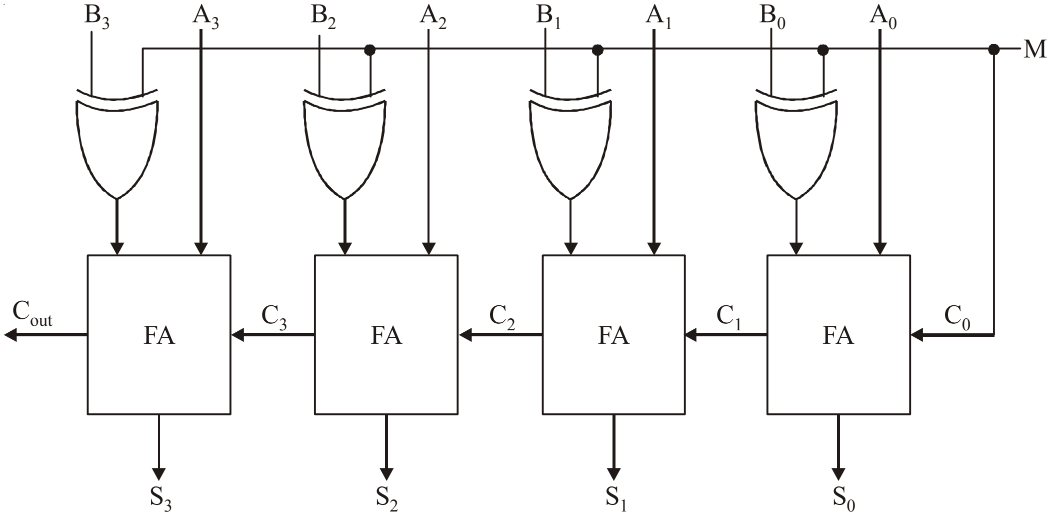


Fig. 6-54. Logic symbol of Subtractor

6-48. Binary Multiplier Circuit

The multiplication of binary number is done same way as in decimal multiplication. To see how a binary multiplier work let us consider the two 2-bit binary numbers as shown in Fig. 6-55. The multiplic bit are B_1 and B_0, the multiplier bit are A_1 and A_0, and the product is $C_3C_2C_1C_0$. The first partial product is formed by multiplying B_1B_0 by A_0. The second partial product is formed by multiplying B_1B_0 by A_1 and shifting one position to the left. The partial products are added with two half adder (HA) circuits. Usually there are mox bits in the partial products and it is necessary to use full adders to produce the sum of the partial products. A combinational circuit binary multiplier with mox bits can be constructed in a similar fashion.

$$\begin{array}{cccc} & & B_1 & B_0 \\ & & A_1 & A_0 \\ \cline{3-4} & & A_0B_1 & A_0B_0 \\ & A_1B_1 & A_1B_0 & \\ \hline C_3 & C_2 & C_1 & C_0 \end{array}$$

Fig. 6-55.

The combinational circuit for 2 bit multiplier is shown in Fig. 6-56. The partial product can be implemented with AND gates as shown in figure. The two partial products are added with half adder circuits.

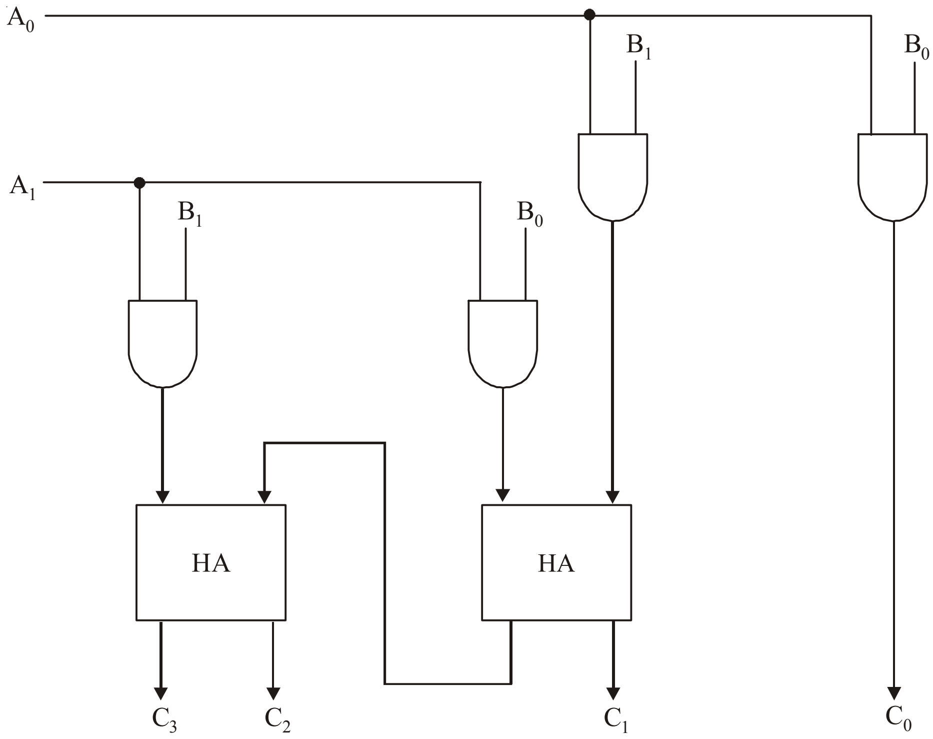


Fig. 6-56.

For *l* multiplier bits and *m* multiplicand bits, we need $(l \times m)$ AND gates and $(l - 1)$ m bit adders to produce a product of $(l + m)$ bits.

Lets take an other example, consider a multiplier circuit that multiplier a binary number represented by four bits by a number represented by three bits. Lets multiplicand $B_3B_2B_1B_0$ and the multiplier by $A_2\,A_1\,A_0$. Since $m = 4$ and $l = 3$. We need 12 AND gates and 2 four bit adders to produce a product of seven bits. The logic implementation of the combinational circuit is shown in Fig. 6-57.

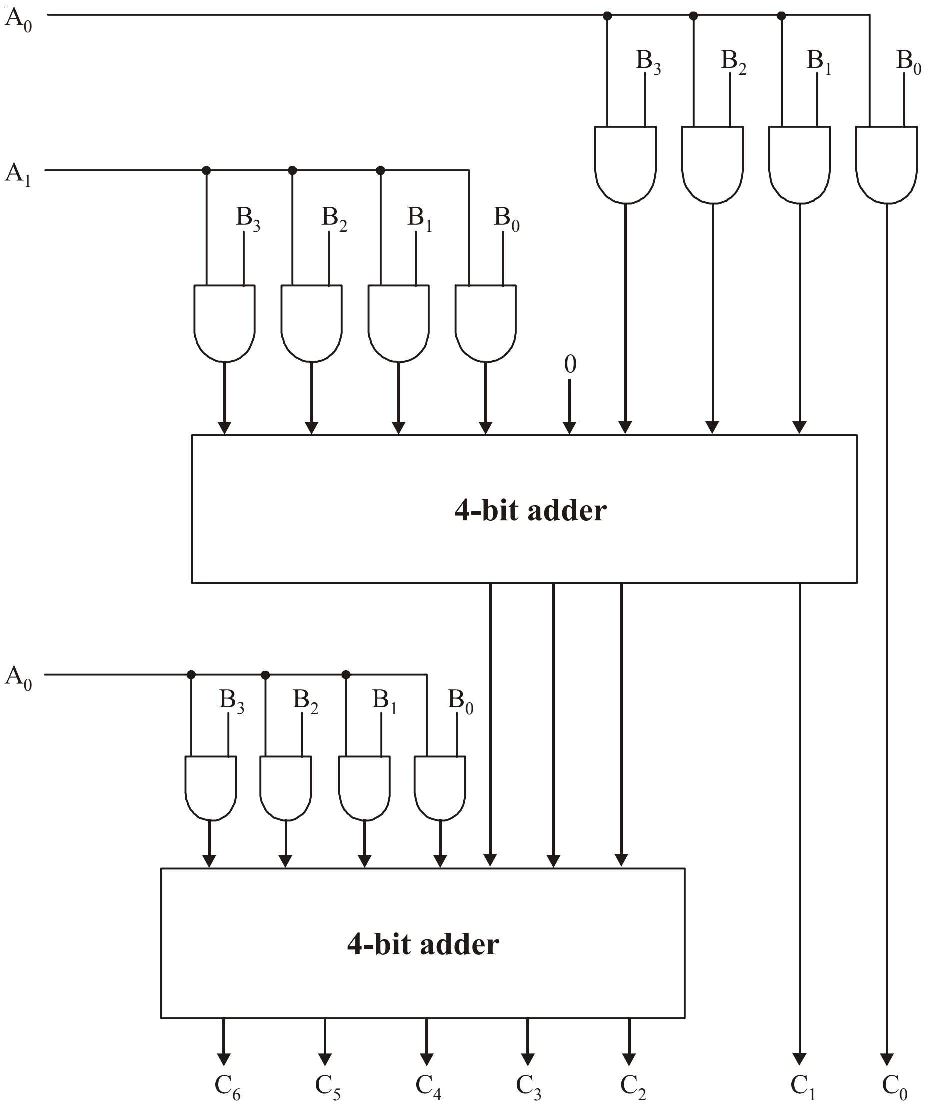


Fig. 6-57.

6-49. Magnitude Comparator

A magnitude comparator is a combinational circuit that compares two number A and B to determine whether the number is greater than less than or equal to the other number. It generates outputs to indicate which one has the greater magnitude.

Fig. 6-58 shows the block diagram of an n-bit magnitude comparators. It receive two n-bit numbers A and B as inputs and the outputs are $A > B$, $A = B$ and $A < B$. Outputs depend upon the relative magnitude of the two numbers.

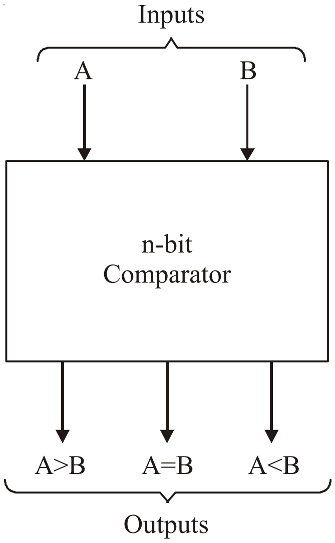


Fig. 6-58. Block diagram of n-bit comparator.

Let take an example of 1 bit comparator. The inputs are *A* and *B*. There will be those outputs,

$$A > B,\ A = B,\ A < B$$

The magnitude comparator is implement with Ex-NOR and AND gate. Fig. 5.59, shows an Ex-NOR gate with two inputs *A* and *B*. If $A = B$ the output of an Ex-NOR gate is equal to 1 otherwise 0.

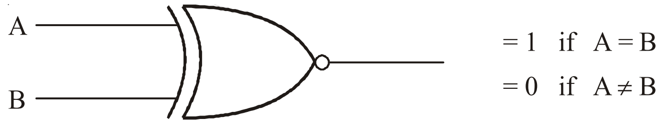
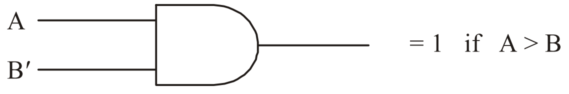


Fig. 6-59.

If the input *A* and *B* are not equal. We use AND gate to find the *A* is greater than *B* or less than *B*. Fig. shows AND gate with input *A* and *B*′. Truth table is shown is Fig. 6-60. From the truth table we seen that the output is 1 if *A* greater than *B* otherwise 0.



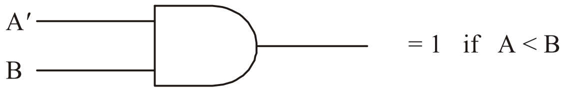
(*a*)

Inputs		***Outputs***
A	***B***	
0	0	0
0	1	0
1	0	1
1	1	0

(*b*)

Fig. 6-60.

Same as in the case, if *B* is greater than *A*. Fig. 6-61 shows the AND gate with input *A*′ and *B*. Truth table is shown in Fig.



(*a*)

Inputs		*Outputs*
A	*B*	
0	0	0
0	1	1
1	0	0
1	1	0

(*b*)

Fig. 6-61.

If Ex-NOR gate and two AND gates are combined as shown in Fig. 6-62. Then this circuit is single bit comparator. The truth table is shown in Fig. 6-63.

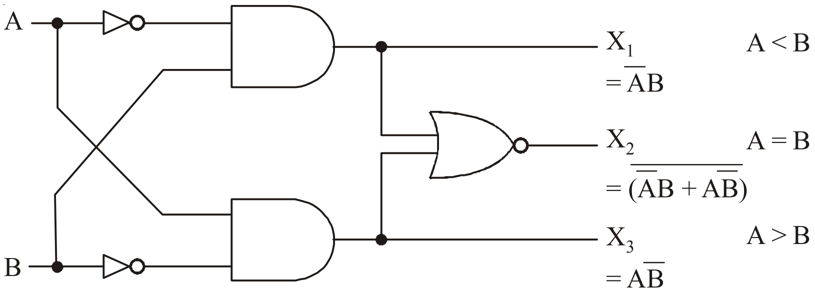


Fig. 6-62.

Inputs		*Outputs*		
A	*B*	X_1	X_2	X_3
		A < *B*	A = *B*	*A* > *B*
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Fig. 6-63.

If clearly shown from the truth table that the output X_1 is high when $A < B$, X_2 is high only when the A is equal to B and X_3 is high when $A > B$.

4-bit Magnitude Comparator

Let us consider two number A and B, with four bit each.

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

The two number A and B are equal if all pairs of significant digits are equal,

$$A_3 = B_3$$

$$A_2 = B_2$$
$$A_1 = B_1$$
$$A_0 = B_0$$

The binary digits are either 1 or 0. We have seen that equality relation is generated by Ex-NOR gate.

$$X_i = A_i B_i + \overline{A}_i \overline{B}_i$$

where $i = 0, 1, 2, 3$

$$X_i = 1 \text{ if } A_i = B_i$$
$$= 0 \quad \text{otherwise.}$$

The combinational circuit display the equality of two number A and B. This output is designate by the symbol $A = B$. This binarey is equal to 1 if the input numbers A and B are equal and is equal to 0 otherwise. Means that the binary variable $A = B$ is equal to 1 only if all pairs of digits of two number are equal.

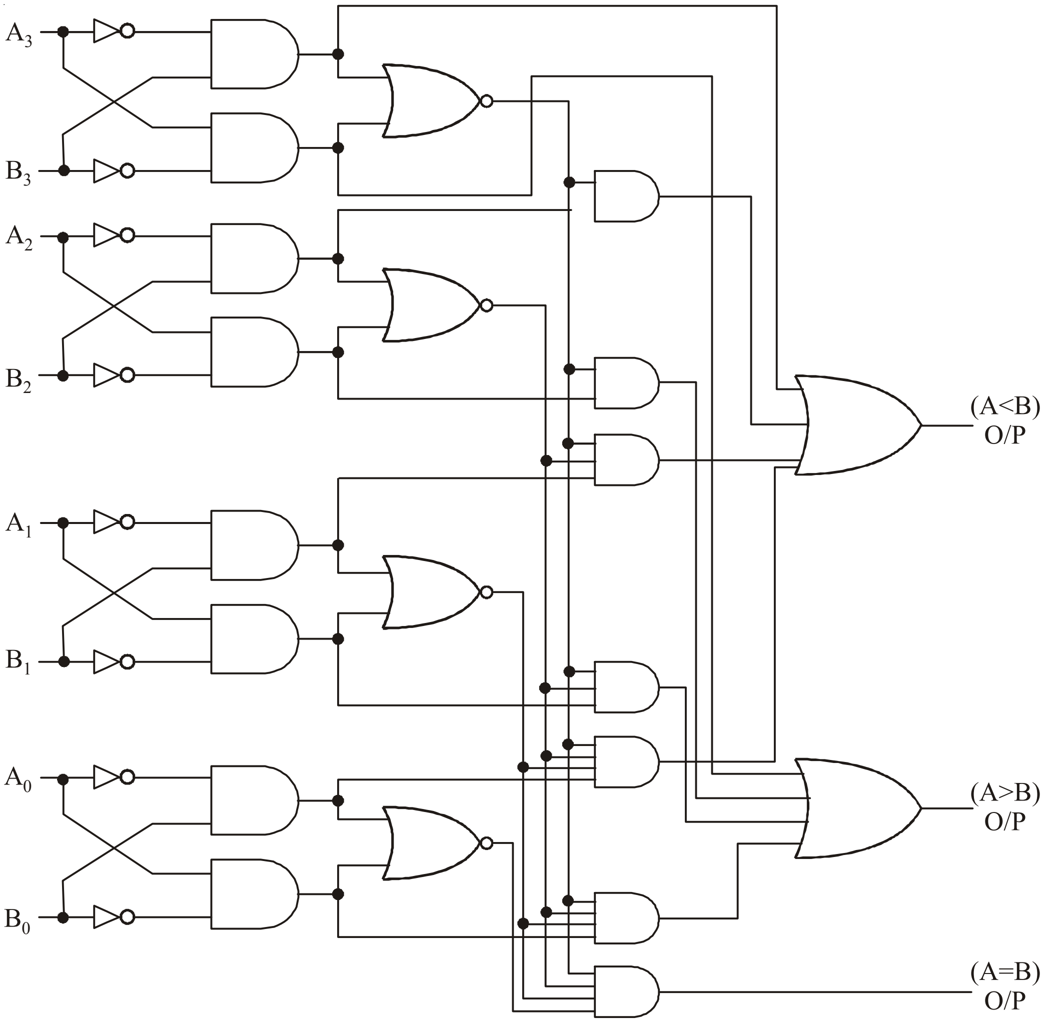


Fig. 6-64.

To determine whether A is greater or less than B. We consider the relative magnitude of pairs of significant digits starting from the MSB position. If the two digits are equal then we compare the next lower significant pair of digits.

The comparision follows until a pair of unequal digits are reached. If the corresponding digit of A is 1 and that of B is 0, then $A > B$. If the corresponding digit A is 0 and that of B is 1, then $A < B$.

This discussion can be expressed logically as

$$(A > B) = A_3 \overline{B}_3 + X_3 A_2 \overline{B}_2 + X_3 X_2 A_1 \overline{B}_1 + X_3 X_2 X_1 A_0 \overline{B}_0$$

$$(\mathrm{A} < \mathrm{B}) = \overline{A}_3 B_3 + X_3 \overline{A}_2 B_2 + X_3 X_2 \overline{A}_1 B_1 + X_3 X_2 X_1 \overline{A}_0 B_0.$$

The logic diagram of 4-bit comparator is shown in Fig. The four output X_0, X_1, X_2 and X_3 are generated with Ex-NOR gates are applied to an AND gate (Gate 1). Its give the output binary A + B. The OR gate (Gate 2) gives the output binary A > B and the OR gate (Gate 3) gives the output binary A < B.

SUMMARY

In this chapter, you have learned that :

1. To represent signed numbers in binary, a sign bit is attached as the most-significant bit. A positive sign is represented by a 0 bit and a negative sign by a 1 bit.

2. The 1's complement of a binary number is obtained by complementing all the bits of a given number.

3. The 2's complement of a binary number is obtained by adding 1 to 1's complement.

4. In an addition operation, overflow is possible when both numbers are positive or when both numbers are negative. An incorect sign bit in the sum indicates the occurrence of an overflow.

5. Subtraction can be performed on signed binary numbers by negating the subtrahend and adding it to the minuend.

6. A full-adder performs the additions on two bits plus a carry input. A parallel binary adder is made up of cascaded full-adders.

GLOSSARY

Augend. It is a number to which an addend is added.

Addend. It is a number to be added to another number.

Carry. A digit or a bit that is generated when two numbers are added and the result is greater than the base for the number system being used.

Full adder. A digital circuit that adds two bits and an input carry to produce a sum and an output carry.

Half adder. A digital circuit that adds two bits and produces a sum and an output carry. It cannot handle input carries.

Minuend. A number from which the subtrahend is to be subtracted.

Look-ahead carry. It is an ability of some parallel adders to predict, without having to wait for the carry to propagate through the full adders.

Over flow. During the process of adding signed binary numbers a carry of 1 is generated from the MSB position of the number into the sign bit position.

Parallel adder. A digital circuit (made from full adders) that is used to add all the bits of the given two numbers simultaneously.

Sign-magnitude system. It is a system for representing signed binary numbers where MSB represents sign of a number and the remaining bits represent the true binary value.

Subtrahend. A number that is used to be subtracted from the minuhend.

2's Coomplement system. It is a system for representing negative binary numbers. In this system the MSB (equals 1) represents the sign of a number and the remaining bits represent 2's complement of a number.

DESCRIPTIVE QUESTIONS

1. Under what circumstances would you use a half-adder instead of a full- adder circuit.
2. Reconstruct the half-adder circuit of Fig 6-64 using (*a*) only NOR gates (*b*) only NAND gates.
3. The circuit shown in Fig 6-65 is an attempt to build a half-adder. Will the sum and carry output function properly? (Hint: Write the Boolean expression for the sum (S) and carryout (C_{out}) terminals.

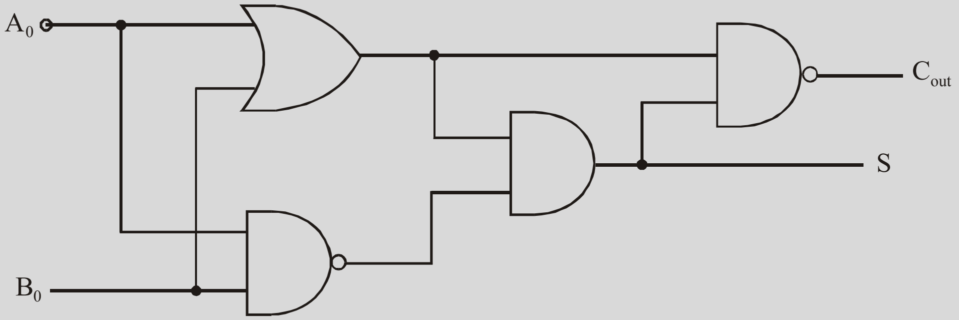


Fig. 6-65.

4. Use the K-map to prove that the carry output (C_{out}) function of the full-adder whose truth table is given in Fig 6-5 (b) can be implemented using Fig 6-66.

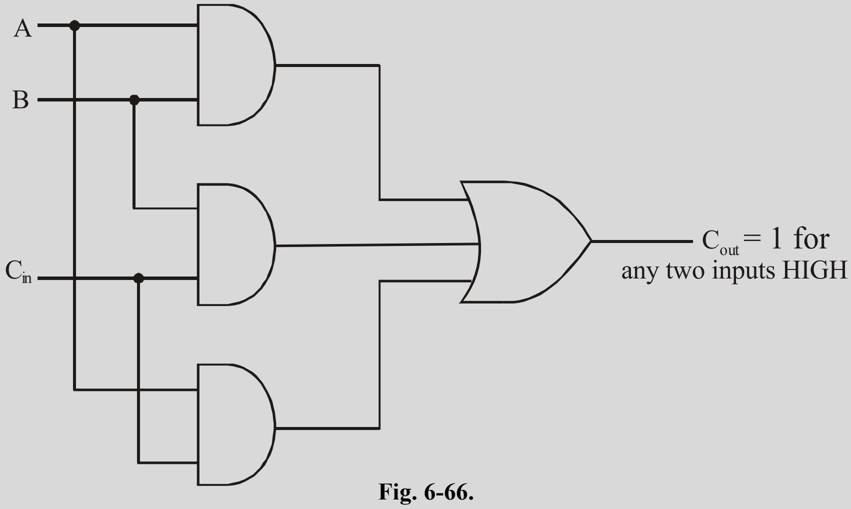


Fig. 6-66.

5. Draw the truth table for a three-input adder. Explain clerly the meaning of input and output symbols of the table. Write the Boolean expression for sum and carry.

6. Draw the schematic of a BCD adder using 4-bit binary adders. Explain the operation with suitable examples.

7. Explain the working of a half-adder circuit with the help of truth tables. Realize a full-adder using half-adders and/or gates.

8. Using 4-bit parallel binary adder, design a circuit to perform BCD addition. Show different conditions with different examples.

9. Sketch a block diagram of a 4-bit full-adder using four full-adders.

10. Implement Full adder using two half adders.

(*Anna University, Nov./Dec. 2007*)

11. Perform 7-15 using 2's complementary.

(*Gauhati University, 2007*)

12. What is sign-magnitude representation of numbers ?

(*Gauhati University, 2006*)

13. Draw a circuit diagram and explain the four –bit addition with carry look ahead logic.

(*Mahatma Gandhi University, Nov. 2005*)

14. Design a 3 bit carry look ahead adder.

(*PTU, May 2007*)

15. How will you detect overflow in signed magnitude and 2's complement integer additions?

(*PTU, Dec., 2009*)

16. Represent a half adder in block diagram and also its logic implementation.

(*Anna University, May / Jun. 2006*)

17. Design a full adder circuit using NAND gates only.

(*PTU, May 2009*)

18. With a block diagram describe the principle of operation of a carry Look-Ahead adder.

(*VTU, Jul. 2006*)

19. Explain a 4-bit parallel adder with carry lookhead scheme.

(*VTU, Jan./Feb. 2006*)

20. Give the truth table for a binary full adder function and obtain the irredundant disjunctive normal expression for the function. Show how the function could be realized using NAND gates.

(*VTU, Jan./Feb. 2005*)

21. Explain a 4 bit parallel adder with the carry look ahead scheme. Clearly indicate how this scheme improves the performance of the operation.

(*VTU, Jan./Feb. 2004*)

22. Describe a full subtractor with its truth table.

(*Mahatma Gandhi University, May 2005*)

23. Realise half subtractor using NAND gates only.

(*Mahatma Gandhi University, Nov. 2005*)

24. Realize a full adder using two half adders.

(*Mahatma Gandhi University, May/Jun. 2006*)

25. What are half and full adders? With the help of necessary truth tables and k-maps show how a full adder can be realized using only half adders and OR gates.

(*Mahatma Gandhi University, Dec. 2007*)

26. Defferentiate between Carry Save adder and Carry look ahead adder.

(*Mahatma Gandhi University, May/Jun. 2006*)

27. Explain with figures the principle of operation of a Look ahead carry adder.

(*Mahatma Gandhi University, Jan. 2007*)

28. How will you implement a full subtractor using half subtractor?

(*Nagpur University, 2008*)

29. What are half subtractor and full subtractor? Describe the functions of a 4-bit binary adder/subtractor circuit.

(*Gauhati University, 2006*)

30. Draw the schematic of a full adder with the help of a truth table. Explain how a full adder works. Give the expressions for sum and carry in full adder.

(*RGPV Bhopal, June 2008*)

31. Explain the full adder with help of circuit diagram using NAND gate write the truth table.

(*Gujarat Technological university, Dec. 2009, GBTU., 2010-11*)

32. What are half subtractor and full subtractor? Explain using truth table.

(*Gauhati University, 2003*)

33. Design a 3 bit magnitude comparator using gates.

(*Gauhati University, 2003*)

34. Implement 8 bit adder using two 4 bit full adders.

(*Nagpur University, 2004*)

35. Discuss the process of complement of a number. Also show the 1's and 2's complement subtraction using example.

(*GBTU/MTU, 2006-07*)

36. Design a two digit BCD adder with the help of 4-bit binary adders.

(*GBTU/MTU, 2006-07*)

37. What are the different types of parallel adders? Explain carry save and carry look-ahead adders.

(*GBTU/MTU, 2006-07*)

38. What is magnitude comparator? Design a two bit comparator using logic gates.

(*GBTU/MTU, 2009-10*)

39. Explain the procedure of binary multiplier.

(*GBTU/MTU, 2009-10*)

40. Design a combinational circuits the adds one to a 4-bit binary number. The circuit can be designed using four half adders.

(*GBTU/MTU, 2009-10*)

41. Show that a full subtractor can be constructed with –two half subtractor and an OR gate.

(*GBTU/MTU, 2009-10*)

42. What is full subtractor?

(*GBTU/MTU, 2006-07*)

43. What do you mean by full adder? Give the truth table for full adder and express it as a function. Show how 8 full adders may be used to form an 8 bit parallel adder.

(*GBTU/MTU, 2007-08*)

44. Explain the signed binary number representation.

(*GBTU/MTU, 2009-10*)

45. What do you mean by arithmetic circuit with respect to digital electronics ? Implement the circuit which performs the addition using logic gates.

(*GBTU/MTU, 2006-07*)

46. Realize a circuit which is capable of performing addition and subtraction with the same circuits.

(*GBTU/MTU, 2006-07, 2007-08*)

47. Design a circuit which compare two binary number whether $A>B$, $A=B$ or $A<B$.

(*GBTU/MTU, 2006-07*)

48. Design BCD to excess-3 converter.

(*Nagpur University, 2004*)

49. Explain how BCD adder (decimal adder) is differ from a binary adder. Also discuss the concept of correction logic in BCD adders.

(*GBTU/MTU, 2006-07*)

50. Design a combinational circuit which accepts BCD number as its input and generate equivalent excess code as its output. Also implement the circuit using NAND logic.

(*GBTU/MTU, 2006-07*)

51. Explain the working of 4-bit serial adder/subtractor.

(*RGTU., June, 2009, 2010*)

52. Design a BCD to excess-3 converter with a BCD to decimal decodes and four OR gates.

(*RGTU., June 2009*)

53. Explain the look ahead carry generator and discuss its utility in adders.

(*RGTU., June 2009*)

54. Describe full adder and full subtractor with diagram and table.

(*RGTU., Dec. 2010*)

55. Explain the working of a look ahead carry generator.

(*RGTU., June 2010, 2011*)

56. Design a BCD adder.

(*RGTU., June 2011*)

57. Implement a full adder circuit using two half adders and an OR gate.

(*RGTU., June 2011, GTU., May 2011*)

58. Design a full adder.

(*RGTU., March-April 2010*)

59. Draw the circuit diagram of half adder with the help of full table. Also give expression for sum and carry.

(*RGTU., Feb. 2010*)

60. Design a 8-bit BCD adder.

(*GBTU., 2010-11*)

61. Design a Binary to BCD code converter.

(*RGTU., Dec. 2009*)

62. Design a BCD adder using full adders.

(*RGTU., Dec. 2009*)

63. Design a BCD to Binary code converter.

(*RGTU., June 2010*)

TUTORIAL PROBLEMS

1. Determine the 2's complement of the following 8-bit numbers:

(*a*) 01011100 (*b*) 00100110 (*c*) 00001111

(**Ans.** (*a*) 10100100, (*b*) 11011010, (*c*) 11110001).

2. Express the decimal number – 37 as a binary number using sign-magnitude system.

(**Ans.** 1100101).

3. Determine the range of signed decimal values that can be represented in 16 bits (including the sign bit). (**Ans.** -2^{15} to $(2^{15}-1)$).

4. Determine the largest negative decimal value that can be represented by a 4-byte number.

(**Ans.** -2^{-31}).

5. List in order, all the signed numbers that can be represented in five bits using the 2's complement system.

6. Perform the following operations in the 2's complement system. Use eight bits (including the sign bit) for each number.

(*a*) Add 22 to 6 (*b*) Add 15 to – 4 (*c*) Add 36 to – 84 (*d*) Add – 48 to – 80

(**Ans.** (*a*) 00011100_2, (*b*) 00001011_2, (*c*) 1010000_2, (*d*) 10000000_2).

7. Memory locations in personal computers are usually given in hexadecimal. If a computer programmer writes a program that requires 100 memory locations, determine the last memory location that is used if the program starts at the location 3C9FH where H = base 16 hexadecimal.

(**Ans.** 3D02 H).

8. A particular model of personal computer indicates in its owner's manual that the following memory locations are used for storage of operating system subroutines: 07A2BH to 0AD68H inclusive and 02D8H to 03E00H inclusive. Determine the total number of memory locations used for this purpose in hexadecimal number. (**Ans.** 62BD H).

MULTIPLE CHOICE QUESTIONS

1. The 1's complement of 10111001 is

(*a*) 01000111 (*b*) 01000110 (*c*) 11000110 (d) 10101010

2. The 2's complement of 11001000 is

(*a*) 00110111 (*b*) 00110001 (*c*) 01001000 (*d*) 00111000

3. In the 2's complement system, the number 10010011_2 is equal to

(*a*) -19_{10} (*b*) $+109_{10}$ (*c*) $+91_{10}$ (*d*) -109_{10}

4. A binary number B can be subtracted from another binary number A by

(*a*) adding A and B and takign 2's complement of sum

(*b*) adding A with 2's complement of B

(*c*) adding 2's complement of A with B

(*d*) adding 2's complement of both A and B

(*AMIE., Winter 2000*)

5. A certain memory location holds the hexadecimal data 77. If this represents an unsigned number, the decimal value is:

(*a*) + 77 (*b*) + 119 (*c*) – 119 (*d*) None of these

6. A certain memory location holds the hexadecimal data E5. If this represents a signed number, the decimal value is

(*a*) +229 (*b*) – 229 (*c*) – 27 (*d*) None of these

7. The owner's manual for a small microcomputer system states that the computer has usable memory locations at the following addresses: 0200 through 03FF and through 4000 through 7FD0. The total number of available memory locations are:

(*a*) 01FF (*b*) 3FD0 (*c*) 41CF (*d*) None of these

8. In a half-adder having two inputs (A and B) and two outputs (Sum (S) and carry (C_{out})), the Boolean expression for S and C_{out} in terms of A and B. are:

(*a*) $S = \overline{A}B + A\overline{B};\ C = A.B$ (*b*) $S = AB + \overline{A}B;\ C = A + B$

(*c*) $S = \overline{A}\overline{B} + AB;\ C = A + \overline{B}$ (*d*) $S = \overline{A}B + A\overline{B};\ C + \overline{A} + B$

9. A full-adder can be made out of:

(*a*) two half-adders (*b*) two half-adders and a NOT gate

(*c*) two half-adders and an OR gate (*d*) two half-adders and an AND gate

(*UPSC Engg. Services 1997)*

10. The m-bit parallel adder consists of

(*a*) m + 1 full adders (*b*) m/2 full adders

(*c*) (m – 1) full adders (*d*) m full adders

(*AMIE., Winter 2000*)

11. A full adder can be constructed by using

(*a*) 2 AND gates, 3 XOR gates and OR gate

(*b*) 3 AND gates, 2 XOR gates and an OR gate

(*c*) 3 AND gates, 2 XOR gates and 2 or gates

(*d*) 2 AND gates, 2 XOR gates and 3 OR gates

(*AMIE., Winter 2000*)

12. A carry look ahead adder is frequently used for addition because it.

(*a*) is faster (*b*) is more accurate (*c*) uses fewer gates (*d*) costs less

ANSWERS

1. (*b*)	**2.** (*d*)	**3.** (*b*)	**4.** (*b*)	**5.** (*b*)	**6.** (*c*)
7. (*c*)	**8.** (*a*)	**9.** (*c*)	**10.** (*b*)	**11.** (*b*)	**12.** (*a*)

7

COUNTERS AND SHIFT REGISTERS

OUTLINE

Objectives

Upon completion of this chapter, you should be able to:

- Describe the difference between asynchronous and synchronous counters.
- Understand the operation of asynchronous and synchronous counters.
- Analyse counter timing diagrams.
- Construct counters with any MOD number less than 2^N.
- Recognize the difference between a 4-bit binary counter and a decade counter.
- Know the various types of presettable counters.
- Understand the different methods to decode counters.
- Compare and contrast between ring and Johnson counters.
- Understand the operation of various types of IC registers.

7-1. Introduction

We have already discussed in Chapter 5 that flip-flops can be connected together to perform counting and shift register operations. Now we will study these two flip-flop applications in more detail.

The counters may broadly be classified into the following two categories.(1) asynchronous counters (also called ripple counters) and (2) synchronous counters. In asynchronous (ripple) counter, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the proceeding flip-flop. On the other hand, in synchronous counters, the clock input is connected to all of the flip-flops. So that they are clocked simultaneously.

Within each of the two categories, counters are classified further primarily by the type of sequence, the number of states or the number of flip-flops in the counter.

Let us now begin our study on counters and registers in the following pages.

7-2. Sequential Circuits

Counters and shift registers fall under the category of sequential circuits. A sequential circuit is a combinational circuit in which a storage elements or memory elements are connected to its feedback path. The block diagram of sequential circuit is shown in Fig. 7-1. The binary information stored in these elements at any given time defines the state of the sequential circuit at that time. The sequential circuit receive binary information from external inputs that together with the present stage of the storage elements, determine the binary value of the outputs.

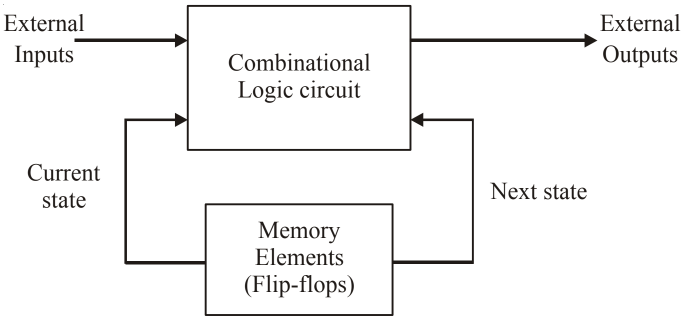


Fig. 7-1. Block diagram of sequential circuits.

There are two types of sequential circuits:

1. Synchronous Sequential Circuit: If is also know as clocked sequential circuit. In this circuit memory elements are clock flip-flop.
2. Asynchronous Sequential Circuit: In this circuit memory element are either unclocked flip-flops or time delay elements. The change in input signals can affect memory element at any instant of time. In asynchronous sequential circuits clock is absent and state change occurs according to delay times of the logic.

The asynchronous sequential circuits are more difficult to design. The asynchronous sequential circuits are faster than the synchronous sequential circuit because the clock is absent in asynchronous sequential circuits.

7-3. Asynchronous (or Ripple) Counters

The term asynchronous refer to the events that do not have a fixed time relationship with each other and generally do not occur at the same time. An asynchronous counter is one in which the flip-flops within the counter do not change states at exactly the same time because they do not have a common clock pulse.

Fig. 7-2 shows a 3-bit counter connected for asynchronous operation. Notice that the clock is applied to the clock input (CLK) of only the first flip-flop (A). The second flip-flop (B) is clocked or (triggered) by the output of flip-flop A. Similarly, the flip-flop C is clocked by the output of flip-flop B. Following are some of the important points concerning its operation:

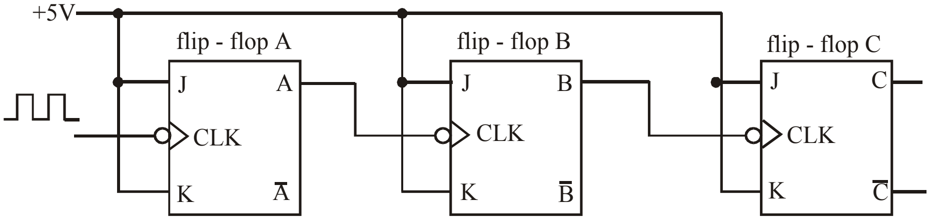


Fig. 7-2. 3-bit Asynchronous (or ripple) counter.

1. The J and K inputs of all the three A,B and C flip-flops are tied to +5V (i.e. HIGH or logic 1 state).
2. The clock pulses are applied to the CLK input of the flip-flop A. Thus flip-flop A will toggle (i.e. change to its opposite state) each time the falling edge of the clock pulse arrives.

3. The normal output of flip-flop A acts as the CLK input for flip-flop B. Thus the flip-flop B will toggle each time the A output goes from 1 to 0. Similarly, the flip-flop C will toggle each time when B output goes from 1 to 0.
4. The flip-flop outputs, C, B and A represents a 3-bit binary number with C as the most-significant bit (MSB) and A as the least-significant bit (LSB). Assume that the circuit goes through 000 to 111 as the clock pulses are continuously applied.
5. After the falling edge of the seventh clock pulse has occurred, the counter flip flops are in 111 condition. On the arrival of eighth clock pulse, flip-flop A goes from 1 to 0 which causes B to go from 1 to 0. This in turn causes flip-flop C to go from 1to 0. Thus the counter is in 000 state. In other words, the counter has gone through one complete cycle i.e. 000 through 111 and recycled back to 000 state. From here onwards, it starts a new counting cycle as subsequent clock pulses are applied.

It may be noted that in the asynchronous counter shown in Fig 7-2, the flip-flops do not change in states in exact synchronism with the applied clock pulses. As seen from the waveforms shown in Fig 7-3, only flip-flop A responds to the clock pulses. The flip-flop B must wait for flip-flop A to change states before it can toggle. Similarly, the flip-flop C must wait for flip-flop B to toggle. Thus there is a delay between the responses of successive flip-flops. The value of this delay is typically 5-20 ns per flip-flop. This delay can be troublesome in some situations.

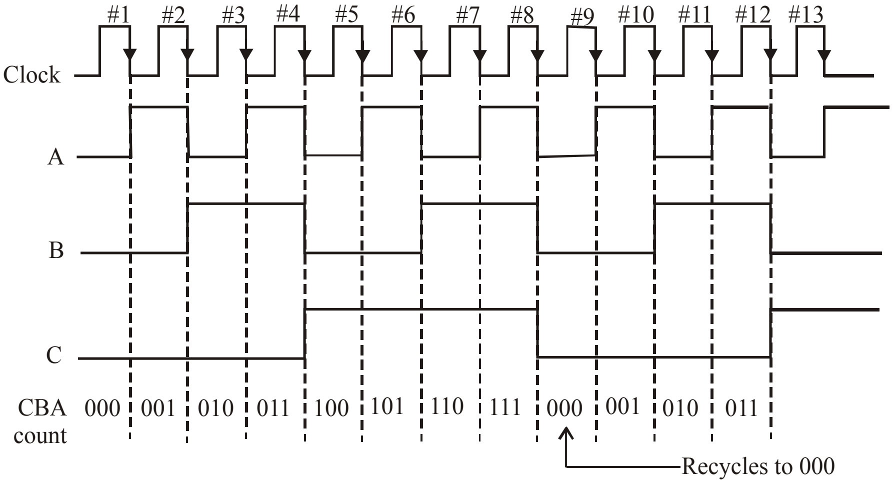


Fig. 7-3. Illustrating the waveforms at flip-flop output for counter operation

The asynchronous counter is also called a ripple counter due to the manner in which the flip-flops respond one after another in a kind of rippling effect. We will use the terms asynchronous counter and ripple counter interchangeably in the latter part of the chapter as well as the book.

Example 7-1. *The counter shown in Fig.* 7-2 *starts off in the* 000 *state and then clock pulses are applied. Sometime later the clock pulses are removed and the counter flip-flops read* 101. *How many clock pulses have occurred.*

Solution. Given : The binary count = 101

Since binary count 101 is equivalent to decimal 5, the answer appears to be 5. However, with the information stated in the question, there is no way to check whether the counter has recycled or not. This means that there could have been 13 clock pulses; the first 8 pulses bring the counter back to 000, and the last 5 bring it to 101. The other possibility could be that there have been 21 pulses (*i.e.* two complete cycles using 16 pulses and then five more), or 29 pulses (three complete cycles using 24 pulses and then five more) and so on.

So there could be several possible answers to this question like 5, 13, 21, 29, 37…

Example 7-2. *Assume that a five bit binary counter starts in the* 00000 *state. What will the counter state after* 182 *pulses*?

Solution. Given: The five bit binary counter state = 00000

We know that a five-bit binary counter will reset after every 32 pulses. Next it will reset after 64, 96, 128 and 160 pulses. After resetting at 160th pulse the count until 182th pulse will be the binary number equivalent to 182 – 160 = 22 i.e. 10110. **Ans.**

7-4. MOD Number of a Counter

The MOD number of a counter is also called modulus number. It is equal to the number of states that the counter goes through in each complete cycle before it recycles back to its starting state. Notice that the counter shown in Fig. 7-2 has 8 distinctly different states (000 through 111).Thus it is a MOD-8 ripple counter.

The MOD number can be increased simply by adding more flip-flops to the counter, i.e.

$$\text{MOD number} = 2^N$$

Where N is the number of flip-flops connected in the arrangement of Fig. 7-2.

Example 7-3. *What would be the MOD number of the counter be if three more flip-flops were added to the counter circuit shown in Fig.* 7-2?

Solution. Given : A counter with 3 flip-flops connected as shown in Fig. 7-2

We know that this counter can count from 0 through 7. So it is a MOD-8 counter. If we add three more flip-flops, the total number of flip-flops would be equal to 6. Therefore, the

$$\text{MOD number} = 2^6$$
$$= 64 \quad \textbf{Ans.}$$

Example 7-4. *Fig.* 7-4 *shows the basic steps involved in building a digital clock. As seen, the first step is to take a* 50-*Hz signal and feed it into a pulse-shaping circuit (i.e. Schmitt- trigger). This produces a* 50-*Hz square wave which is then put into a MOD*-50 *counter. This counter divides the* 50-*Hz frequency by exactly* 50 *to produce a* 1-*Hz waveform. This* 1-*Hz waveform is fed to a series of counters, which then count seconds, minutes, hours and so on. Determine the number of flip-flops required for the mod*-50 *counter.*

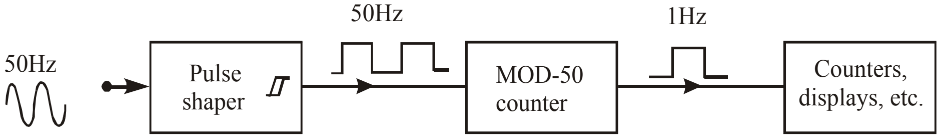


Fig. 7-4.

Solution. Given : A MOD-50 counter.

We know that there is no integer power of 2 that will equal 50. The closest is $2^6 = 64$

∴ No. of flip-flops required for the MOD-50 counter = 6 **Ans**.

Note. As a matter of fact, a counter with six flip-flops would act as a MOD–64 counter. This does not satisfy the requirement for a MOD-50 counter. It seems that there is no solution to this problem. However, we will discuss in the following pages, that it is possible to modify a binary counter so that virtually any MOD number can be obtained. In other words, the count is not limited to the value of 2^N.

Example 7-5. *A grocery retail outlet entrance door needs a counter that will count the number of customers who come for shopping each day. A photodetector and a light source combination is used to generate a single pulse, each time a person enters the door. The counter must be able to count as many as* 1000 *persons. How many flip-flops are required to construct the counter*?

Solution. Given the maximum number of persons that a counter must count = 1000.

In order to construct a counter that must count up to 1000, we need a counter that goes through at least 1000 states i.e.,

$$2^N \geq 1000$$

or $$N \geq 10$$

This means the counter will have at least 10 flip-flops. Of course we can have the number of flip-flops higher than 10 but it will be a waste of flip-flops because any flip-flop past the tenth one will not be needed. **Ans.**

7-5. Frequency Division

We have already discussed in Art. 5-30 (chapter 5) that in the basic counter, each flip-flop provides an output waveform that is exactly half the frequency of the waveform at its CLK input. In order to illustrate the point, suppose that the clock signal in Fig. 7-2 is 8 kHz. Fig 7-5 shows the flip-flop output waveforms. The waveform at output A is a 4 kHz squarewave, at output B is 2 kHz, at output C, it is 1 kHz. Notice that the output of flip-flop C has a frequency equal to the original clock frequency divided by 8. In general, in any counter, the signal at the output of the last flip-flop (i.e. most-significant bit) will have a frequency equal to the input clock frequency divided by the MOD number of the counter. In the present case, frequency at the C output.

$$(f_{\text{output}})_C = \frac{f_{\text{input}}}{\text{MOD Number}}$$

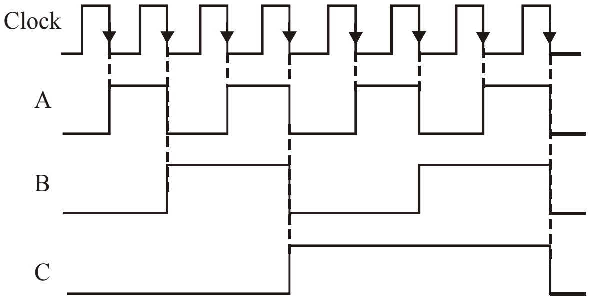


Fig. 7-5.

For example, in a MOD- 8 counter, the output from the last flip-flop will have a frequency of 1/ 8 of the input clock frequency. Thus it can also be called a divide-by- 8 counter. Likewise, a MOD-16 counter has an output frequency of 1/16 the input frequency, it is a divide by-16 counter.

Note.

1. Usually the duty cycle of the clock input is 50 percent. However if it is anywhere between 0 and 100 percent, (i.e 0 < duty cycle < 100) the duty cycle of the frequency output at A,B and C will always be 50 percent.

Example 7-6. *A binary counter is being pulsed by a* 512 *kHz clock signal. The output frequency from the last flip-flop is* 4 *kHz. Determine* (*a*) *the MOD number and* (*b*) *the counting range.*

Solution. Given, frequency of the clock signal = 256 kHz = 256 × 10^3 Hz and output frequency = 2 kHz = 2 × 10^3 Hz.

(*a*) **MOD Number**

We know that MOD number

$$= \frac{\text{Frequency of the Input clock signal}}{\text{Output frequency}}$$

$$= \frac{512 \times 10^3}{4 \times 10^3} = 128 \textbf{ Ans.}$$

(*b*) **Counting range**

We know that the MOD number,

$$128 = 2^N \quad \text{... (N is the number of flip-flops)}$$

and the maximum count,

$$= 2^N - 1$$

$$= 128 - 1 \quad \text{.... } (\because 2^N = 128)$$

$$= 127$$

Thus the counting range is 0 to 127 **Ans.**

Example 7-7. Construct a binary counter that will convert a 64 kHz signal into a 2 kHz square wave.

Solution. Given: A 64 kHz clock signal.

We know that in order to get a 2 kHz square wave from a 64-kHz signal, we need a binary counter which divides the input frequency by a factor of 64 kHz / 2 kHz = 32.

Thus MOD number of the required counter has to be 32.This is possible to achieve using five flip-flops (because $2^5 = 32$) connected together as shown in Fig 7-6.

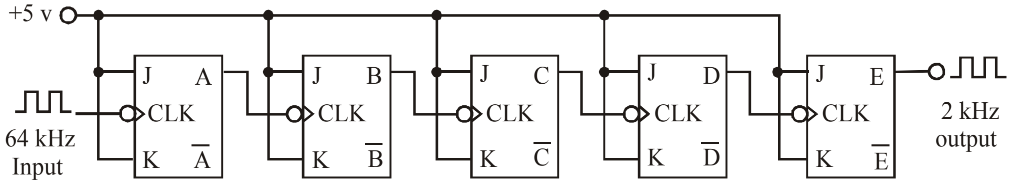


Fig. 7-6.

As shown in the diagram, the output E has a 2 kHz square wave. Notice that all the JK flip-flops are shown to be triggered at the falling edge of the input signal. These can also be indicated as being triggered at the rising edge of the input signal.

Example 7-8. *Fig.* 7-7 *shows a five-bit asynchronous counter. If the clock input frequency is* 12 *MHz.*

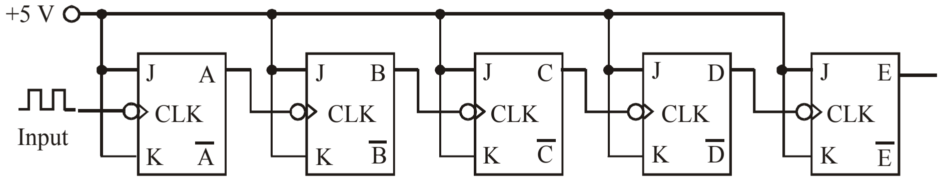


Fig. 7-7.

(*a*) *Determine the MOD number of the counter and the frequency at the E output. What is the duty cycle of this signal.*

(*b*) *Repeat part* (*a*) *if the duty cycle of the clock signal is* 25 *percent.*

(*c*) *Determine the frequency at the C output.*

Solution. Given: A five-bit asynchronous (or ripple counter) with clock input frequency $f_{input} = 12\text{ MHz} = 12 \times 10^6$ Hz

(*a*) **Frequency at the E output.**

Let $(f_{output})_E$ = The frequency at the E output

We know that the MOD number of a counter.

$$\text{MOD number} = 2^N$$

Substituting the value of N = 5 (for a 5-bit counter), we find that,

MOD number = 2^5 = 32 **Ans.**

Now we know that, the frequency at the E output

$$\therefore \quad (f_{output})_E = \frac{f_{input}}{\text{MOD Number}} = \frac{12 \times 10^6}{32} = 375{,}000$$

$$= 375 \text{ kHz}$$ **Ans.**

The duty cycle of the output signal is the same as that of the input signal, i.e. 50 %

(*b*) **Duty cycle at the E output if duty cycle of clock signal is 25 percent.**

If the duty cycle of the clock signal is reduced to 25 %, there will be no change in the f_{output} at E. The duty cycle at E output will still be 50 % **Ans.**

(*c*) **Frequency at the C output.**

Let $(f_{output})_C$ = The frequency at the C output,

We know that the frequency at the C output

$$(f_{output})_C = \frac{f_{input}}{\text{MOD Number}}$$

The MOD number in this situation has to be obtained for 3 flip-flops only (i.e. flip-flops A, B and C). Its value is 8.

$$\therefore \quad (f_{output})_C = \frac{12 \times 10^6}{8} = 1.5 \times 10^6 = 1.5 \text{ MHz}$$ **Ans.**

7-6. Counter with MOD Number Less than 2^N

The basic asynchronous counter of Fig 7-2 is limited to MOD numbers that is equal to 2^N where N is the number of flip-flops. This value is actually the maximum MOD number that can be obtained using N flip-flops. The basic counter can be modified to produce MOD numbers less than 2^N by allowing the counter to skip states that are normally part of the counting sequence.

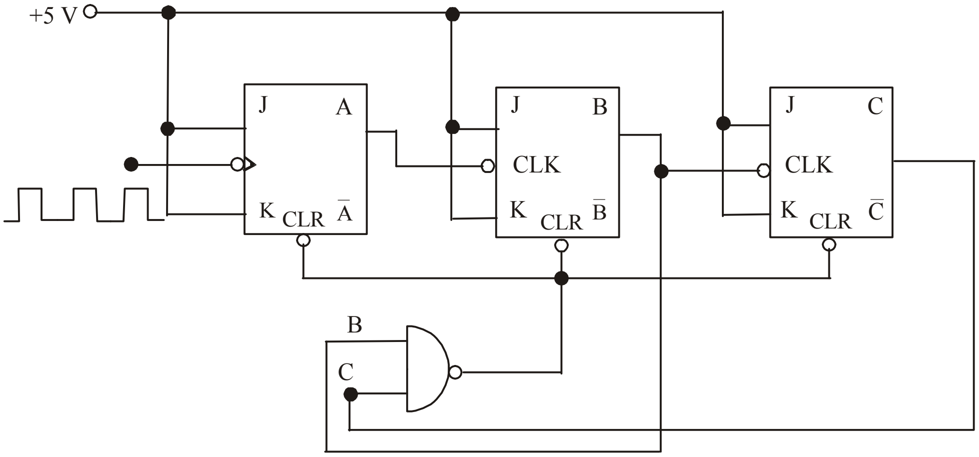


Fig. 7-8. MOD-6 Counter

One of the most common methods for doing this is illustrated in Fig 7-8 where a three-bit counter is shown. Disregarding the NAND gate for a moment, we can see that the counter is a MOD-8 binary counter which will count in sequence from 000 to 111. However, the presence of NAND gate will alter this sequence as follows:

1. The NAND output is connected to the asynchronous CLEAR inputs of each flip-flop. As long as the NAND output is HIGH, it will have no effect on the counter. However when it goes low, it will clear all of the flip-flops so that the counter immediately goes to the 000 state.

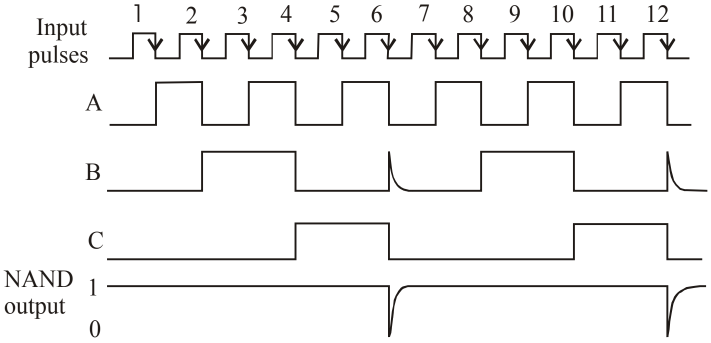
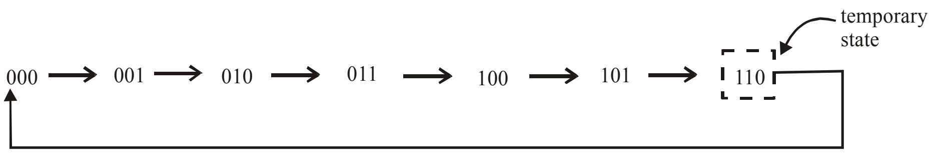


Fig. 7-9. Illustrating the operation of MOD-6 counter

2. The inputs to the NAND gate are the outputs of the B and C flip-flops and so the NAND output will go low whenever B = C = 1. This condition will occur when the counter goes from the 101state to 110 state on the falling edge of input pulse 6 (refer to Fig. 7-9). The LOW at the NAND output will immediately (generally within a few nanoseconds) clear the counter to the 000 state. Once the flip-flops have been cleared, the NAND output goes back HIGH, since B = C = 1 condition no longer exists.
3. Therefore the counting sequence is:



It may be noted that although the counter does go to the 110 state, it remains there for only a few nanoseconds before it recycles to 000.Thus we can say that this counter counts from 000 (zero) to 101 (five) and then recycles to 000. In other words it skips 110 and 111 so that it goes through only six different states-thus it is a MOD-6 counter.

Example 7-9. *Fig* 7-10 *shows the block diagram of a four-bit ripple counter.*

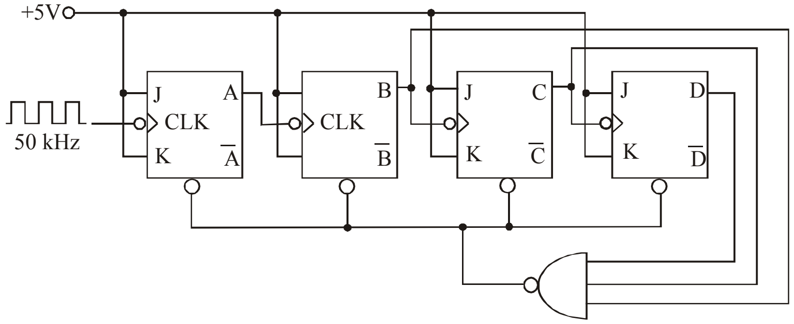


Fig. 7-10.

Determine the MOD number of the counter. Also determine the frequency output at the D output.

Solution. Given : A four-bit ripple counter.

We know that a four-bit ripple counter would normally count from 0000 through 1111. But notice the presence of a NAND gate with inputs D, C and B. This means that the counter will immediately recycle to 0000 when the 1110 (i.e. decimal 14) count is reached. Thus the counter actually has 14 stable states 0000 through 1101 and is therefore a MOD –14 counter.

Further since the clock input frequency (f_{in}) = 50 kHz, therefore the frequency at output, D.

$$(f_{out})_D = \frac{50\,\text{kHZ}}{14} = 3.57 \text{ kHz } \textbf{Ans.}$$

Example 7-10. *Design a MOD-*2 *and MOD-*6 *counter and use it to construct a MOD-*12 *counter.*

(*MOD-*6 *counter must progress through* 000 ⟶ 001 ⟶ 010 ⟶ 011 ⟶ 100 ⟶ 101 ⟶ 000)

(*Nagpur University Mar/April 1998*)

Solution. We have already seen the design of a MOD-6 counter in Art 7.6. A Mod –2 counter is basically just one J-K flip-flop. Thus a MOD-6 counter and a MOD-2 counter can be constructed as shown in Fig 7-11. The output of MOD-6 counter can be connected to MOD-2 counter to obtain a MOD-12 counter.

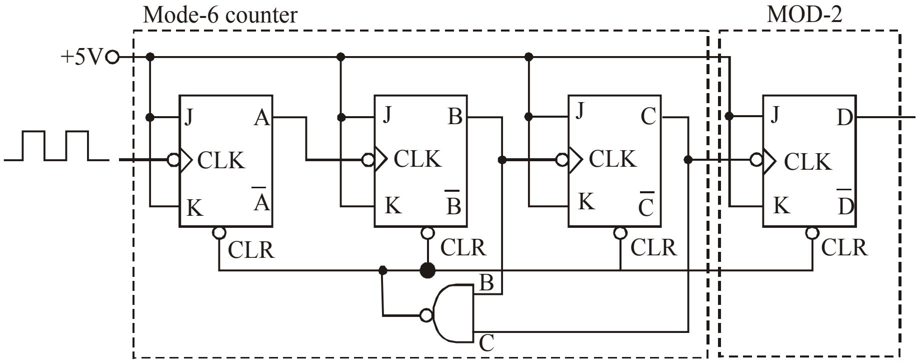


Fig. 7-11.

A MOD-12 counter means if the frequency of the clock input of the flip-flop A is (say) 12 kHz, then the output frequency of the D flip-flop would be 1 kHz.

Example 7-11. *Use J-K flip-flops and any other necessary logic to construct a MOD-*24 *asynchronous counter.*

Solution. Given : The MOD number of an asynchronous counter = 24.

We know that to construct an asynchronous counter with MOD number of 24, we need five J-K flip-flops connected as shown in Fig 7-12.

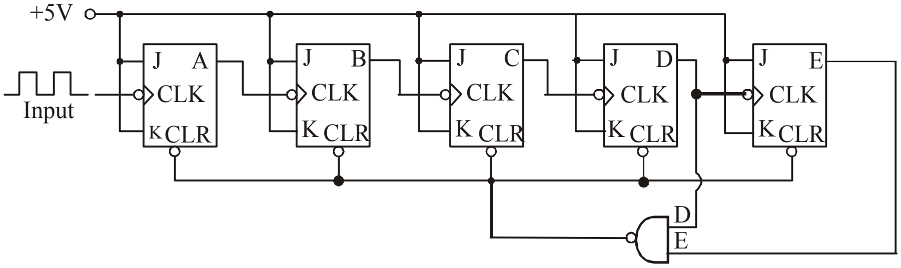
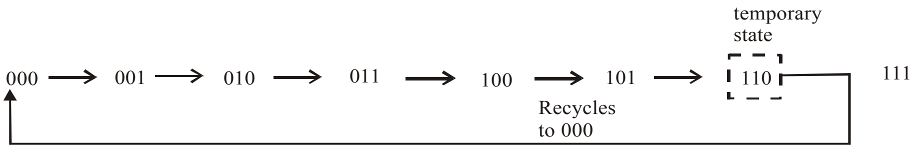


Fig. 7-12.

This counter can counter from 0 to 31. But we need a counter that could count from 0 to 23 only. In order to achieve this we have to clear the counter, the moment it reaches 24 (i.e. 11000). This is indicated by using a NAND gate with the two inputs taken from the outputs of D and E flip-flop. The output of the NAND gate is shown connected to the CLR inputs of J-K flip-flops.

7-7. State Transition Diagram of a Counter

We have already seen in the last article that each time the MOD-6 counter received a clock pulse, it goes through different states. These can be represented as.



Another systematic way of representing the different counter states is through a state transition diagram. Fig 7-13 shows such a diagram for a MOD-6 counter. Here each circle represents one of the possible counter states and the arrow indicates how one state changes to another one in response to the clock pulse.

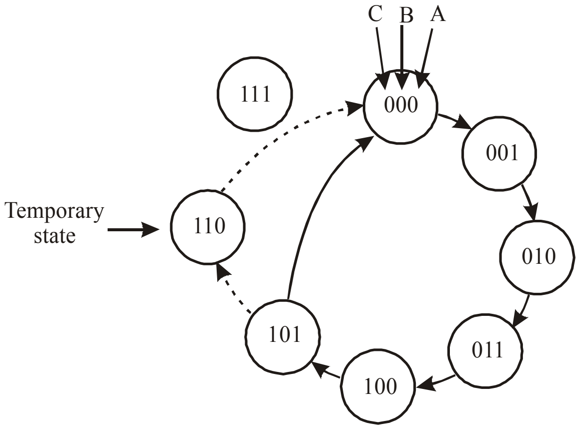


Fig. 7-13. State transition diagram of a MCD-6 counter

Let us assume the starting count of 000. Then the state transition diagram shows that the states of the counter change normally up until the count of 101. When the next clock pulse occurs, the counter temporarily goes to the 110 count before going to a stable 000 count. The dotted lines in the diagram indicate the temporary nature of 110 state. As mentioned earlier, the duration of this temporary state is so short that for most practical purposes we can consider that the counter goes directly from 101 to 000. This is represented by a solid arrow linking 101 to 000.

It may be noted that there is no arrow into the 111 state because the MOD-6 counter can never go to that state. However, the 111 state can occur on power-up when the flip-flops come up in random states. If that happens, the 111 condition will produce a LOW at the NAND gate output and immediately clear the counter to 000. Thus the 111 state is also a temporary condition that ends up at 000.

7-8. Displaying States of a Counter

Sometimes it is necessary to have a visible display of how a counter is changing states in response to the input pulses. There are several ways of doing this as we will study later in the chapter. Fig. 7-14 shows a simple method of displaying the counter states using individual LEDs for each flip-flop output. As seen from the figure, each flip-flop output is connected to an INVERTER whose output provides the current path for the LED. For example when output B is HIGH, the INVERTER output goes LOW and the LED turns ON. The bright LED indicates B = 1. When output B is LOW, the INVERTER output is HIGH and the LED turns OFF. The dark LED indicates B = 0.

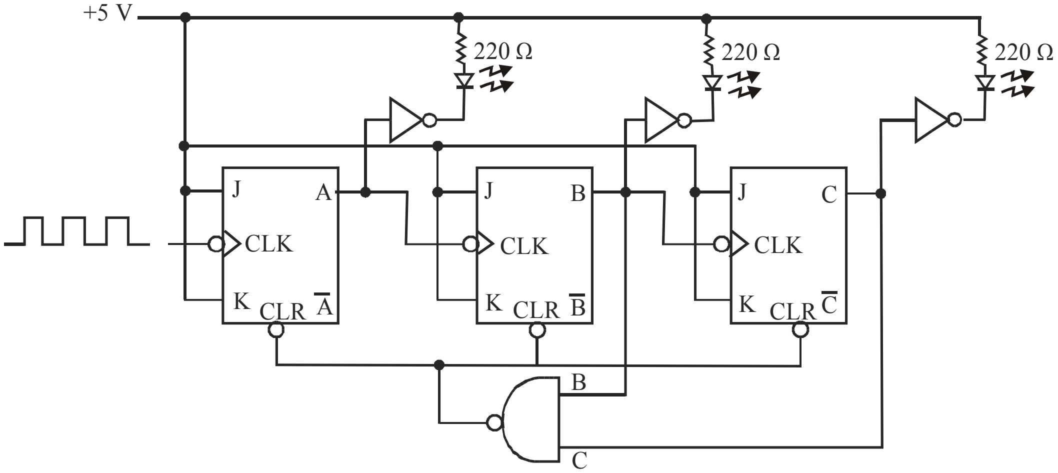


Fig. 7-14. Displaying counter states

Note. Another method of displaying the counter state is to connect the LED cathodes directly to the INVERTED outputs of the flip-flop, A, B, and C. The operations of the circuit will still remain the same.

7-9. Changing the MOD Number of a Counter

We have already in seen Art 7-6 as how a MOD-6 counter can be obtained by using a NAND gate with a MOD-8 counter. In general, any desired MOD number can be obtained by changing the NAND gate inputs. For example, using a three-input NAND gate with inputs A, B, and C, the counter would function normally until the 111 condition was reached. At 111 condition, the counter would immediately reset to 000 state. If we ignore the temporary excursion of the counter to the 111 state, it would count from 000 through 110 and then recycle back to 000. This results in a MOD-7 counter.

Followings are the steps to construct a counter that starts counting from all 0s and has a MOD number of X :

1. Find the smallest number of flip-flops such that $2^N \geq$ X and connect them as a counter. If $2^N =$ X, there is no need to do steps 2 and 3.
2. Connect a NAND gate output to the asynchronous CLEAR inputs of all the flip-flops.
3. Find which flip-flops will be in the HIGH state at a count = X, then connect the normal outputs of these flip-flops to the NAND gate inputs.

Example 7-12. (*a*) *How many flip-flops are required to build a binary counter that counts from* 0 *to* 1023?

(*b*) *Determine the frequency at the output of the last flip-flop of this counter for an input clock frequency of* 4 *MHz.*

(*c*) *What is the counter's MOD number*?

(*d*) *If the counter is initially at zero, what count will it hold after* 2060 *pulses.*

Solution. (*a*) Given : A binary counter that counts from 0 to 1023.

We know that a binary counter that counts from 0 to 1023, has 1024 states. Therefore its MOD number is 1024.

Now let N = Number of flip-flops, then we know that, the MOD number,

$$1024 = 2^N \text{ or } N = 10 \textbf{ Ans.}$$

(b) **Frequency at the output of last flip-flop.** Given: The input clock frequency, $f_{in} = 4\text{ MHz} = 4 \times 10^6\text{Hz}$.

Let f_{out} = output frequency

We know that the MOD number,

$$1024 = \frac{f_{in}}{f_{out}} = \frac{4\times10^6}{f_{out}}$$

$$\therefore \qquad f_{out} = \frac{4\times10^6}{1024} = 3906.25\text{ Hz} = 3.906\text{ kHz} \textbf{ Ans.}$$

Example 7-13. *A digital clock requires a MOD-*50 *counter to divide the* 50 *Hz line frequency down to* 1*Hz. Construct an appropriate MOD-*50 *counter.*

Solution. We know that a five-bit counter up to 2^5 (equals 32) and a six-bit counter can count up to 2^6 (equals 64). Thus to count until 50, we need a six-bit counter as shown in fig 7-15. The counter is to be cleared when it reaches 50 (i.e. 110010). Thus the outputs of flip-flops Q_5, Q_4 and Q_1 must be connected to the NAND gate. The output of flip-flop '5', Q_5 will have a frequency of 1 Hz.

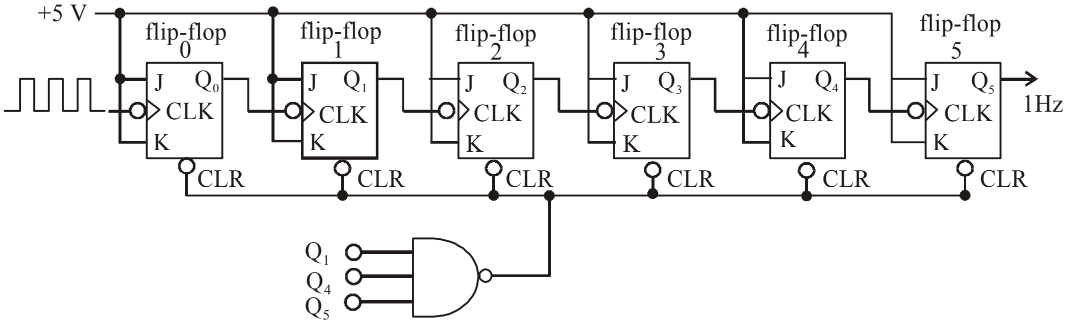


Fig. 7-15.

Example 7-14. (*a*) *Construct a MOD-*10 *counter that will count from* 0000 (*i.e. zero*) *through* 1001 (*i.e. decimal* 9)

(*b*) *Determine the frequency output if the frequency input is* 100 *kHz.*

Solution. (*a*) Recall the steps given in Art 7-9 to construct a counter with any MOD number X.

The smallest number of flip-flops required to construct a MOD-10 counter can be obtained by using the equation,

$$2^N \geq 10 \text{ which gives } N = 4$$

Notice that with N = 3, the counter will count only up to 8 and with N = 4 it can count up to 16.

Since the MOD-10 counter is to have stable operation up to the count of 1001, it must be reset to zero when the count of 1010 is reached. Therefore flip-flop outputs D and B must be connected to the NAND gate inputs. Fig 7-16 shows the arrangement of a MOD-10 counter.

(*b*) If the input frequency is 100 kHz, then

$$\text{the output frequency} = \frac{100\text{ kHz}}{10} = 10\text{ kHz} \textbf{ Ans.}$$

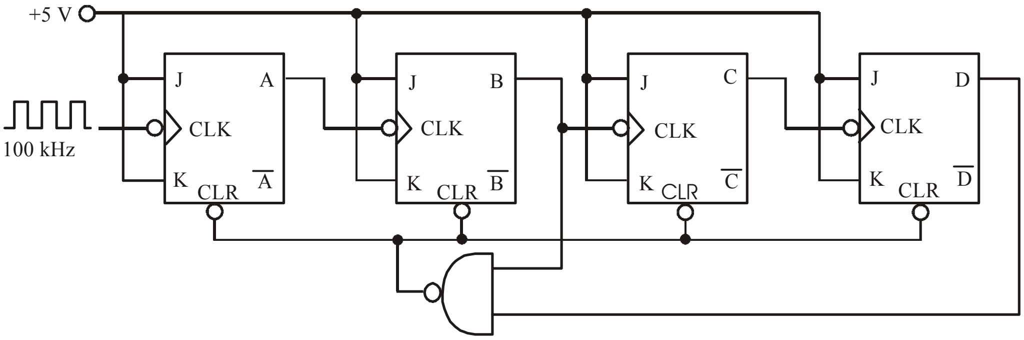


Fig. 7-16.

7-10. Decade (or BCD) Counters

The MOD-10 counter discussed in the last article is also known as *decade counter*. As a matter of fact, a decade counter is any counter that has 10 distinct states, no matter what the sequence is. A decade counter counts in sequence from 0000 (zero) through 1001 (decimal 9) is also referred to as a *BCD counter*. It is called BCD counter because it uses only the 10 BCD code groups 0000, 0001,….., 1000 and 1001. Thus in general, any MOD-10 counter is a decade counter and any decade counter that counts in binary from 0000 to 1001 is a BCD counter.

Decade counters, especially the BCD type are widely used in applications where pulses or events are to be counted and the results displayed on some type of decimal numerical readout. A decade counter is also often used for dividing a pulse frequency exactly by 10.

Example 7-15. *A decade counter is clocked from a* 60 *kHz signal. Determine its output frequency.*

Solution. Given: A decade counter clocked from a 60 kHz signal (refer to Fig 7-17).

We know that a decade counter divides the input frequency by 10, therefore the output frequency,

$$f_{out} = \frac{f_{in}}{10} = \frac{60 \text{ kHz}}{10}$$

$$= 6 \text{ kHZ } \textbf{Ans.}$$

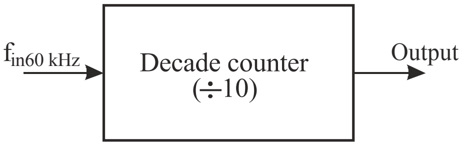


Fig. 7-17.

7-11. Integrad-Circuit Asynchronous Counters

There are several TTL and CMOS asynchronous (or ripple) counter ICs available in the market. Table 7-1 shows some of the popular 4-bit ripple counters with there features and common application.

Table 7-1. 4-bit IC ripple counters

	Device	*Features*	*Application*
1	7490	4-bit ripple counter consisting of divide-by-2 section and a divide-by-5 section. The two sections can be cascaded together to form a divide-by-10 (MOD-10) counter	used as a BCD counter
2	7492	4-bit ripple counter consisting of divide-by-2 section and divide-by-6 section. The two sections can be cascaded together to form a divide-by-12 (MOD-12) counter.	used in digital clocks
3	7493/74293	4-bit ripple counter consisting of divide-by-2 and divide-by-8 section. The two sections can be cascaded together to form a divide-by-16 (MOD-16) counter.	used dasa counter with any MOD number ≤ 16

The pin configuration of 7490, 7492 and 7493 are given in Appendix C (C-2). We shall discuss in detail about IC 74293 and its applications.

Fig. 7-18 shows the logic diagram for the 74293 A as it would appear in manufacturer's TTL data book. Following are some of the points about 74293 A which are important from the subject point of view.

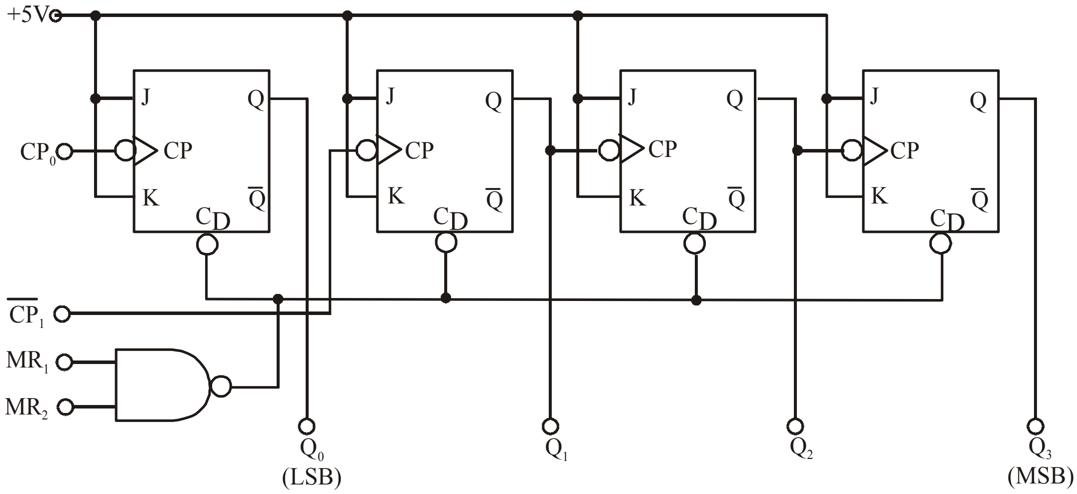


Fig. 7-18. Logic diagram of 74293A

1. The IC 74293A has four J-K flip-flops with outputs Q_0, Q_1, Q_2 and Q_3 where Q_0 is the least-significant bit (LSB) and Q_3 is the most significant bit (MSB).
2. Each flip-flop has a clock pulse (CP) input. This is just another name for the CLK input. The clock inputs to the Q_0 and Q_1 labelled $\overline{CP}_0$ and $\overline{CP}_1$ respectively are externally accessible.

The bars over these inputs indicate that they are activated by the falling edge of the input signal.

3. Each flip-flop has an asynchronous active-LOW CLEAR input. These are connected together to the output of a two- input NAND gate with inputs MR_1 and MR_2. The MR stands for master reset. Both MR_1 and MR_2 inputs must be HIGH to clear the counter to 0000.
4. The flip-flops with outputs Q_1, Q_2, and Q_3 are already connected as a three-bit ripple counter. The flip-flop with output Q_0 is **not** connected internally. This allows the user the option of either connecting Q_0 to Q_1 to form a four-bit counter or using Q_0 separately if desired. In other words the device can be used as a divide by a 2 counter if only single flip-flop output Q_0 is used. Alternatively it can be used as a MOD-8 counter if 3-bit counter portion is used. Fig 7-19 shows a simplified logic symbol of the 74293. Notice that V_{CC} supply is at pin 5 and GND is at pin 10.

The 74293A can be wired to produce counters with different MOD numbers. This is illustrated through the examples in the following pages.

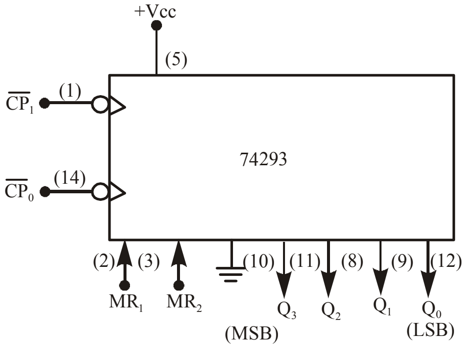


Fig. 7-19. A simplified logic symbol for 74293 Counter

Example 7-16. *Using the logic symbol of* 74293, *shown in Fig.* 7-20 *show how it can be used as a MOD*-16 *counter.*

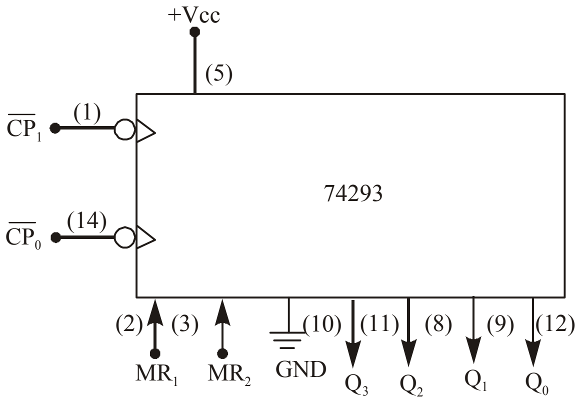


Fig. 7-20.

If the input clock frequency is 32 *kHz, what will be the frequency at* Q_3 ?.

Solution. (*a*) Given : The logic symbol of 74LS293A counter

We know that 74293 has four flip-flops. Refer to the logic diagram shown in Fig. 7-18. In order to connect it as a MOD –16 counter, we must connect the Q_0 output to $\overline{CP_1}$ input (i.e. the clock-input of flip-flop Q_1) as shown in Fig 7-21. The 32 kHz input pulses are applied to $\overline{CP}_0$ (i.e. the clock input of Q_0). The output at Q_3 will have a frequency 1/16 of the clock input frequency, i.e. 32 kHz/16 = 2 kHz **Ans.**

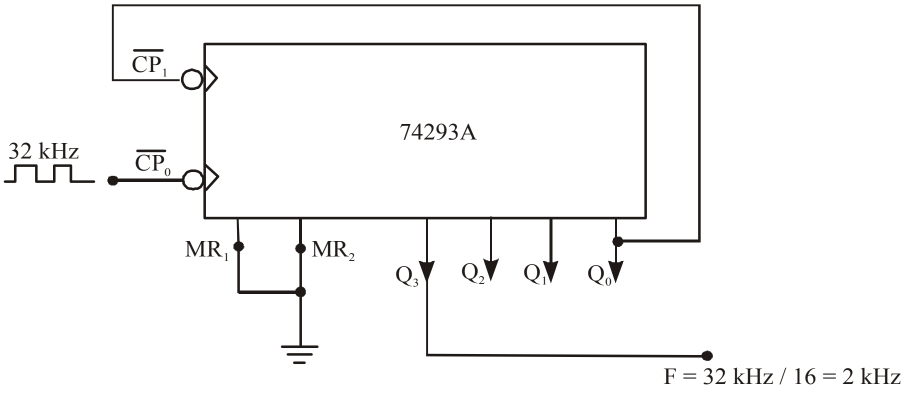
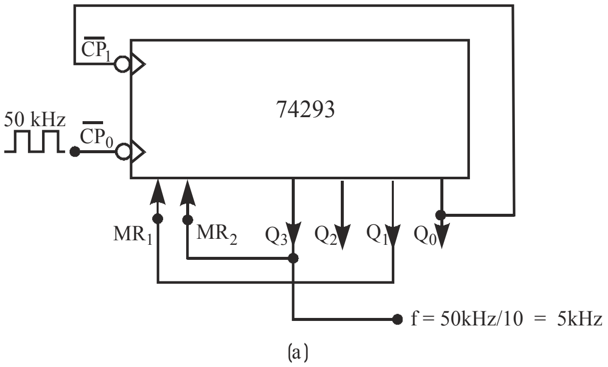


Fig. 7-21. Illustrating the wiring of 74293 as a MOD-16 counter

Example 7-17. *Using the logic symbol of* 74293 *shown in Fig* 7-20 *show how it can be wired as a MOD-*10 *counter. Determine the Q_3 output frequency if the input signal frequency is* 50 *kHz.*

Solution. Given: The logic symbol of 74293 counter.

We know that MOD-10 counter requires four flip-flops. So we need to connect Q_0 to $\overline{CP}_1$. Since we want the counter to recycle back to 0000 when it tries to go to the count of 1010 (i.e. 10). Thus the Q_3 and Q_1 outputs must be connected to the master reset (MR) inputs. When they both go HIGH at the count of 1010, the NAND gate output will immediately reset the counter to 0000.



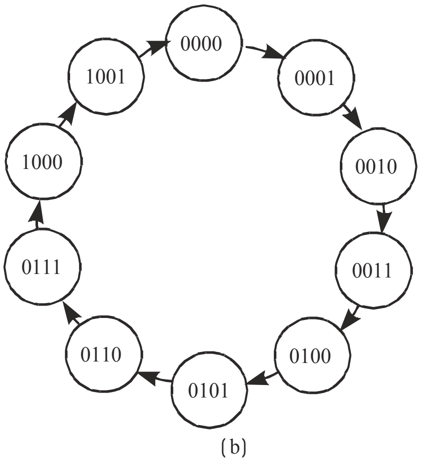


Fig. 7-22

Fig. 7-22 (a) shows the wiring of 74293 as a MOD-10 counter. The state transition diagram is shown in Fig. 7-22 (b). Notice carefully that the temporary 1010 state is not indicated in the figure. Further the states 1011, 1100, 1101, 1110 and 1111 are not indicated because these are not part of the normal operation of the counter. However, there could be a possibility that a counter could come up in one of these states at power up. But note that the moment $Q_3 = Q_1 = 1$, the counter will reset automatically. If the input signal frequency is 50 KHz, then the Q_3 output frequency, 50 kHz /10 = 5 kHz **Ans.**

Example 7-18. *Fig.* 7-23 *shows the logic symbol of*74293.

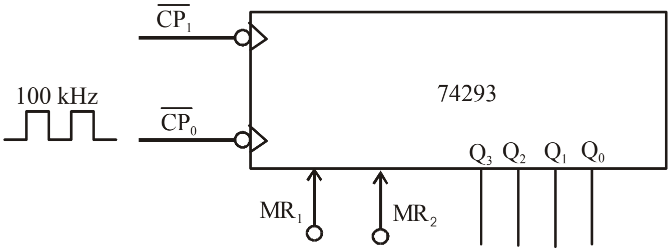


Fig. 7-23.

*Using the logic symbol, show how it can be wired as a MOD-*14 *counter. Determine the Q_3 output frequency if the input signal frequency is* 100 *kHz.*

Solution. Given : The 74293 counter.

We know that MOD-14 counter requires four flip-flops. So we need to connect Q_0 to $\overline{CP1}$. Further for MOD-14 counter, we want this counter to recycle back to 0000 when it tries to go to the count of 1110 (i.e. 14). When the counter reaches at 1110, the Q_3, Q_2 and Q_1 outputs are all HIGH. But the 74293 's built in reset NAND gate has only two inputs. Thus we must add extra logic in order to make sure that the counter will reset to 0000 when $Q_3 = Q_2 = Q_1 = 1$. One possible solution is to connect one of the outputs (say Q_3) directly to one of the master reset inputs (say MR_2) and the other two outputs Q_1 and Q_3 are connected to the two inputs of an external AND gate. The output of the AND gate is then connected to MR_1 input. Fig. 7-24 shows the connection of the external AND gate and the other necessary connections to wire the 74293 as a MOD-14 counter.

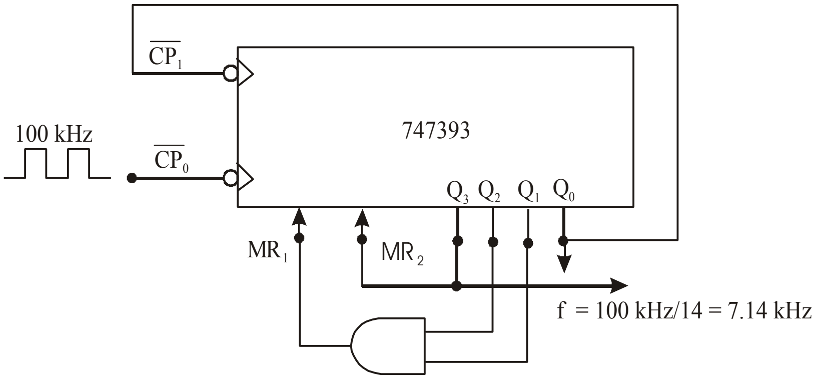


Fig. 7-24.

Example 7-19. *Fig* 7-25 *shows two* 74293*'s combined to provide a frequency division of* 50 *by successive divisions by* 10 *(MOD-*10*) and* 5 *(MOD-*5*). Explain the functioning of this circuit.*

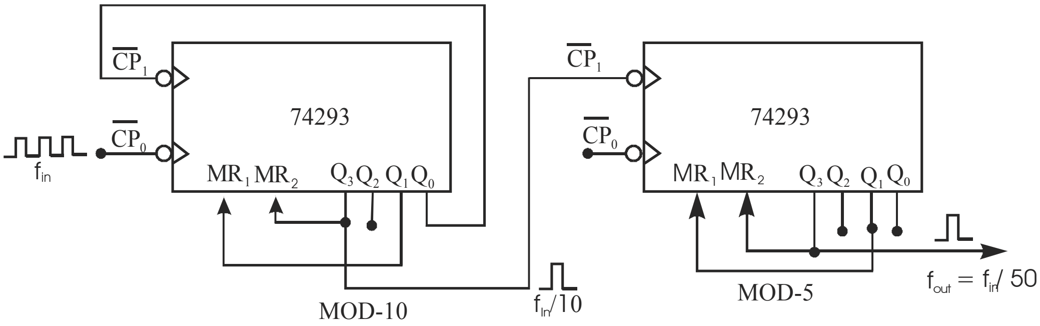


Fig. 7-25.

Solution. Given: A circuit with two 74293's to divide the input frequency by 50.

As seen from the Fig. 7-25, the circuit divides the input frequency by 50 in two steps. The 74293 counter on the left is connected as a MOD-10 counter. Its output Q_3 has a frequency of $f_{in}/10$. This signal is connected to $\overline{CP_1}$ input of the second 74LS293A counter. This counter is connected as a MOD-5 counter (Notice carefully that Q_0 of the 74LS293A counter on the right is not used). Thus the Q_3 of the second counter will have a frequency output,

$$f_{out} = \frac{f_{in}/10}{5} = \frac{f_{in}}{50}$$

Thus if f_{in} = 50 Hz, f_{out} = 50/50 = 1 Hz. This design approach can be used to count seconds, minutes and hours.

7-12. Asynchronous Down Counter

We have already discussed in the previous articles about the 3 and 4-bit asynchronous counter that counts from 000 (or 0000) to 111 (or 1111) respectively. Such a counter is known as *up counter*. It is possible to construct asynchronous (or ripple) down counter i.e. a counter that counts down from a maximum count to zero.

Fig . 7-26 shows the illustration of a 3-bit down counter. As seen from this figure, the input pulses are applied to the A flip-flop. The $\overline{A}$ output serves as the CLK input for the B flip-flop. Similarly the $\overline{B}$ output serves as the CLK input of the C flip-flop.

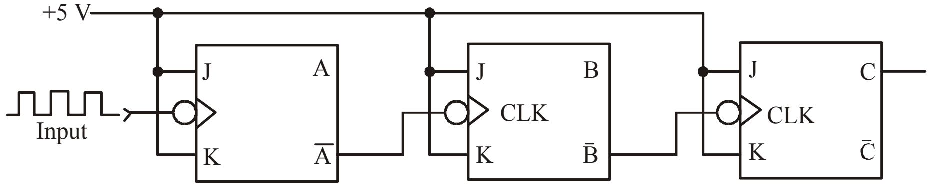


Fig. 7-26. Illustration of MOD-8 down counter

Fig. 7-27 shows the waveforms at A, B, and C outputs of the counter. As seen from the diagram, B toggles whenever A goes LOW to HIGH (alternatively $\overline{A}$ goes HIGH to LOW). Similarly C toggles whenever B goes LOW to HIGH (alternatively $\overline{B}$ goes HIGH to LOW). This results in the desired count-down sequence at the three outputs (C, B and A). Notice that as the counter starts from 000 state, it goes to 111 state.

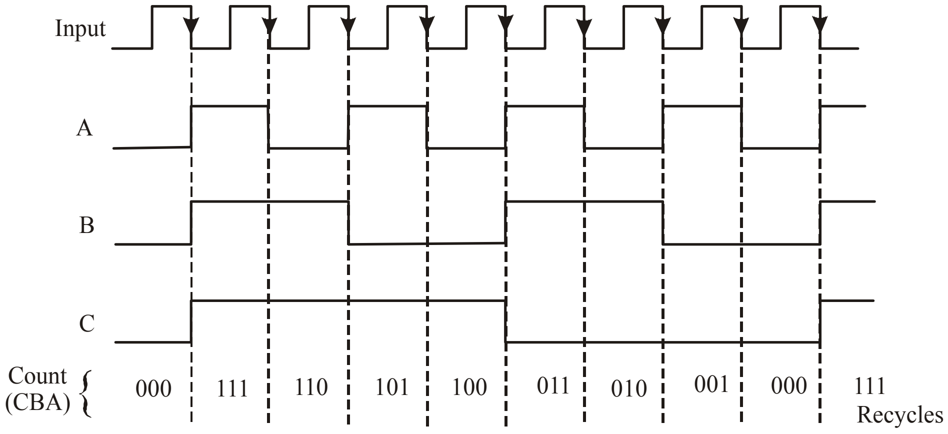
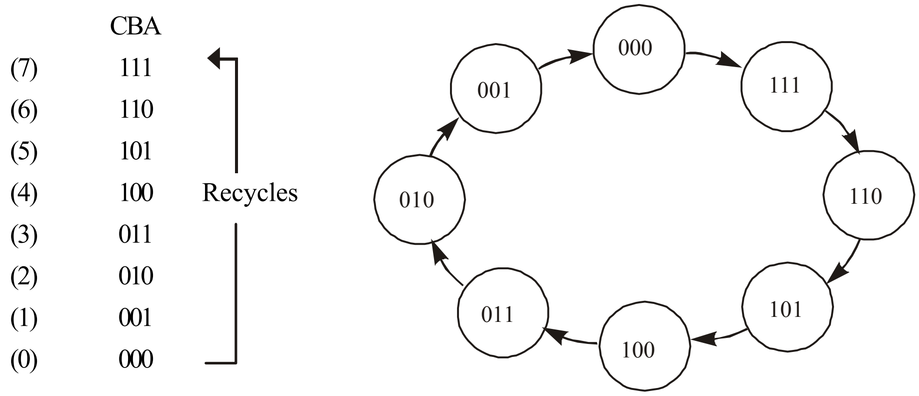


Fig. 7-27. Waveforms at A, B and C outputs of the 3-bit down counter

From there onwards it counts down in accordance with the sequence as shown in Fig. 7-28(a). The numbers in the brackets in the first column is the decimal number while the second column shows the binary equivalent.



	CBA
(7)	111
(6)	110
(5)	101
(4)	100
(3)	011
(2)	010
(1)	001
(0)	000

(*a*) Tabular diagram (*b*) State-transition diagram

Fig. 7-28. Illustrating the count-down sequence

The count down sequence can also be illustrated in the form of a state transition diagram as shown in Fig 7-28(b). The down sequence can easily be examined from this figure.

Applications

It may be carefully noted that down counters are not as widely used as up counters. Their major application is in situations when it must be known when a desired number of input pulses have occurred. In these situations, the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state, it is detected by a logic gate whose output then indicates that the preset number of pulses have occurred.

7-13. Propagation Delay in Asynchronous Counters

As a matter of fact, asynchronous (ripple) counters are the simplest types of binary counters as they require the fewest components to produce a given counting operation. However, such counters have one major drawback. This drawback is caused by the counter's operation i.e. each flip-flop is triggered by the rising or falling edge at the output of the preceeding flip-flop. Since each flip-flop has some propagation delay (t_{pd}), therefore the second flip-flop will not respond until a time, t_{pd} after the first flip-flop receives an active clock edge. Similarly, the third flip-flop will not respond until a time equal to 2 × t_{pd} after that clock edge and so on. In other words, the propagation delays of the flip-flops accumulate so that the N^{th} flip-flop cannot change states until a time equal to N × t_{pd} after that clock edge occurs. Fig. 7-29 illustrates this situation with the help of waveforms of a 3-bit asynchronous counter.

As seen from this diagram, an input pulse occurs, after every 200 ns (i.e. the clock period, T = 200 ns). Further it is assumed that each flip-flop has a propagation delay of 10 ns (i.e. t_{pd} = 10 ns). It may be carefully noted that output of flip-flop A toggles 10 ns after the falling edge of each input pusle. Similarly B toggles 10 ns after A goes from 1 to 0 and C toggles 10 ns after B goes from 1 to 0. As a result of this, when the fourth input pulse falling edge occurs, the C output goes HIGH after a delay of 30 ns. In this type of situation, the counter does operate properly in the sense that the flip-flops do eventually get to their correct states, representing the binary count. However, the situation worsens if the input pulses are applied at a much higher frequency.

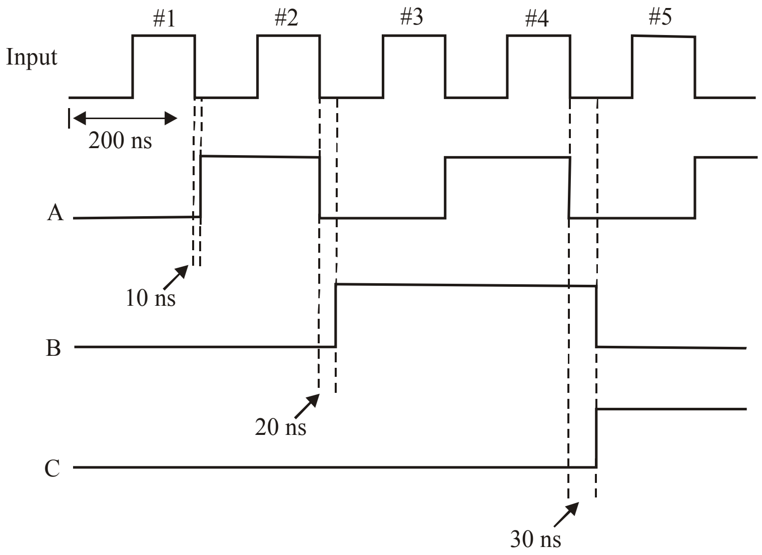


Fig. 7-29. Waveforms of a 3-bit asynchronous counter illustrating the propagation delay for 5 MHz input signal

Fig. 7-30 shows the waveforms for a input pulse that occurs after every 20 ns (i.e. the clock period is 20 ns.)

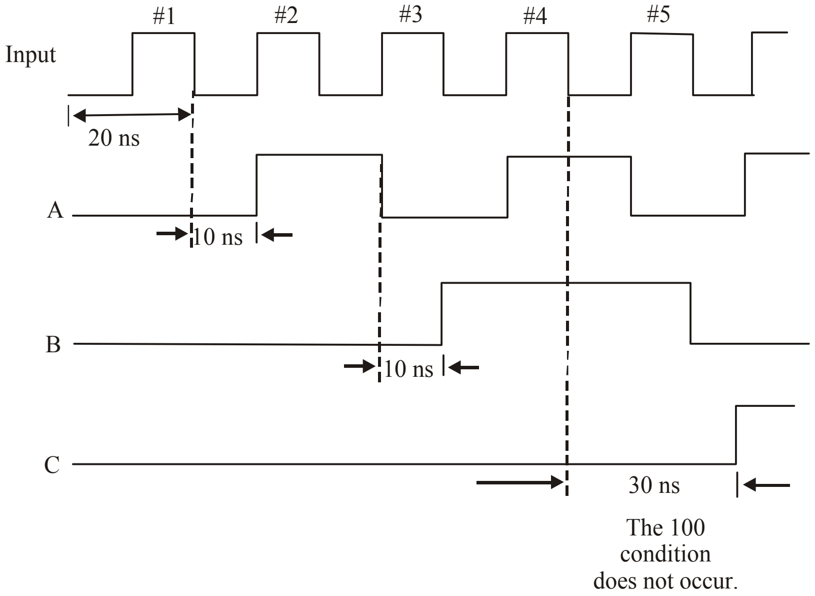


Fig. 7-30.

As seen from this diagram, each flip-flop responds 10 ns after the falling edge at its clock input. The change in relative time scale should be noted carefully. Examine the situation after the fourth clock pulse, where the C output does not go HIGH until 30 ns later which is the same time that the A output goes HIGH in response to the fifth input pulse. In other words, the condition C = 1, B = A = 0 (i.e. count of 100) never appears. This is because of the reason that input frequency is too HIGH. This condition could produce a serious problem if it were supposed to be used to control some other operation in a digital system. This type of problem can be avoided if the period between the input pulses is made longer than the total propagation delay of the counter. Mathematically, it can be represented by the equation,

$$T_{clock} \geq N \times t_{pd}\,. \qquad ...(i)$$

Where T_{clock} = period between the input pulses,

N = Number of flip-flops, and

t_{pd} = Propagation-delay of each flip-flop

The equation (i) can also be stated in terms of the input-clock frequency. Thus the maximum input-clock frequency is given by the equation.

$$f_{max} = \frac{1}{N \times t_{pd}}$$

Notes.

1. As a matter of fact, asynchronous counters are not useful at very high frequencies, especially for large number of bits.
2. There is another problem caused by propagation delays in a asynchronous counters. This problem occurs when the counter outputs are decoded. This problem is discussed in Art. 7-23.
3. Despite the problems discussed above, the simplicity of asynchronous counters makes them very useful for applications where frequency limitation is not critical.

Example 7-20. *A certain J-K flip-flop has t_{pd} = 12 ns. What is the largest MOD counter that can be constructed from these flip-flops and still operate up to 10 MHz.*

Solution. Given: The propagation delay of J-K flip-flop, t_{pd} = 12 ns = 12×10^{-9} s, Maximum clock frequency, f_{max} =10 MHz =10×10^6 Hz.

Let N = The number of flip-flops that can be connected to construct a counter with largest MOD number.

Then we know that the maximum clock frequency, (f_{max}),

$$10 \times 10^6 = \frac{1}{N \times t_{pd}} = \frac{1}{N \times 12 \times 10^{-9}}$$

$$\therefore \qquad N = \frac{1}{(10 \times 10^6) \times (12 \times 10^{-9})} = \frac{100}{12}$$

Thus the MOD number of a counter with 8 flip-flops,

MOD number = 2^8 = 256 **Ans.**

Example 7-21. *A four-bit asynchronous (ripple) counter is constructed using the 74LS112 J-K flip-flop. The 74LS112 has t_{PLH}= 16 ns and t_{PHL} = 24 ns as the propagation delays from CLK to Q. Determine the maximum input clock frequency for the counter.*

Solution. Given: The 74LS112 J-K flip-flop, Number of flip-flops, N = 4 t_{PLH} = 16 ns, and t_{PHL} = 24 ns.

Since both t_{PLH} and t_{PHL} represents the propagation delay (t_{pd}), we will consider the "worst case" i.e. t_{pd}= t_{PHL} = 24 ns = 24×10^{-9} s.

Now we know that maximum input clock pulses frequency,

$$f_{max} = \frac{1}{N \times t_{pd}} = \frac{1}{4 \times 24 \times 10^{-9}}$$

$$= 10.4 \times 10^{6}\ \text{Hz} = 10.4\ \text{MHz}.\ \textbf{Ans.}$$

7-14. Synchronous (or Parallel) Counters

We have already discussed in the last article about the problems encountered with asynchronous (or ripple counters). The problems in ripple counters are caused by propagation delays. It can also be stated in another way that the outputs of flip-flops in a ripple counter do not change states simultaneously in synchronism with the input pulses. This drawback can be overcome with the use of synchronous (or parallel) counters. In synchronous counters, all the flip-flops are triggered simultaneously (or in parallel) by the clock input pulses. Since the input pulses are applied to all the flip-flops, we must use some method to control when a flip-flop is to toggle and when it is to remain unaffected by a clock-pulse. This is achieved by using J and K inputs of the flip-flops.

7-15. A 4-bit Synchronous Binary Counters

Fig. 7-31 shows a four-bit MOD-16 synchronous binary counter As seen from this figure, the counter consists of four flip-flops and hence counts upto 2^4 (equals sixteen). Notice that the CLK inputs of all the flip-flops are connected together so that the input clock signal is applied to each flip-flop simultaneously.

Further, the J and K inputs of flip-flop A are tied permanently to + 5 V level. The J and K inputs of flip-flop B are connected to the flip-flop output A. The flip-flop outputs A and B are ANDed by a two input AND gate (labelled as '1' in Fig. 7-31). The output of AND gate '1' is connected to the J and K inputs of flip-flop C. Finally the flip-flop outputs of A, B and C flip-flops are ANDed together by a three-input AND gate (labelled as '2' in Fig. 7-31). The output of AND gate ('2') is connected to J and K inputs of flip-flop D.

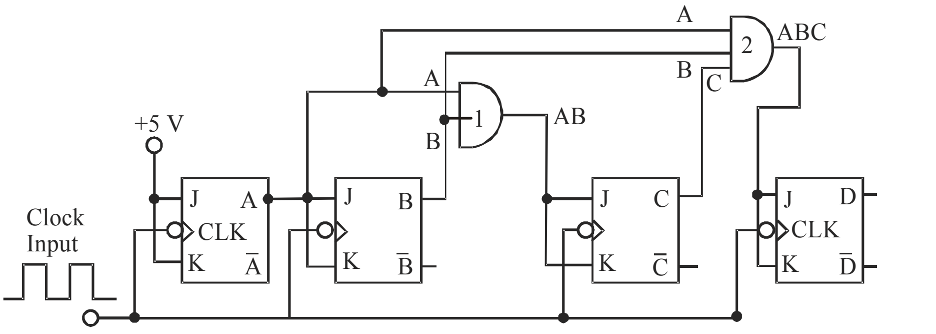


Fig. 7-31. MOD-16 Synchronous binary counter

Circuit Operation

The circuit shown in Fig. 7-31 will operate properly if those flip-flops that are supposed of toggle on the falling edge have J = K = HIGH (logic 1).

Let us study the counting sequence of the four-bit synchronous counter shown in Fig. 7-32 to understand the counter operation.

	Flip-flop outputs			
Count	***D***	***C***	***B***	***A***
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
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
.	.	etc	.	.

Fig. 7-32. Counting Sequence of a 4-bit synchronous counter.

Notice that the flip-flop A changes states on each falling edge of the input clock pulse. The flip-flop B changes its states on each falling edge of the input clock pulse when A =1. For example, when the count is 0001, the next falling edge of the input clock pulse toggles B to 1 state. Similarly when the count is 0011, the next falling edge must toggle B to the 0 state and so on. This operation has been possible because the output A is connected to J and K inputs of flip-flop B. Thus when ever A = 1, the J and K inputs of flip-flop B equals 1.

The flip-flop C changes the state on each falling edge that occurs while A = B = 1. For example, when the count is 0011, the next falling edge toggles C to the one state. Similarly, when the count is 0111, the next falling edge toggles C to the 0 state and so on. The operation of flip-flop C has been accomplished by connecting the logic signals from flip-flop A and B to J and K inputs of flip-flop C.

Following the same argument, the flip-flop D changes the state on each falling edge while A = B = C = 1. For example, when the count is 0111, the next falling edge toggles D to the 1 state. Similarly, when the count is 1111, the next falling edge toggles D to the 0 state. This operation of flip-flop D has been accomplished by connecting the outputs of flip-flops A, B and C to the J and K inputs of flip-flop D.

Note. From the above discussion, we can state the basic principle of constructing a synchronous counter as follows:

Each flip-flop should have its J and K inputs connected in such a way that they are HIGH only when the outputs of all lower-order flip-flops are in the HIGH state.

7-16. Advantages of Synchronous Counters Over Asynchronous Counters

We have already discussed in the previous articles about both the synchronous and asynchronous counters. In a synchronous (also called parallel counter), all the flip-flops change states simultaneously. In other words, all the flip-flops are synchronized to the falling edge of input clock pulses. Thus unlike the asynchronous counters, the propagation delays of flip-flops in a synchronous counter *do not* add together to produce the overall delay. Instead, the total response time of a synchronous counter is equal to the time it takes one flip-flop to toggle (t_{pd}) plus the time for new logic levels to propagate through a *single* AND gate (t_{pd}) to reach the J and K inputs. Mathematically,

$$\text{total delay} = t_{pd} \text{ of flip-flop} + t_{pd} \text{ of AND gate}$$

It may be carefully noted that the total delay is the same no matter how many flip-flops are in the counter. Generally the total delay in a synchronous counter is much lower than with an asynchronous counter for same number of flip-flops. Because of this fact, a synchronous counter can operate at a much higher input frequency. However, the circuitry of a synchronous counter is more complex than that of an asynchronous counter.

Example 7-22. *For a* 4-*bit MOD*-16 *synchronous binary counter shown determine* f_{max} *if* t_{pd} *for each flip-flop is* 10 *ns and* t_{pd} *for each AND gate is* 3 *ns. Compare this value with* f_{max} *for a MOD*-16 *ripple counter.*

Solution. Given : t_{pd} for each flip-flop = 10 ns and t_{pd} for each AND gate = 3 ns.

We know that for a synchronous counter,

$$\text{Total delay} = t_{pd} \text{ of flip-flop} + t_{pd} \text{ of AND gate}$$
$$= 10 \text{ ns} + 3 \text{ ns} = 13 \text{ ns}$$

Thus the period of input clock pulses,

$$T_{clock} \geq 13 \text{ ns}$$

The maximum frequency of input clock pulses, for a synchronous counter,

$$f_{max} = \frac{1}{13 \text{ ns}} = \frac{1}{13 \times 10^{-9}} = 76.9 \times 10^{6} \text{ Hz} = 76.9 \text{ MHz } \textbf{Ans.}$$

We also know that a 4-bit ripple counter uses four flip-flops. Since each flip-flop has t_{pd} = 50 ns, therefore, the total time delay = 4 × 10 ns = 40 ns. Therefore the period of input clock pulses,

$$T_{clock} \geq 40 \text{ ns}$$

and the maximum frequency,

$$f_{max} = \frac{1}{40 \text{ ns}} = \frac{1}{40 \times 10^{-9}} = 25 \text{ MHZ } \textbf{Ans.}$$

7-17. Actual Integrated Circuits for Synchronous Counters

Although there are several synchronous ICs in both the TTL and CMOS logic families, yet the following are important from subject point of view:

1. **IC 74 ALS 160 and 74 HC 160/162.** These are 4-bit synchronous BCD decade counters
2. **IC 74 ALS 161/163 and 74 HC 161/163.** These are 4-bit synchronous binary counters.

7-18. Synchronous Down Counter

We have already discussed in Art. 7-12 that a ripple counter could be made to count down by using the inverted output of each flip-flop to drive the next flip-flop in the counter. A synchronous down counter can be constructed in a similar manner, *i.e.* by using the inverted flip-flop outputs to drive the J, K inputs of the successive flip-flop. The 4-bit synchronous binary counter shown in Fig 7-31 can be converted to a down counter by connecting the inverted outputs, $\overline{A}$, $\overline{B}$ and $\overline{C}$ in place of A, B, C respectively. The counter will then proceed to count 15, 14, 13, 12, 11, ...3, 2, 1,0, 15, 14, 13, 12 and so on.

7-19. Synchronous Up/Down Counter

A up/down counter is a counter that is capable of counting in either direction through a certain sequence. It is also called bidirectional counter. In general most up/down counters can be reversed at any point in their sequence. For instance, the 3-bit binary counter can be made to go through the following sequence:

$$\underbrace{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6}_{\text{UP}} \quad \overbrace{5,\ 4,\ 3,\ 2}^{\text{DOWN}} \quad \underbrace{3,\ 4,\ 5,\ 6,\ 7,}_{\text{UP}} \quad \overbrace{6,\ 5,\ 4,}^{\text{DOWN}} \quad \text{etc}$$

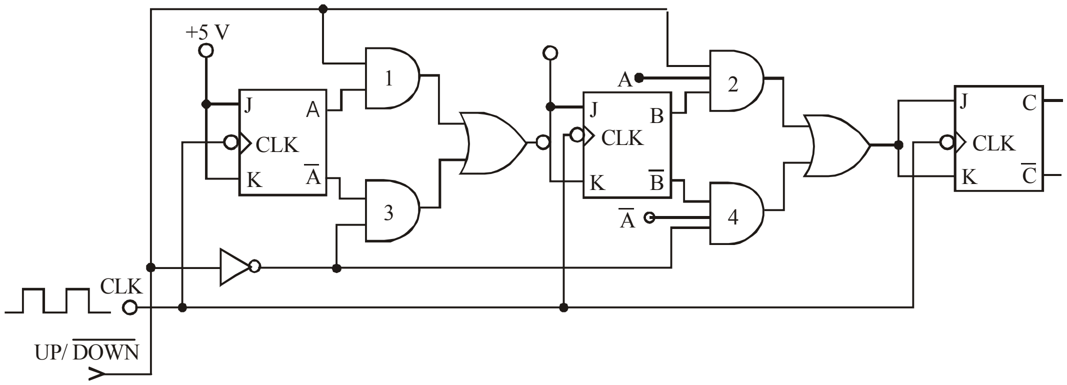


Fig. 7-33. A 3-bit synchronous up/down Counter

Fig. 7-33 shows the circuit of a 3-bit synchronous up/down counter. As seen from this circuit, $UP/\overline{DOWN}$ controls whether the normal flip-flop outputs or the inverted flip-flop outputs are fed to the J and K inputs of the successive flip-flops. When $UP/\overline{DOWN}$ is held HIGH, the AND gates 1 and 2 are enabled while AND gates 3 and 4 are disabled (notice the presence of inverter). This allows the A and B outputs to pass through gates 1 and 2 to the J and K inputs of flip-flops B and C.

On the other hand, when $UP/\overline{DOWN}$ is held LOW, the AND gates 1 and 2 are disabled while AND gates 3 and 4 are enabled. This allows the $\overline{A}$ and $\overline{B}$ outputs to pass through the AND gates 3 and 4 to the J and K inputs of flip-flops B and C.

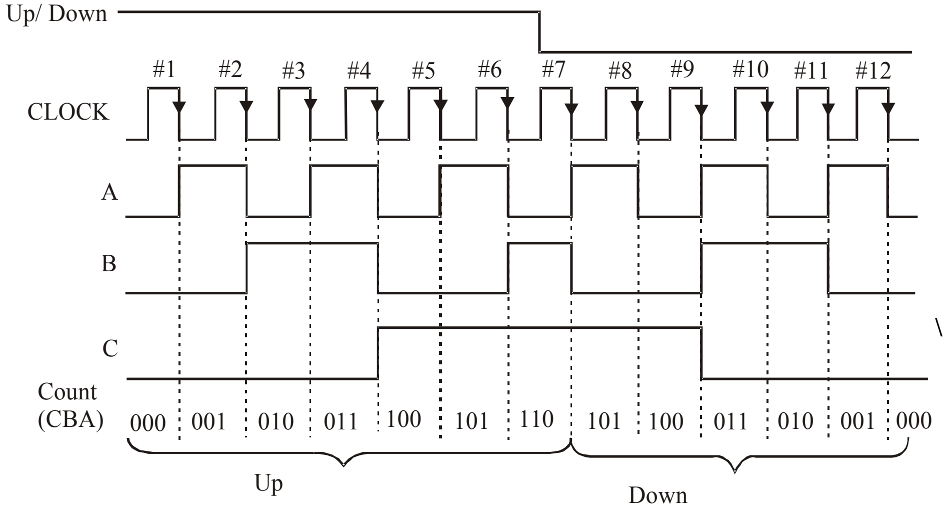


Fig. 7-34. Illustration of the waveforms for a 3-bit synchronous up/down counter

Fig. 7-34 shows the waveforms to illustrate the up/down counter operation. Notice that for the first six clock pulses, $UP/\overline{DOWN}$ = 1 and the counter counts up. However, for the last six pulses, $UP/\overline{DOWN}$ = 0 and the counter counts down.

7-20. Presettable Synchronous Counters

As a matter of fact, many synchronous (or parallel) counters are available as ICs. These are designed to be presettable *i.e.* they can be preset to any desired starting count either asynchronously (i.e. independent of clock signal) or synchronously (i.e. on the rising/falling edge of clock signal). This presetting operation is also known as parallel loading of the counter.

Fig. 7-35 shows a logic circuit for a 3-bit presettable synchronous up counter. Notice carefully the wiring of J, K, CLK PRESET and CLEAR inputs. The counter can be loaded with any desired count at any time by performing the following two operations:

1. Apply the desired count to the parallel data inputs P_0, P_1 and P_2
2. Apply a LOW pulse to the Parallel Load input $\overline{PL}$.

This will perform an asynchronous transfer of the P_0, P_1 and P_2 levels into flip-flops $Q_{0,}$ $Q_{1,}$and Q_2 respectively. The transfer occurs independently of J, K and CLK inputs. It may be carefully noted that the effect of CLK input will be disabled as long as $\overline{PL}$ is in its active-LOW state since each flip-flop will have one of its asynchronous inputs activated while $\overline{PL}$ = 0. Once $\overline{PL}$ goes HIGH, the flip-flops can respond to their CLK inputs and can resume the counting-up operation starting from the count that was loaded into the counter.

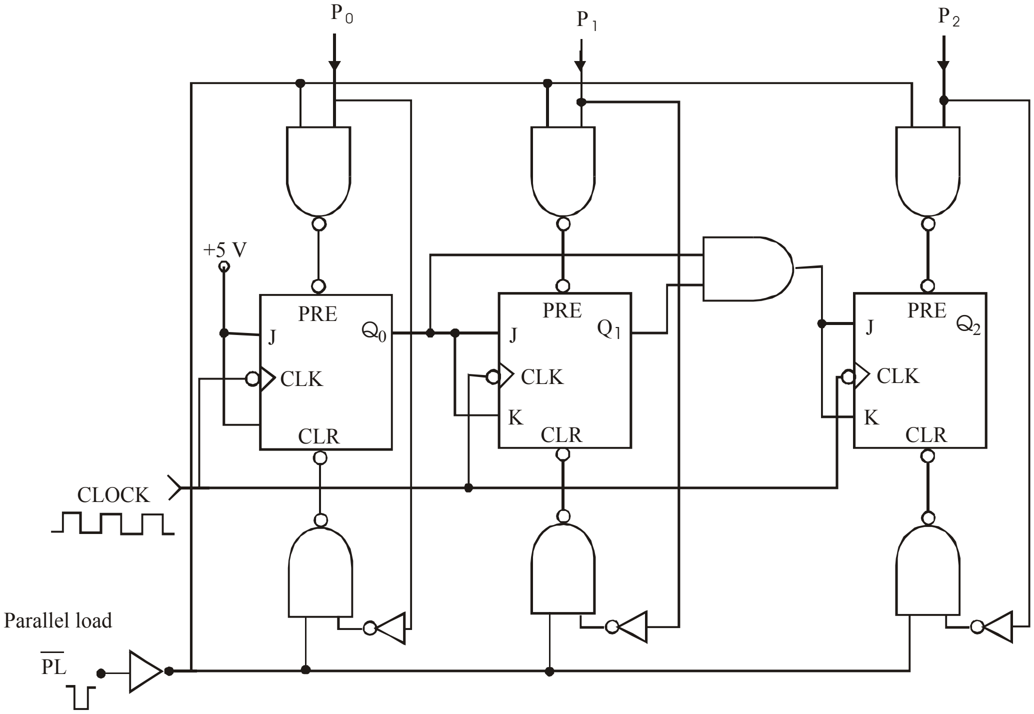


Fig. 7-35. 3-bit presettable synchronous up counter

The asynchronous presetting is available in several IC counters, such as the TTL 74190, 74191, and the CMOS equivalents 74190, 74191, 74192 and 74193.

7-21. Synchronous Up/Down Counter ICs

The 4-bit synchronous binary counters are available in a single IC package. Two popular synchronous IC counters are the 74192 and 74193. They can count up/down and can be preset to any count that you desire. The 74192 is a BCD decade up/down counter and 74193 is a 4-bit binary up/down counter.

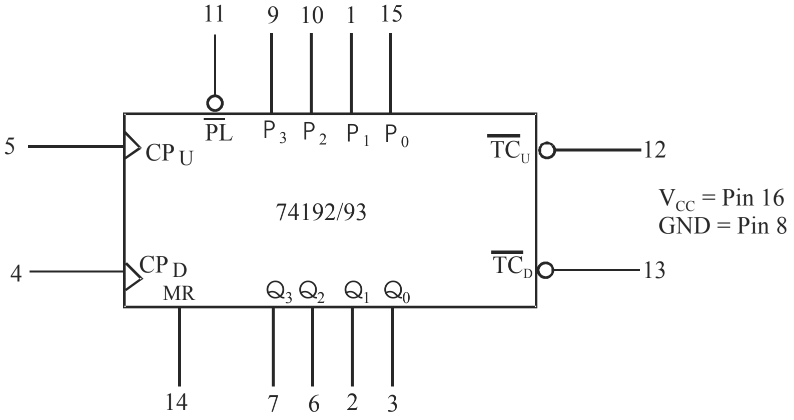


Fig. 7-36. Logic symbol for 74 192/S 193 synchronous counter IC

Fig 7-36 shows the logic symbol for 74192/193 synchronous IC counter. As seen from this diagram, there are two separate clock inputs 1) CP_U for counting up and 2) CP_D for counting down. One clock must be held HIGH while counting with the other. The binary output count is taken from Q_0 to Q_3. These are the outputs from four internal J-K flip flops. The output Q_0 is least significant bit (LSB) of the count while Q_3 is the most-significant bit (MSB). The Master Reset (MR) is an active-HIGH reset for setting the Q outputs to zero.

The counter can be preset by placing any binary value on the parallel data inputs P_0 through P_3 and then driving the Parallel Load ($\overline{PL}$) line LOW. The parallel load operation will change the counter outputs regardless of the conditions of the clock inputs.

The Terminal Count UP $(\overline{TC}_U)$ and Terminal Count Down $(\overline{TC}_D)$ are normally HIGH. The $\overline{TC}_U$ is used to indicate that the maximum count is reached and the count is about to recycle to zero (carry condition). The $(\overline{TC}_U)$ line goes low for 74193 when the counter reaches 15 and the input clock CP_u goes HIGH to LOW. $(\overline{TC}_U)$ remains LOW until $\overline{CP}_U$ returns HIGH. This low pulse at $(\overline{TC}_U)$ can be used as a clock input to the next-high order stage of a multi-stage counter.

The $(\overline{TC}_U)$ count output for the 74192 is similar except that it goes LOW at 9 and a LOW $\overline{CP}_U$. The Boolean equation for $\overline{TC}_U$ for 74193 counter may be written as,

$$\text{LOW at } \overline{TC}_U = Q_0 \; Q_1 \; Q_2 \; Q_3 \; \overline{CP}_U \qquad \text{................(for 74193 counter)}$$

and for 74192 counter,

$$\text{LOW at } \overline{TC}_U = Q_0 \; Q_3 \; \overline{CP}_U \qquad \text{..............(for 74192 counter)}$$

The Terminal Count Down $\left(CP_D\right)$ is used to indicate that the minimum count is reached and the count is about to recycle to the maximum count (i.e. 15 for 74193 and 9 for 74192) borrow conditions. Therefore $\overline{TC}_D$ goes LOW when the down-count reaches and the input clock, $\overline{CP}_D$ goes LOW. The Boolean equation at $\overline{TC}_D$ is,

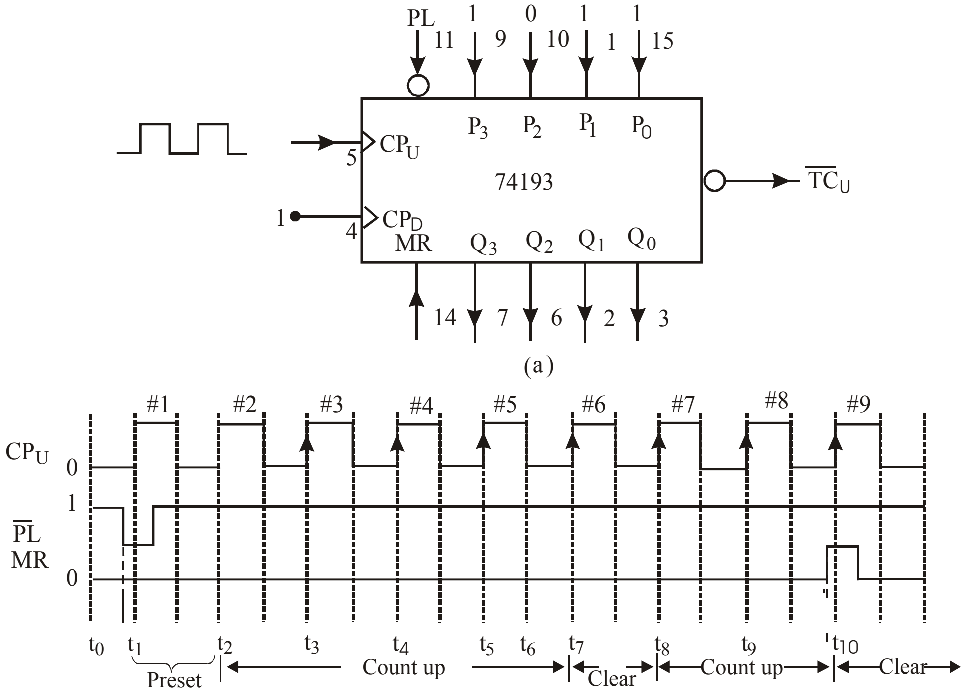
$$\text{Low at } \overline{TC}_D = \overline{Q}_0 \; \overline{Q}_1 \; \overline{Q}_2 \; \overline{Q}_3 \; \overline{CP}_D \qquad \text{.............(for 74192/74193)}$$

Fig. 7-37 shows the truth table for the IC74193 counter. This table can be used to show the four operating modes (i.e. Reset, Load,Count up and Count down) of 74192/74193 counter.

Operating Mode	***Inputs***								***Outputs***					
	MR	$\overline{PL}$	CP_U	CP_D	P_0	P_1	P_2	P_3	Q_0	Q_1	Q_2	Q_3	$\overline{TC}_U$	$\overline{TC}_D$
Reset	H	X	X	L	X	X	X	X	L	L	L	L	H	L
	H	X	X	H	X	X	X	X	L	L	L	L	H	H
Parallel load	L	L	X	L	L	L	L	L	L	L	L	L	H	L
	L	L	X	H	L	L	L	L	L	L	L	L	H	H
	L	L	L	X	H	H	H	H	H	H	H	H	L	H
	L	L	H	X	H	H	H	H	H	H	H	H	H	H
Count up	L	H	↑	H	X	X	X	X	Count up				H	H
Count down	L	H	H	↑	X	X	X	X	Count down				H	H

Fig. 7-37. Truth table for IC 74192/74193 Synchronous Counter.

Example 7-23. *Fig.* 7.38 (*a*) *shows the* IC74193 *wired as an up counter. The parallel data inputs are permanently connected as* 1011*, and the* CP_U , $\overline{PL}$ *and MR input waveforms are as shown in Fig* 7-38 (*b*). *Assuming the counter initially in the* 0000 *state, determine the counter output waveforms.*



(a)

(*b*)

Fig. 7-38.

Solution. Given the 74193 wired as an up counter.

The counter is preset to 1011. When the first clock pulses (i.e. #1) is applied , the counter does not count because $\overline{PL}$ is low. So only the value 1011 is loaded in the counter and the counter outputs indicate, $Q_3 = 1$, $Q_2 = 0$, $Q_1 = 1$ and $Q_0 = 0$ respectively. Refer to Fig. 7-39.

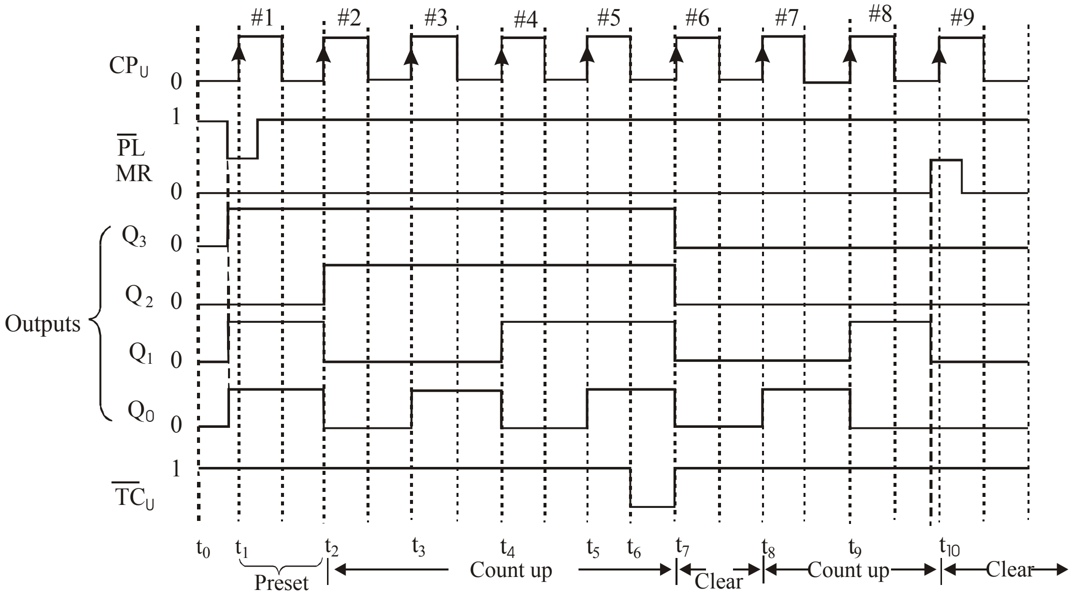
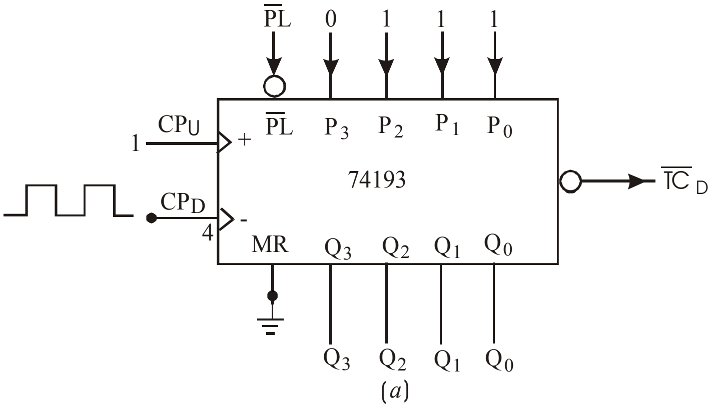


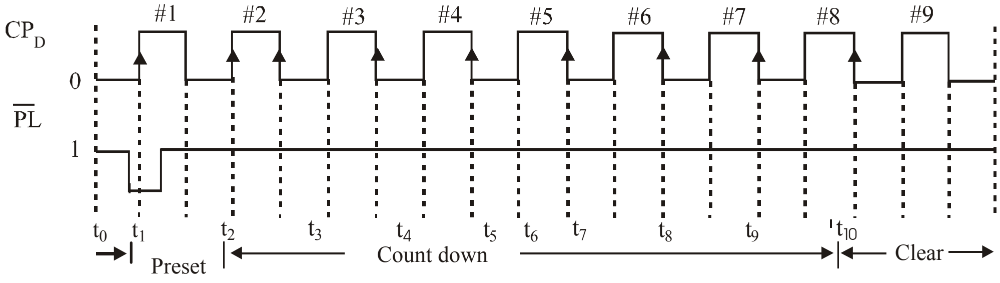
Fig. 7-39.

With the arrival of # 2 pulse, the counter counts to 1100. With the arrival of # 3 pulse, it counts to 1101, with # 4 pulse, to 1110, with # 5 pulse, to 1111. As a result $\overline{TC}_U$ goes LOW for a short duration and then goes HIGH again. Notice that at the arrival of # 6 pulse, the counter clears to 0000 and then with subsequent pulses #7 and #8 it counts as 0001, 0010. Just before the arrival of pulse # 9, MR goes HIGH, this clears the counter back to 0000 as shown in the waveforms of Fig. 7-39.

Example 7-24. *Fig.* 7-40 (*a*) *shows the* 74193 *wired as a down counter. The parallel data inputs are permanently wired as* 0111 *and the* CP_D *and* $\overline{PL}$ *waveforms are shown in Fig.* 7-40 (*b*). *Assuming that the counter is initially in the* 0000 *state, determine and sketch the output waveforms.*



(*a*)



(*b*)

Fig. 7-40.

Solution. Given: the 74193 wired as down counter.

The clock pulses are applied at CP_D. Notice that the counter is triggered at the rising edge of the clock pulse. The arrival of clock pulse #1, the count does not count because $\overline{PL}$ is LOW. As a result the counter is preset to the value 0111. Refer to Fig 7-41 (a). With the arrival of # 2 clock pulse, the counter counts down, i.e. it goes to 0110 state. With the arrival of #3 clock pulse, the counter goes to 0101 state. With the subsequent clock pulses it continues to count down i.e. 0100, 0011, 0010 and then to 0001. The counter clears at the arrival of # 9 clock pulse. Notice that when the count reaches 0000, $\overline{TC}_D$ goes LOW for a short duration and goes HIGH again.

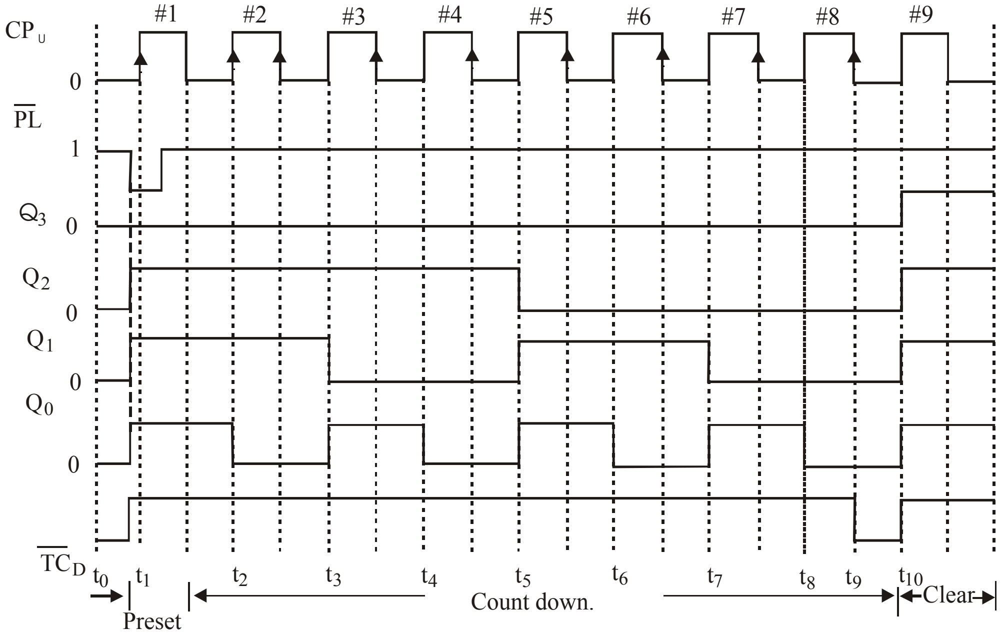


Fig. 7-41.

7-22. Counter with Variable MOD Number using 74193

It will be interesting to know that presettable counters can easily be wired for different MOD numbers without the need for additional logic circuitry. Let us illustrate this with the help of the diagram shown in Fig. 7-42.

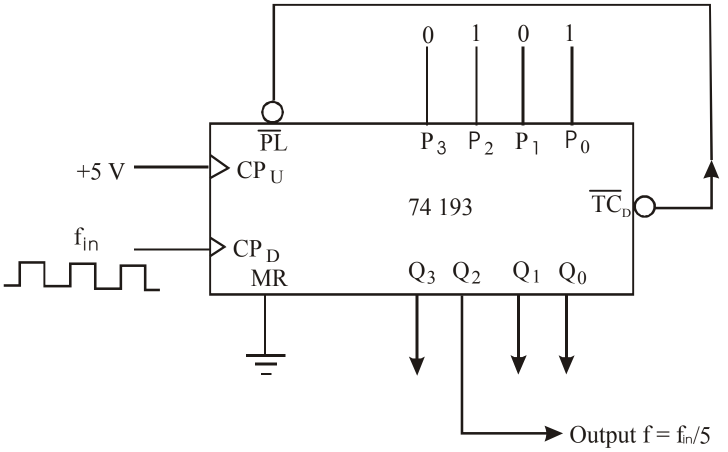


Fig. 7-42. Illustrating the use of 74193 as a counter with variable MOD Number

As seen from this figure, 74193 is used as a down counter with its parallel load inputs permanently connected to 0101 (i.e. 5_{10}). It may be carefully noted that $\overline{TC}_D$ output of 74193 is connected to its $\overline{PL}$ input. We will begin our study by assuming that the counter is in the 0101 state at time t_0, i.e. before the arrival of # 1 clock pulse as shown in Fig 7-43.

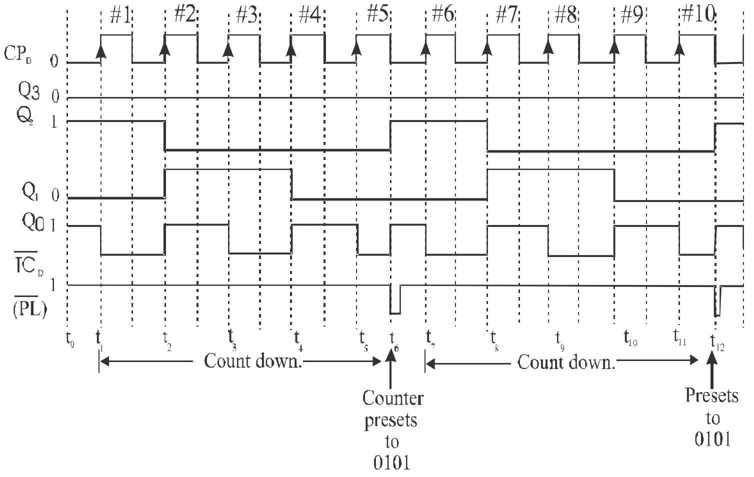


Fig. 7-43.

With the arrival of # 1 clock pulse, the counter will decrement. It will continue to decrement until # 5 clock pulse. At this instant, (i.e. t_5), the counter has reached 0000. When the # 5 clock pulse goes LOW, it drives $\overline{TC}_D$ LOW for a short duration (refer to Fig. 7-43). This immediately activates $\overline{PL}$ input and presets the counter back to 0101 state. The counter will decrement again down to 0000 with the next 5 clock pulses. A close examination of Q_2 shows that it goes through one complete cycle after every five cycles between the rising edge of Q_2 at t_6 and the rising edge of Q_2 at t_{12}. Thus the frequency of the Q_2 waveform is one-fifth (1/5 th) of the clock frequency.

There is an interesting issue about a counter with a variable MOD number shown in 7-42 and waveforms shown in Fig. 7-43. Notice that it counts through six different states i.e. 5, 4, 3, 2, 1 and 0, yet it divides a frequency by 5. This is due to the unusual way the count gets preset back to 5 in the middle of a clock cycle.

Thus this counter's operation violates our general rule that the number of states and the frequency division are the same. Since this arrangement is used principally for frequency division, we will ignore the counting sequence and call it a MOD-5 counter because it divides the clock frequency by 5.

It may be carefully noted that it is not a coincidence that the frequency – division ratio (5) is the same as the number applied to parallel data inputs of the counter IC shown in Fig. 7-42. As a matter of fact, we can vary the frequency division by changing the logic levels applied to the parallel data inputs.

Note. The arrangement shown in Fig 7-42 can be used to implement a variable frequency divider circuit. This is achieved by connecting the switches to the parallel data inputs of 74193 counter IC. The switches can be set to a value equal to the desireable frequency divisiion ratio. However, care must be taken to choose the appropriate Q output, depending upon the frequency – division ratio that is selected.

Example 7-25. *Fig.*7-44 *shows the logic symbol of* 74193 *counter.*

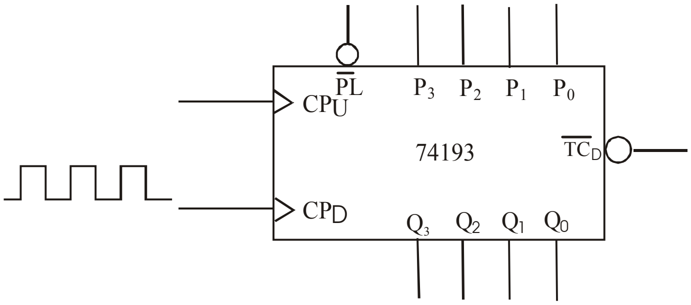


Fig. 7-44.

Show how you will wire this as a MOD-9 counter. Sketch the timing waveforms at the four outputs Q_3, Q_2, Q_1 and Q_0 and $\overline{TC}_D$ output.

Solution. Given : The logic symbol of 74193 counter.

The counter can be wired as a MOD – 9 counter as shown in Fig. 7-45.

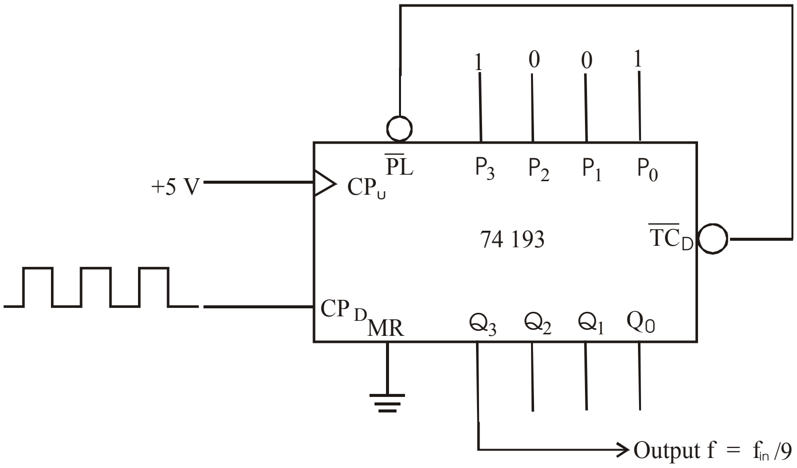


Fig. 7-45.

As seen from the figure above, 74 HC 193 is used as a down counter with its parallel load inputs connected to 1001 (i.e. 9_{10}). The $\overline{TC}_D$ output is connected to $\overline{PL}$ input. The MR input is connected to the ground while CP_U input is connected to +5 V (i.e. logic HIGH).

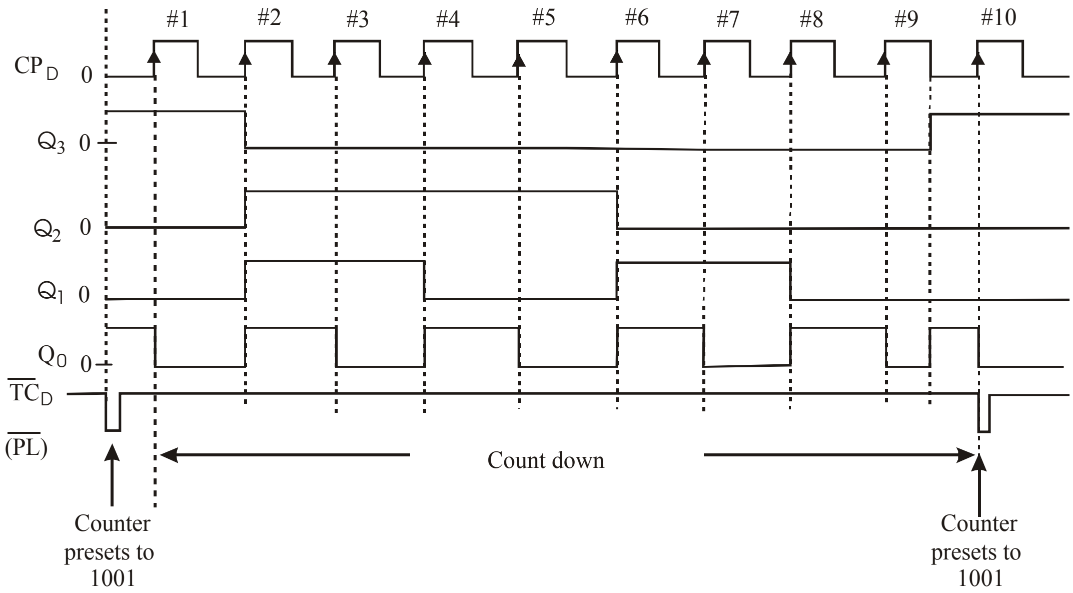


Fig. 7-46. Illustrating waveforms for a counter wired as shown in Fig. 7-45

Fig. 7-46 shows the waveforms at Q_3, Q_2, Q_1 and Q_0 outputs and at $\overline{TC}_D$ (or $\overline{PL}$). These waveforms can be sketched by following the similar argument as discussed in Art 7-22.

7-23. Multistage Arrangement to Extend Maximum Counting Range

We have already discussed in Art 7-21 about the 74193 counter. As discussed, 74193 is a 4 – bit binary counter that can be used to count up from 0 ⟶ 15 or to count down from 15 ⟶ 0. The counting range can be increased by cascading more than one 74193s. Fig. 7-47 shows a two-stage up/ down counter arrangement using 74193. This arrangement increases the up-count range to 0 ⟶255 and the down count range to 255 ⟶ 0.

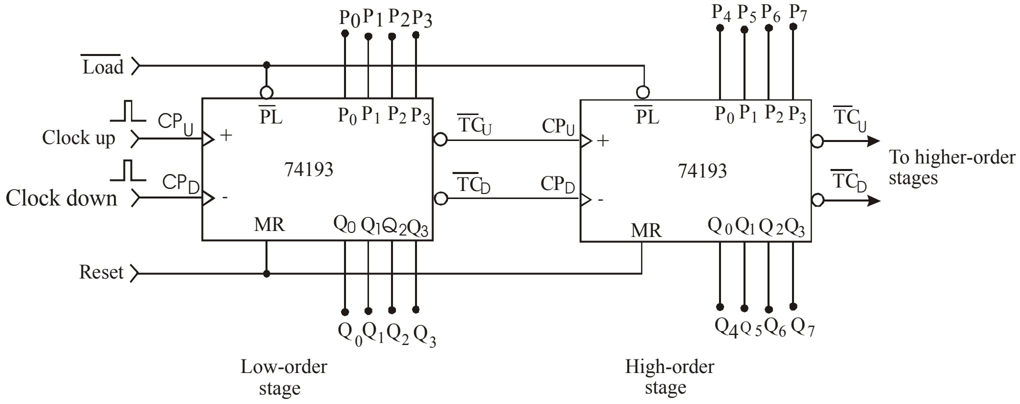


Fig. 7-47.

As seen from the figure, the block on the left is the low-order stage. This stage is clocked by one or the other of the clock inputs. Further the $\overline{TC}_U$ and $\overline{TC}_D$ outputs of this stage drive the clock inputs of the high-order stage. The load input ($\overline{P}_L$) and the master reset (MR) of the two-stages are connected together to $\overline{Load}$ and Reset respectively. It may be noted that the parallel data inputs to the high-order stage are labelled as P_4 P_5 P_6 P_7 and the outputs as Q_4 Q_5 Q_6 Q_7 . This arrangement allows the counter to operate as an 8-bit counter. It can be preset into any 8–bit value, can be made to count-up or count-down. The 8-bit count at any time appears at Q_0-Q_7 outputs.

Example 7-26. *Fig.* 7-48 *shows the logic symbol of a* 74193 *counter. Using two such IC's show how the arrangement can be wired so that it can be used to divide the input frequency by* 200.

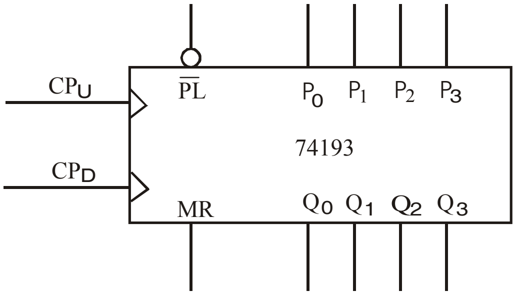


Fig. 7-48.

Solution. Given the logic symbol of a 74193 counter.

We know that a single 74193 counter is a 4 – bit counter. It can be used to count to 0 ⟶ 15 or 15 ⟶ 0. In order to count beyond this up to 255, we need to cascade two 74193s. The arrangement is shown in Fig. 7-49 and is called an 8-bit counter that is capable of counting 256 states ($\because$ 2^8 = 256).

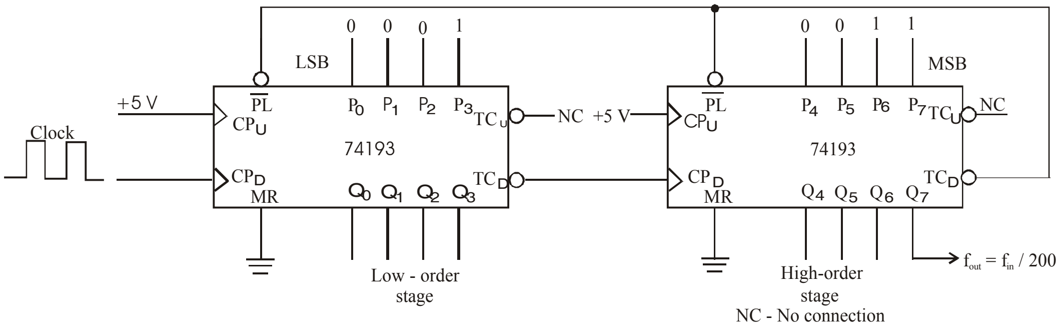


Fig. 7-49.

In order to build a counter that can divide the input frequency by 200, we have to preset the cascaded arrangement of 74193s to a binary equivalent of the number 200 (i.e. 11001000_2) and count-down to zero. We can use the $\overline{\text{TD}}_{\text{C}}$ of the high-order stage to drive the parallel load ($\overline{\text{PL}}$) line LOW to reset to 200.

Note. The output can be taken either for $\overline{\text{TD}}_{\text{D}}$ of the high-order stage or its Q_7 output. If the pulse duration is short, it can be widened to produce any duty cycle without affecting the output frequency by using a one-shot (refer to chapter 5, Art 5-40).

7-24. Decoding Counter States

As a matter of fact, digital counters are used in applications where the count represented by the states of flip–flops must somehow be determined or displayed. One of the simplest approach to do so is to connect an LED at the output of each flip–flop (as seen earlier in Fig. 7-14). In this approach, the count can be mentally determined by decoding the binary states of LEDs (bright = 1, dark = 0). But this method becomes inconvenient as the counter size increases. Now we will study a method for counter decoding that would require no mental operation.

Fig. 7-50 shows a decoding logic for MOD-8 counter. As seen, the decoder consists of eight three – input AND gates. Each AND gate produces a HIGH output for one particular state of the counter. Consider for example, an AND gate '0' whose inputs are $\bar{A}$, $\bar{B}$ and $\bar{C}$. Its output will be LOW at all times except when A = B = C = 0, i. e. on the count of 000_2 (O_{10}). Similarly AND gate 1 has A, $\bar{B}$, and $\bar{C}$ as its inputs. Its output will be LOW at all times except when A = 1, B = C = 0 (i.e on the count of $001_2 (1_{10})$. Further AND gate 5 has A, B and C as its inputs. Its output will be LOW at all times except when A = 1, B = 0 and C = 1, i.e. on the count of $101_2(5_{10})$. The rest of AND gates operate in the same way for the other possible counts.

It is evident from the above discussion that only one AND gate output is HIGH, the one which is decoding for the particular count that is present in the counter. This method of decoding logic is called Active-HIGH decoding.

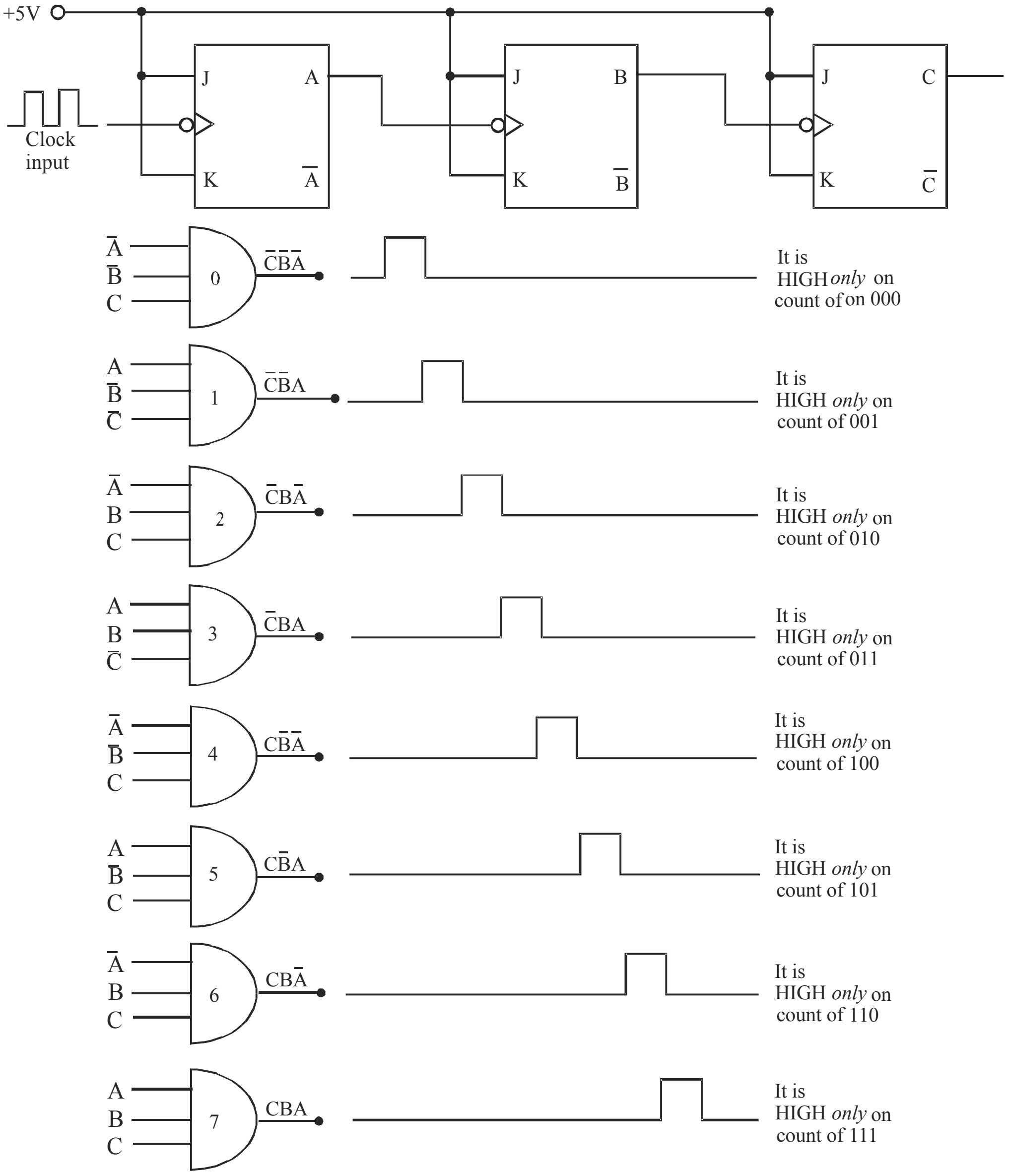

+5V
Clock input
J
K
A
Ā
J
K
B
B̄
J
K
C
C̄
Ā
B̄
C̄
0
C̄B̄Ā
It is HIGH *only* on count of on 000
A
B̄
C̄
1
C̄B̄A
It is HIGH *only* on count of 001
Ā
B
C̄
2
C̄BĀ
It is HIGH *only* on count of 010
A
B
C̄
3
C̄BA
It is HIGH *only* on count of 011
Ā
B̄
C
4
CB̄Ā
It is HIGH *only* on count of 100
A
B̄
C
5
CB̄A
It is HIGH *only* on count of 101
Ā
B
C
6
CBĀ
It is HIGH *only* on count of 110
A
B
C
7
CBA
It is HIGH *only* on count of 111

Fig. 7-50. Illustrating the use of AND gates to decode MOD – 8 counter states

It will be interesting to know that if NAND gates are used in place of AND gates, the decoder outputs will produce a normally HIGH signal. This signal goes LOW only when the number being decoded occurs. This method of decoding logic is called Active-LOW decoding.

In actual practice both the Active- HIGH as well as the Active-LOW method of decoding are used. The choice depends on the types of circuits being driven by the decoder outputs.

Example 7-27. *How many AND gates are required to completely decode all of the states of a MOD-*32 *binary counter*? *What are the inputs to the gate that decodes for the count of* 21?

Solution. Given : A MOD–22 binary counter.

We know that the number of AND gates required to completely decode all the states of a MOD-32 binary counter is equal to the number of counter states, i.e. 32 **Ans.**

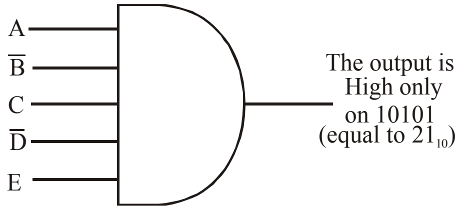
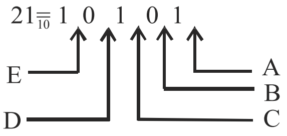


Fig. 7-51.

Fig. 7-51 shows the inputs A, B, C, D and E to the AND gate that decodes for the count of 21. Notice that



where A, B, C, D and E represents their normal outputs of the flip-flops respectively.

7-25. Decoding the Glitches

We have already discussed in Art. 7-13 about the effects of flip-flop propagation delays in ripple counters. Recall that the accumulated propagation delays serve essentially to limit the frequency response of ripple counters. The delays between flip-flop transitions can also cause problems when decoding a ripple counter. The problem occurs in the form of decoding glitches or spikes at the outputs of some of the decoding gates.

In order to understand it in more detail let us consider a MOD-4 ripple counter shown in Fig. 7-52 (a) The outputs of each flip-flop and decoding gates are shown in the Fig. 7-52 (b) & (c). Notice the propagation delay between the clock input waveform and the flip-flop A output waveform. Also notice the propagation delay between the flip-flop. A output waveform and B output waveform. The glitches in the Y_0 and Y_2 decoding waveforms are caused by the delay between the A and B output waveforms. Notice that Y_0 is the output of the AND gate decoding for the normal 00 count. The 00 condition also occurs momentarily as the counter goes from the 01 to the 10 counter as shown by the waveforms. This is because of the fact that the flip-flop B cannot change states until flip-flop A goes LOW. This momentary 00 state lasts only for several nanoseconds (depending on t_{pd} of flip-flop B) but can be detected by the decoding gate if the gate's response is fast enough. Hence, the spike at the Y_0 output.

A similar situation produces a glitch at the X_2 output. The X_2 output is decoding for the 10 condition. This condition occurs momentarily as the counter goes from 11 to 00 in response to the fourth clock pulse as shown in Fig. 7-52 (c). Again this is due to the delay of flip-flop B's response after A has gone LOW.

Although the situation is illustrated for a MOD- 4 counter, yet the same type of situation can occur for any ripple counter. This is because of the fact that ripple counters work on the principle of "chain reaction", i.e. each flip-flop triggers the next and so on. The spikes at the decoder outputs may or may not present a problem, depending on how the counter is used. When the counter is being used only to count pulses and display the results, the decoding spikes are of no consequence because they are very short in duration and will not even show up on the display. However when the counter is used to control other logic circuits, the spike can cause improper operation.

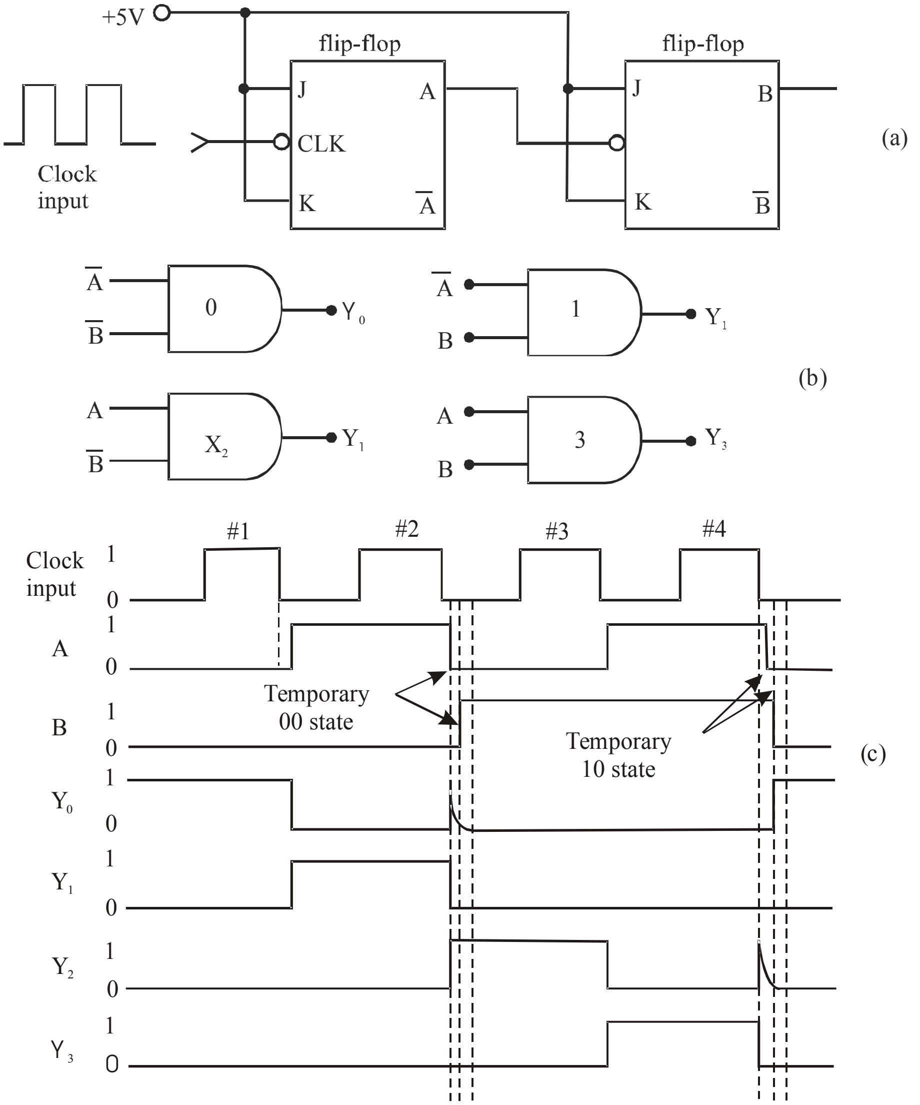


Fig. 7-52. Illustrating the decoding of glitches for a MOD-4 counter

In situations where the decoding spikes cannot be tolerated, there are two basic solutions to the problem. The first possibility is to use a parallel counter instead of a ripple counter. Recall that in a parallel counter, all the flip-flops are triggered at the same time by the clock pulses so that it appears that the conditions that produced the decoder spikes cannot occur. However in a parallel counter, the spikes may occur because the flip-flops will not all necessarily have the same propagation delay (t_{pd}) especially when some flip-flops may be loaded more heavily than others.

The second method is to use a technique called strobing. This is discussed separately in the next article.

7-26. Strobing – a method for Eliminating Decoder Spikes

Strobing is a very reliable method for eliminating the decoder spikes. It requires a signal called "strobe signal to keep the decoding AND gates disabled (i.e. outputs at 0) until all the flip-flops have reached a stable state in response to the falling edge of the clock pulse.

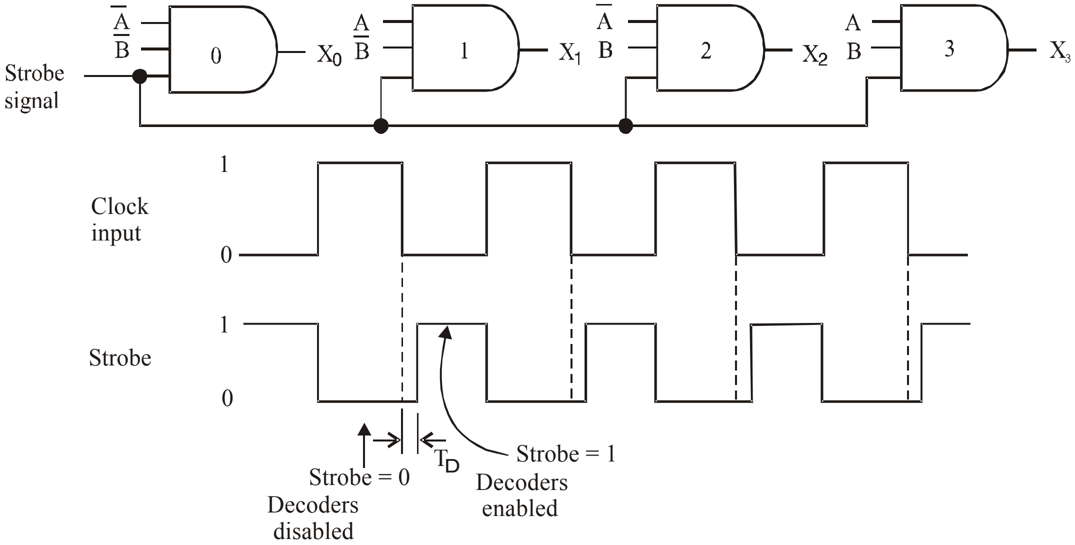


Fig. 7-53. Illustrating the use of strobe signal to eliminate decoding spikes

Fig. 7-51 Illustrate the use of strobe signal to eliminate decoding spikes. As seen from this figure, the strobe signal is connected as an input to each decoding AND gate. Notice that the strobe signal goes LOW when the clock pulse goes HIGH. During the time that the strobe is LOW, the decoding gates are kept LOW. The strobe signal goes HIGH to enable the decoding gates some time T_D, after the clock pulse goes LOW. The time T_D is chosen to be greater than the total time it takes the counter to reach a stable count. The value of T_D depends on two factors :

1. The flip-flop delays and
2. Number of flip-flops in the counter.

It may be noted that the strobe method is not used if a counter is used only for display purposes. It is because of the fact that decoding spikes are too narrow to affect the display. The strobe signal is used when the counter is used in control applications.

7-27. Cascading BCD Counters

We have already discussed in Art 7-10 that a single BCD counter counts from 0 through 9 and then resets to 0. In order to count beyond 9, we can cascade BCD counters as shown in Fig. 7-54. This counter can be used to count up to 99. The operation of the cascaded arrangement may be understood from the following discussion :

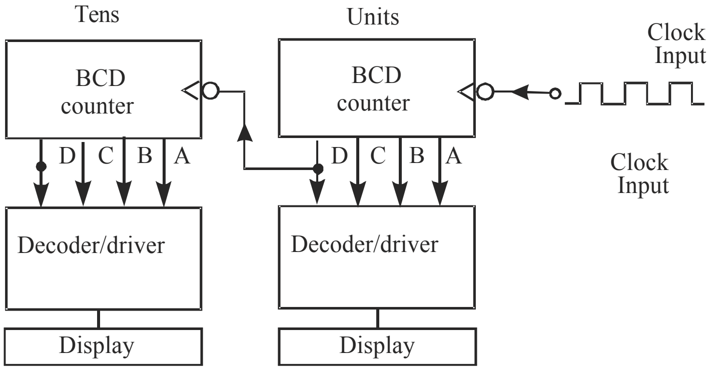


Fig. 7-54. Illustrating the cascade connection of BCD counters to count from 00 to 99.

1. To start with, all counters are cleared to the 0 state. The decimal display is 000.
2. As the clock input pulses arrive, the units BCD counter advances one count per pulse. After nine pulses have occurred, the hundreds and tens BCD counter's are still at 0, and the units counter is at 9 (i.e. binary 1 001). Thus the decimal display indicates 09.
3. On the arrival of 10th clock input pulse, the units counter resets to 0, causing its D flip-flop to go from 1 to 0. This 1 to 0 transition acts as a clock input for the tens counter and causes it to advance one count. Thus after 10 input pulses, the display indicates 10.
4. As additional pulses occur, the units counter advances one count per pulse and each time the units counter resets to 0, it advances the tens counter one count. Thus after 99 input pulses have occurred, the tens counter is at 9, as is the units counter. The decimal display indicates 99 at this instant.
5. On the hundredth input pulse, the units counter resets to 0, which in turn causes the tens counter to reset to 0. The flip-flop output of the tens counter thus makes a 1 – to – 0 transition. In other words on the 100th pulse both the counters resets to 0.

Notes.

1. The cascade arrangement can be expanded to any desired number of decimal digits, simply by adding more stages.
2. Each BCD counter in a cascaded arrangement shown in Fig. 7-54, could be a variable MOD counter such as the 74293 wired as MOD-10 counter or it could be an IC that is internally wired as a BCD counter such as 7490 or 74192.

7-28. Shift Register Counters

We have already discussed in chapter 5, Art 5-28 as how the flip-flops can be connected in a shift register arrangement to transfer data left to right or vice versa, one bit at a time (called serial data transfer). Shift register counters are the counters that use feedback, i.e. the output of the last flip-flop is connected back to the first flip-flop in some way.

There are two different shift register counters namely :

(1) ring counter and (2) Johnson counter.

Both these shift register counters are used to creat sequential control waveforms for digital systems. These are discussed one by one in the following pages.

7-29. Ring Shift Counter

Fig. 7-55 shows a four-bit ring shift counter using D-type flip-flops. The J-K flip-flops can also be used instead of D flip-flops. For convenience purpose, the PRESET amd CLEAR inputs are not indicated on the flip-flop. They are assumed to be connected HIGH. As seen from the figure, the information shifts from left to right and back around from Q_0 to Q_3 . In most instances, only a single

1 is in the register and it is made to circulate around the register as long as clock pulses are applied. It is because of this that this counter is called a ring shift counter.

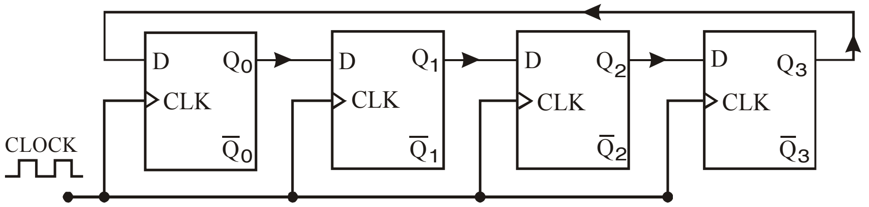
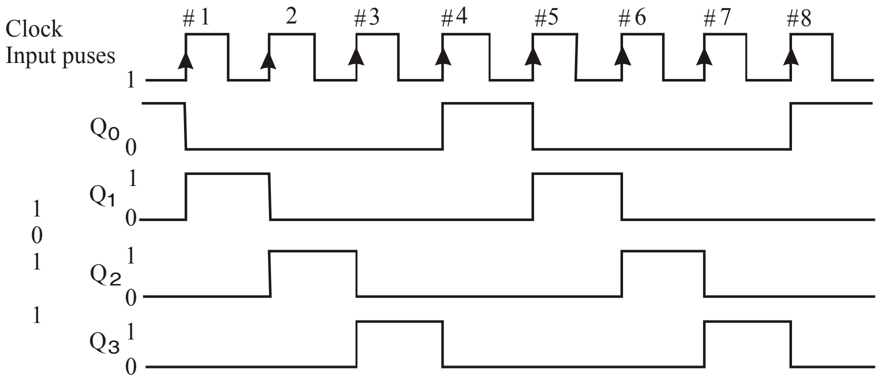


Fig. 7-55. Ring shift Counter

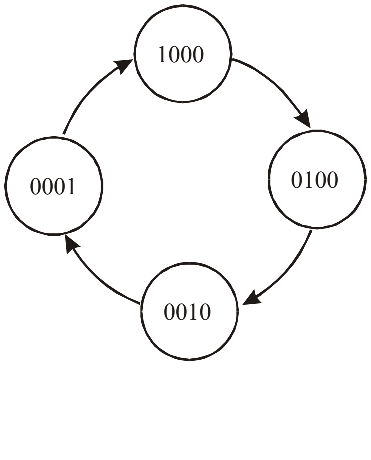
Fig. 7-56 (a), (b) and (c) shows the waveforms, sequence table and the state diagram for the ring shift counter shown in Fig. 7-55. The diagram shows the various states of the flip-flops as the clock pulses are applied. It is assumed that the starting state is $Q_0 = 1$ and $Q_1 = Q_2 = Q_3 = 0$. After the first clock pulse, the 1 has shifted from Q_1 to Q_2 so that the counter is in the 0100 state. The second pulse produces the 0010 state and the third pulse produces 0001 state. On the fourth pulse, the 1 from the Q_3 is transferred to Q_0, resulting in the 1000 state (i.e. the initial state). Subsequent pulses cause the sequence to repeat.



(*a*) Waveforms

Flip-Flop outputs				*CLK*
Q_0	Q_1	Q_2	Q_3	*pulse*
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7
1	0	0	0	8
.	.	.	.	.
.	.	.	.	.

(*b)* Sequence table



(*c)* State-transition diagram

Fig. 7-56.

The ring shift counter shown in Fig. 7-55 is a MOD-4 counter because it has 4 distinct states before the sequence starts. Although this circuit does not progress through the normal binary counting sequence, it is still a counter because each count corresponds to a unique set of flip-flop states. It may be noted that each flip-flop output waveform has a frequency equal to one fourth of clock frequency.

As a matter of fact, ring shift counter can be constructed for any desired MOD number. However, a MOD-N ring counter uses N flip-flops connected in the arrangement shown in Fig. In general, a ring shift counter will require more flip-flops than a binary counter for the same MOD number. For example, a MOD-4 ring counter requires four flip-flops while a MOD-4 binary counter requires two. A ring shift counter is very useful in applications where the counter is being used to control the sequencing of operations in a system.

7-30. Starting of a Ring Shift Counter

Strictly speaking, in order to operate a ring shift counter properly, it must start off with only one flip-flop in the 1 state and all the others in the 0 state. Since the starting states of the flip-flop will be unpredictable on power-up, the counter must be preset to the required starting state before clock pulses are applied. One way to do this is to apply a momentary pulse to the $\overline{\text{PRE}}$ input of one of the flip-flops (say flip-flop Q_0 in Fig. 7.55) and $\overline{\text{CLR}}$ input of all other flip-flops.

Fig. 7-57 shows another method to preset / clear the flip-flops in the desired state before the clock pulses are applied. On power-up, the capacitor will charge up relatively slowly towards $+V_{CC}$. The output of Schmitt-trigger INVERTER 1 will stay HIGH and the output of INVERTER 2 will remain LOW until the capacitor voltage exceeds the positive-going threshold voltage (V_{T+}) of the INVERTER 1 input (about 1.7 V). This will hold the $\overline{\text{PRE}}$ input of Q_0 and the $\overline{\text{CLR}}$ inputs of Q_0, Q_1 and Q_2 in the LOW state long enough during power up to ensure that the counter starts at 1000.

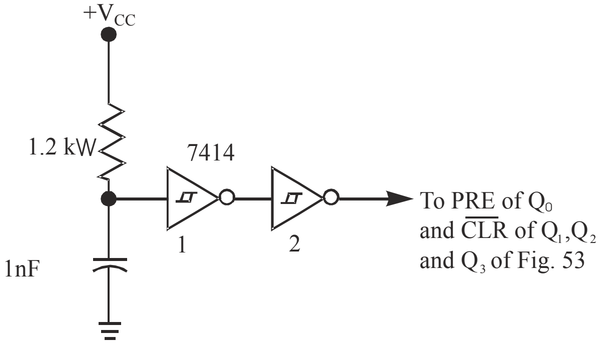


Fig. 7-57.

7-31. Johnson Shift Counter

The ring counter discussed in Art 7-29 can be modified slightly to produce another type of shift-register counter called Johnson or twisted-ring counter. This counter is constructed exactly like a normal ring counter except that the inverted output of the last flip-flop is connected to the input of the first flip-flop.

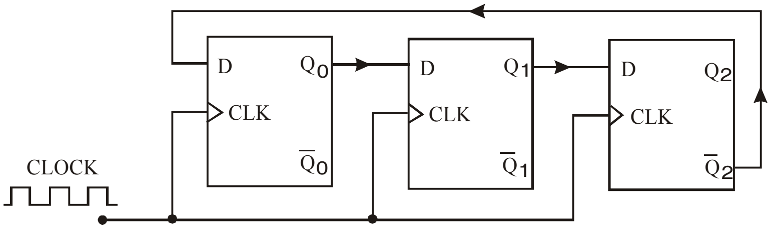
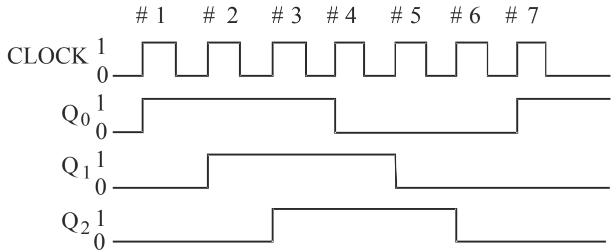


Fig. 7-58. 3-bit Johnson shift counter.

Fig. 7-58 shows a circuit of 3-bit Johnson shift counter. As seen from the figure, $\overline{Q_2}$ output is connected back to the D input of Q_0. This means that the inverse of the level stored in Q_2 will be transferred to Q_0 on the clock pulse.

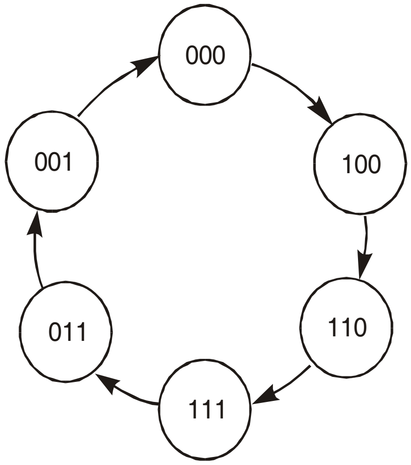
In order to understand the operation of Johnson shift counter, refer to the waveforms, sequence table and the state diagram shown in Fig. 7-59. Note that on each rising edge of the clock pulse, the level at Q_0 shifts into Q_1, the level at Q_1 shifts into Q_2 and the inverse of the level at Q_2 shifts into Q_0.



(a)

Flip-Flop outputs			*CLK*
Q_0	Q_1	Q_2	*pulse*
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3
0	1	1	4
0	0	1	5
0	0	0	6
1	0	0	7
1	1	0	8
.	.	.	.
.	.	.	..

(*b*)



(*c*)

Fig. 7-59. Illustrating the waveforms, sequence table and the state diagram of a 3-bit Johnson shift counter

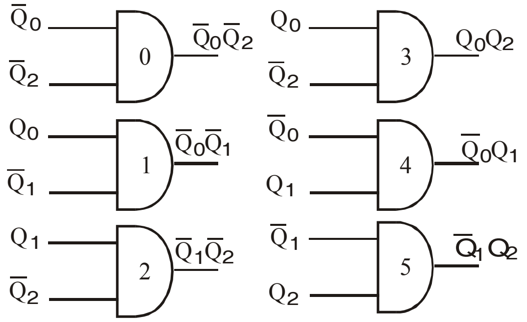
Following are some of the points which can be observed from Fig. 7-59 (a), (b) and (c),

1. The 3-bit Johnson shift counter has six distinct states : 000, 100, 110, 111, 011 and 001. Thus it is a MOD-6 Johnson counter. It may be carefully noted that this counter does not count in a normal binary sequence.
2. The output waveforms of each flip-flop is a square wave (i.e. duty cycle = 50 %). The frequency output of each flip-flop is 1/6th the frequency of the input clock pulses. Further, the flip-flop waveforms are shifted by one clock period with respect to each other.

It will be interesting to know that the MOD number of a Johnson shift counter will always be equal to twice the number of flip-flops. For example, if we connect four flip-flops in the arrangement of Fig. 7-59, the circuit will be a MOD-8 Johnson shift counter. In general, if there are 'N' number of flip-flops (where N is an even number), the resulting Johnson shift counter is a counter with MOD number = 2N.

7-32. Decoding a Johnson Shift Counter

Strictly speaking, for a given MOD number, a Johnson counter requires only half the number of flip-flops that a ring counter requires. However, a Johnson counter requires decoding gates whereas a ring counter does not. As in the binary counter, the Johnson counter uses one logic gate to decode for each count, but each gate requires only two inputs regardless of number of flip-flops in the counter. Fig. 7-60 shows the decoding gates for the six states of the Johnson counter of Fig. 7-59.



Q_0	Q_1	Q_2	*Active gate*
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3
0	1	1	4
0	0	1	5

Fig. 7-60. Illustrating a decoding logic for a MOD-6 Johnson shift Counter.

As seen, each decoding gate has only two inputs, even though there are three flip-flops in the counter. This is because for each count, two of the three flip-flops are in unique combination of states. For example, the combination $Q_0 = Q_2 = 0$ occurs only once in the counting sequence, at the count of 0. Thus AND gate 0 with inputs $\overline{Q}_0$ and $\overline{Q}_2$ can be used to decode for this count. The same characteristic is shared by all the other states in the sequence.

Note. A Johnson shift counter requires fewer flip-flops than a ring shift counter but generally more than a binary counter. It has more decoding circuitry than a ring counter but less than a binary counter. Thus it sometimes represents a logical choice for certain applications.

Example 7-28. *Determine the frequency of the pulses at points W, X, Y and Z in the circuit of Fig.* 7-61.

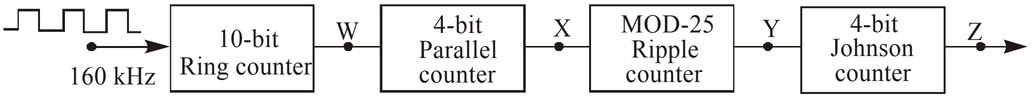


Fig. 7-61.

Solution. We know that a 10-bit ring shift counter is a MOD-10 counter. Therefore it divides the input frequency by 10. Thus the frequency at the W output,

$$f_W = \frac{160\,\text{kHz}}{16} = 16\ \text{kHZ}\ \textbf{Ans.}$$

Next, 4 – bit parallel counter is basically a MOD – 16 counter. This will divide its input frequency by 16. Therefore, the frequency at X output,

$$f_X = \frac{16\ \text{kHz}}{16} = 1\ \text{kHZ}\ \textbf{Ans.}$$

Next, MOD – 25 ripple counter will divide further its input frequency by a factor of 25. Therefore the frequency at Y output,

$$f_Y = \frac{1\ \text{kHZ}}{25} = \frac{1000}{25} = 40\ \text{Hz}$$

Finally, 4 – bit Johnson counter is basically a MOD – 8 counter. It will divide its input frequency by a factor of 8. Therefore frequency at Z output,

$$f_Z = \frac{40\,\text{HZ}}{8} = 5\ \text{Hz}\ \textbf{Ans.}$$

7-33. Design of Counters using K-maps

Sometimes counter is required that follows a sequence that is not counting binary; for example 000, 010, 101, 110, 000… Several methods are used for designing counters that follows arbitrary sequences. Generally *J-K*, *T* or *D* flip-flop are used. In synchronous counter, all of the flip-flops are clocked at the same time and input of each flip-flop in the counter must be at the correct level to ensure that each flip-flop goes to the correct state. For example, assume that the present state is ABC (101) and next state is ABC (01)1. When the next clock pulse occurs, the inputs of the flip-flops must beat the correct levels that will cause flip-flop *A* to change 1 to 0, flip-flop *B* from 0 to 1, and flip-flop *A* from 1 to 1 i.e., (No change)

Design procedure of counter with K-map is given below:

1. Obtain the state table from the given circuit information.
2. Determine the number of flip-flops needed.
3. Choose the type of flip-flops to be used.
4. From the state table, derive the circuit excitation table.
5. Use K-map or any other simplification method to drive the circuit flip-flop input functions.
6. Draw the logic diagram.

The counter with N flip-flops has maximum mod number 2^N. For example, 3-bit binary counter is a MOD-8 counter. The basic counter can be modified to produce MOD numbers less than 2^N by allowing the counter to skip those that are normally a part of the counting sequence.

Let's design a MOD-5 synchronous counter using *JK* flip-flop and implement it. Number of flip-flop required are

$$2^N \geq X$$
$$2^N \geq 5$$
$$N = 3$$

Three flip-flops are required assume that three flip-flops are *J-K* and are named as *A*, *B* and *C*. State diagram for MOD-5 counter are shown in Fig. 7-62 for the count 101, 110 and 111 the next state is not defined and is considered as don't care

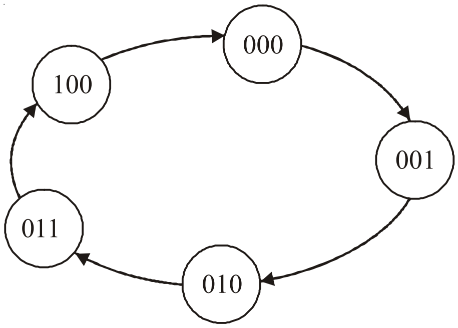


Fig. 7-62.

The excitation table for *JK* flip-flop is shown in Table 7-2.

Table 7-2.

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

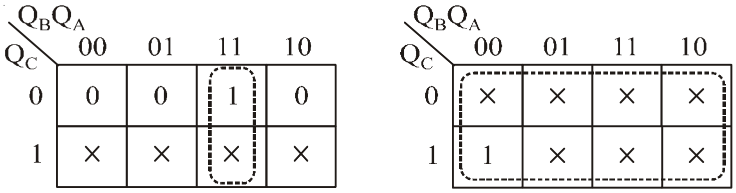
The state table for MOD-5 counter is shown in Table 7-3. This contains present state, next state and flip- flop inputs.

Table 7-3.

Present			*Next State*			*Flip-Flop Inputs*					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	0	0	0	×	1	0	×	0	×
1	0	1	×	×	×	×	×	×	×	×	×
1	1	0	×	×	×	×	×	×	×	×	×
1	1	1	×	×	×	×	×	×	×	×	×

Simplification of flip-flop function J_C, K_C,, J_B, K_B, J_A and K_A, with K-map is shown in Fig. 7-63. The functions are

$$J_C = Q_BQ_A$$
$$K_C = 1$$
$$J_B = Q_A$$
$$K_B = Q_A$$
$$J_A = \overline{Q}_C$$
$$K_A = 1$$



$Q_C \backslash Q_BQ_A$	00	01	11	10
0	0	0	1	0
1	×	×	×	×

$J_C = Q_BQ_A$

$Q_C \backslash Q_BQ_A$	00	01	11	10
0	×	×	×	×
1	1	×	×	×

$K_C = 1$

(a) For *C* Flip-Flop

$Q_C \backslash Q_BQ_A$	00	01	11	10
0	0	1	×	×
1	0	×	×	×

$J_B = Q_A$

$Q_C \backslash Q_BQ_A$	00	01	11	10
0	×	×	1	0
1	×	×	×	×

$K_B = Q_A$

(b) For *B* Flip-Flop

$Q_C \backslash Q_BQ_A$	00	01	11	10
0	1	×	×	1
1	0	×	×	×

$J_A = \overline{Q}_C$

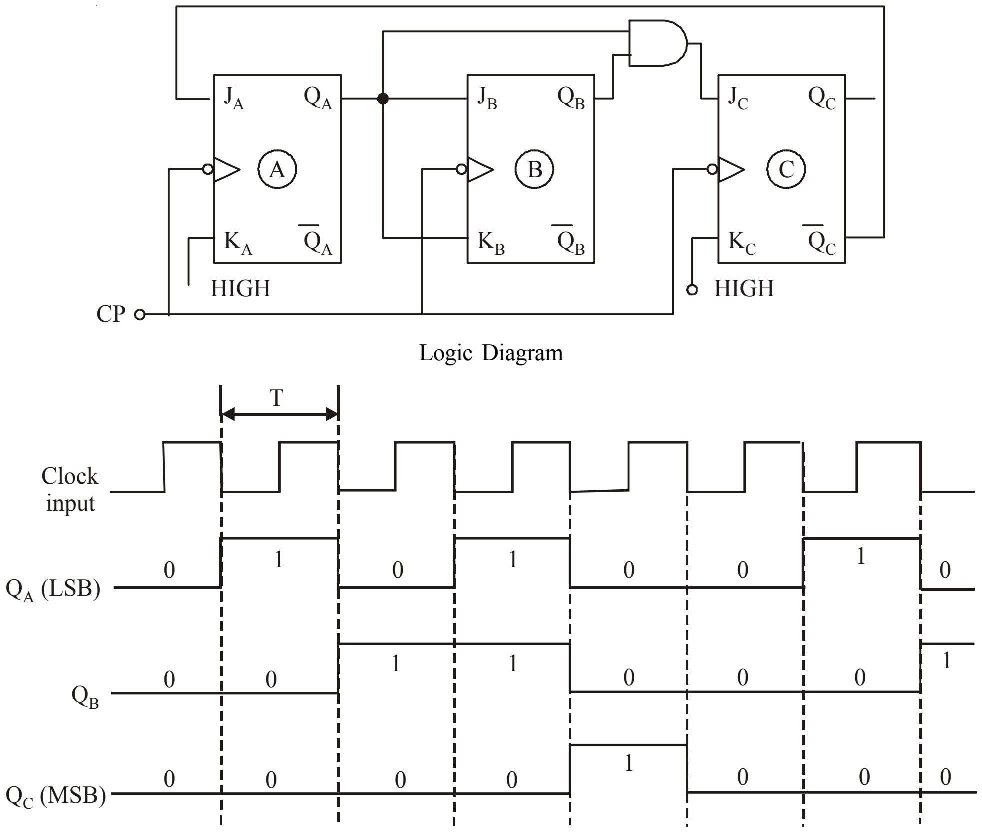
$Q_C \backslash Q_BQ_A$	00	01	11	10
0	×	1	1	×
1	×	×	×	×

$K_A = 1$

(c) For *A* Flip-Flop

Fig. 7-63.

Logic diagram and timing diagram of MOD-5 counter is shown in Fig. 7-64.



Logic Diagram

Timing Diagram

Fig. 7-64.

7-34. Integrated-Circuit Shift Register Counters

Strictly speaking, there are very few ring shift counters or Johnson shift counters that are available as ICs. This is due to the reason that it is relatively simple to take a shift-register IC and to wire it as either a ring shift counter or a Johnson counter. Some of the more commonly available CMOS Johnson-counter ICs are 74 HC 4017 and 74 HC 4022. These ICs include the complete decoding circuitry on the same chip as the counter.

7-35. Applications of Counters

Although there are several applications of the counters in digital systems yet the following are important from the subject point of view.

1. Frequency Counter and
2. Digital Clock

Now we shall discuss both these application one by one in the following pages.

7-36. Frequency Counter

An electronic circuit that can measure and display the frequency of an electrical signal is called a frequency counter. There are several methods of constructing a frequency counter but one of the simplest method is shown in Fig. 7-65. As seen from this figure, it contains a counter with its associated decoder / display circuitry and a two input AND gate. One of the inputs to an AND gate is the unknown frequency (f_x) while the other input is a sample pulse. The sample pulse controls the time duration (or interval) for which the unknown frequency pulses are allowed to pass through the AND gate.

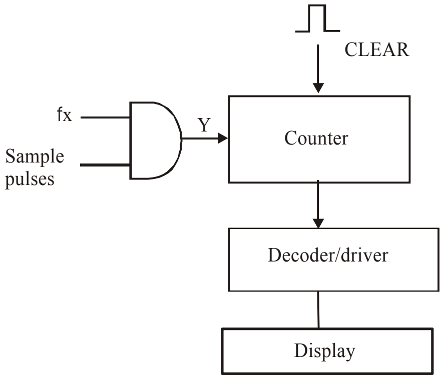


Fig. 7-65. Basic frequency counter circuit

In order to understand the operation of a basic frequency counter circuit, let us refer to the waveforms shown in Fig. 7-66. As shown in this figure, a CLEAR pulse is applied to the counter at t_0 to start the counter. Prior to t_1, the sample pulse waveform is LOW, therefore the output of the AND gate, Y is LOW. As a result, the counter will not be counting. The sample pulse goes HIGH from t_1 to t_2. This duration is called sampling interval. During this period, the unknown frequency pulses will pass through the AND gate and will be counted by the counter. After t_2 , the AND output goes LOW. Therefore the counter stops counting. Thus the counter will have counted the number of pulses that occurred during the sampling interval. This number is a measure of frequency of the pulse waveform.

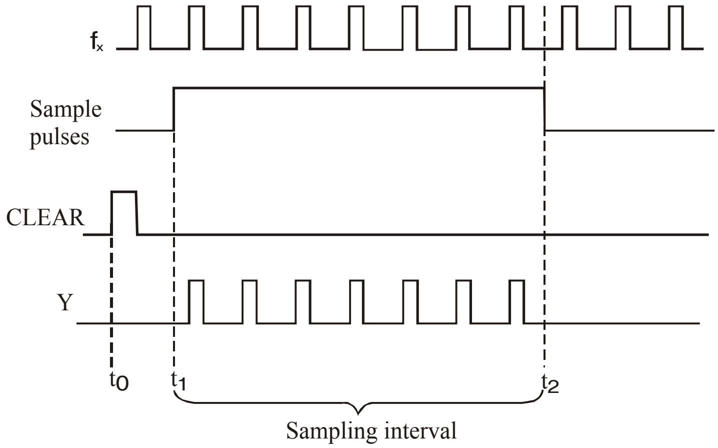


Fig. 7-66. Waveforms illustrating the operation of a basic frequency counter

It may be carefully noted that the accuracy of this method depends entirely on the duration of the sampling interval. This must be controlled very accurately.

Example 7-29. *An unknown frequency of* 5327 *pulses per second* (*pps*) *is applied to a counter through an AND gate as shown in Fig.* 7-67. *The counter is cleared to the* 0 *state prior to* t_1. *Determine the counter reading after a sampling interval of* (*a*) 1*s* (*b*) 0.1*s and* (*c*) 10 *ms.*

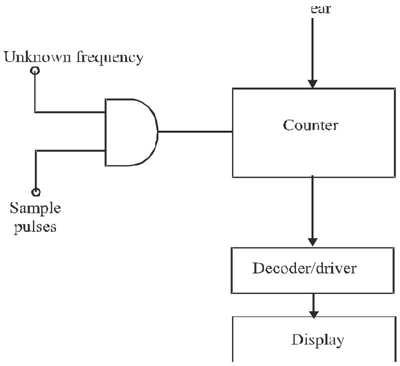


Fig. 7-67.

Solution.

(*a*) **Sampling interval = 1 s.** We know that within a sampling interval of 1 second, there will be 5327 pulses entering the counter and so after 't_2', the contents of the counter will read 5327.

(*b*) **Sampling interval = 0.1 s.** We know that with a 0.1 s sampling interval, the number of pulses passing through the AND gate into the counter will be counting 5327 pulse/s × 0.1 s = 532.7.

(*c*) **Sampling interval = 10 ms.** This means that either 532 or 533 pulses will be counted depending upon what part of a pulse cycle t_1 occurs on.

We know that with a sampling interval of 10 ms = 0.01 s, the counter will be counting 5327 pulses/s × 0.01 s = 53.27.

7-37. Digital Clock

Fig. 7-68 shows a basic block diagram of a digital clock operating from a 50 Hz frequency. As seen from this figure, a 50 Hz signal goes through a Schmitt trigger circuit. As a result, a square wave of 50

Hz frequency is produced. This 50 Hz or 50 pps (pulses per second) waveform is fed into a MOD-50 counter. This counter divides the 50 pps down to 1 pps. The 1 pps signal is fed into the "seconds" section. This section is used to count and display seconds from 0 through 59.

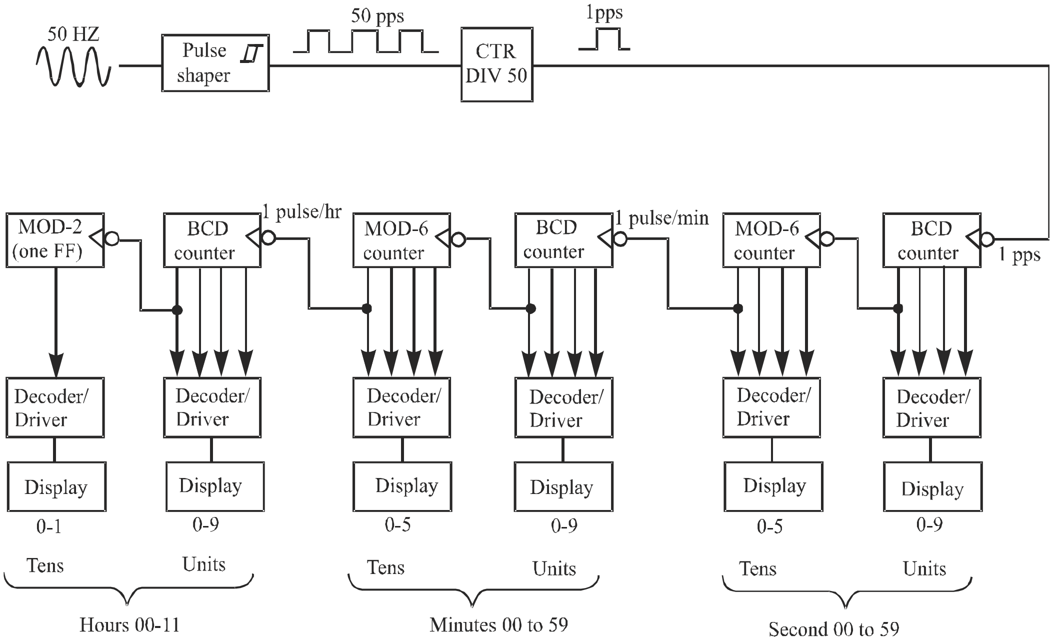


Fig. 7-68. Digital clock block diagram

The BCD counter (in the seconds section) advances one count per second. After 9 seconds the BCD counter resets (or goes to 0). This triggers the MOD-6 counter and causes it to advance one count. This continues for 59 seconds. At this point, the MOD-6 counter is at the 101_2 (i.e. 5_{10}) count and the BCD counter is at 1001_2 (i.e 9_{10}). As a result the display will read 59 seconds. The next pulse resets the BCD counter to 0. This in turn resets the MOD-6 counter to 0.

It may be noted that the output of the MOD-6 counter in the "seconds" section has a frequency of 1 pulse per minute (because the MOD-6 counter resets after every 60 seconds). This signal is fed to the "Minutes" section. This section counts and displays minutes from 0 through 59. Notice that the "Minutes" section is identical to "seconds" section and operates exactly in the same manner.

The output of the MOD-6 counter in the "Minutes" section has a frequency of 1 pulse per hour (because the MOD-6 counter resets after every 60 minutes). This signal is fed to the "Hours" section. This counts and displays hours from 1 through 12. It may be noted that the "Hours" section is different from the "seconds" and the "Minutes" section in the sense that it never goes to the "0" state.

The operation of the "Hours" section may be better understood with the help of a detailed circuitry shown in Fig. 7-69. As seen from the figure, it includes a BCD counter to count units of hours and a single flip-flop (MOD-2) to count tens of hours. The BCD counter is IC 74192 which can count up/down like 74193 that we studied earlier (in Art 7-20) except that it counts only between 0000_2 and 1001_2 However, 74192 in the "Hours" section is used to count up in response to the 1 pulse per hour signal coming from the "Minutes" section. Notice the presence of an INVERTER gate on the CP_U input. This gate is needed because 74192 responds to the rising edge of the input signal and we want it to respond to the falling edge that occurs when the "Minutes" section resets to '0'.

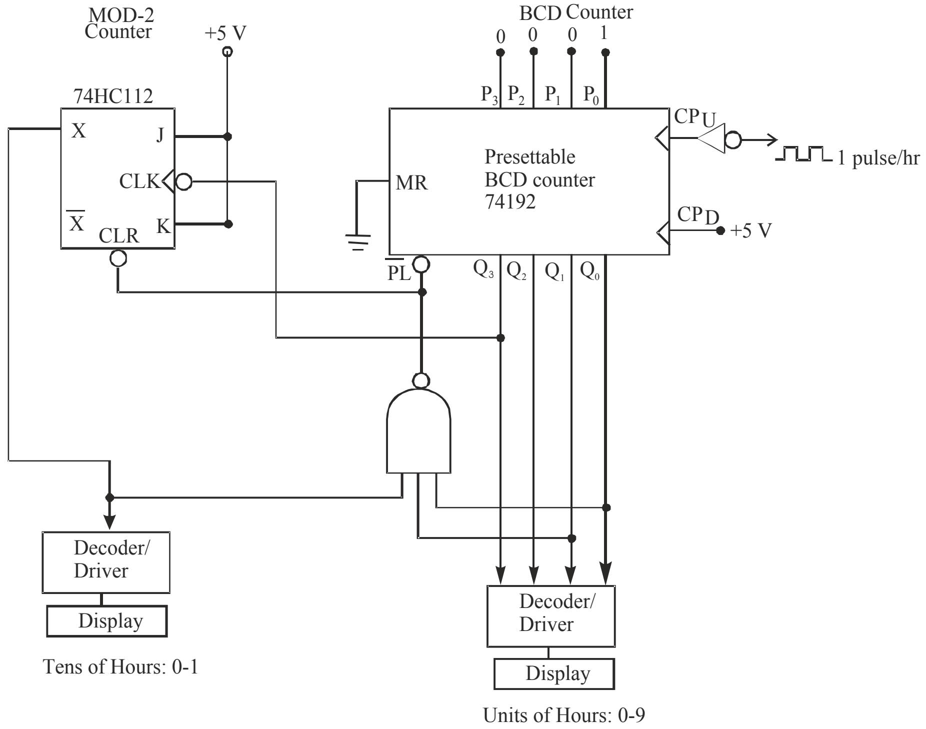


Fig. 7-69.

7-38. Integrated Circuit Shift Registers

Four-bit and eight-bit shift registers are commonly available in integrated circuit packages. These registers may broadly be classified into the following four types :

1. Parallel in / parallel out
2. Serial in / serial out
3. Parallel in/ serial out
4. Serial in / parallel out

Now we will study a representative IC from each of the above four categories, one by one in the following pages.

7-39. The 74/74 6-bit Parallel In / Parallel Out Shift Register

Fig. 7-70 shows the logic diagram for the IC 74174. This IC is a 6-bit register that has parallel inputs D_0 through D_5 and parallel outputs Q_0 through Q_5 . Parallel data are loaded into the register on the rising edge of clock input (CP). A master reset input ($\overline{MR}$) can be used to reset asynchronously all of the register flip-flops to 0.

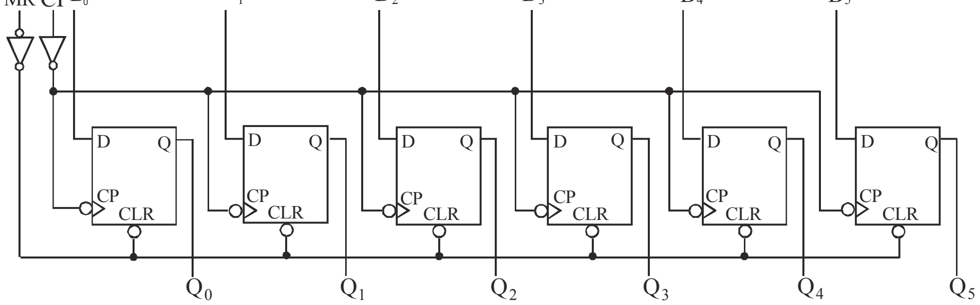


Fig. 7-70. Logic diagram for 74174

Fig. 7-71 shows the logic symbol for the IC 74174. The logic symbol is used in digital circuit diagrams to represent the circuitry of Fig. 7-70. It may be carefully noted that the IC 74174 is normally used for synchronous parallel data transfer where by the logic levels present at the data (i.e. D) inputs are transferred to the Q-outputs when a rising edge occurs at the clock (CP). However this IC can also be wired for serial data transfer as discussed below.

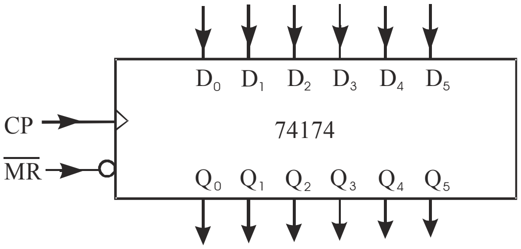


Fig. 7-71.

Fig. 7-72 shows the connections of a 74174 in order to operate it as a serial shift register. The data shifts on the rising edge of the clock pulse (CP) as follows :

$$D_0 \rightarrow D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow D_4 \rightarrow D_5$$

In other words, serial data will enter at D_0 and will output at Q_5.

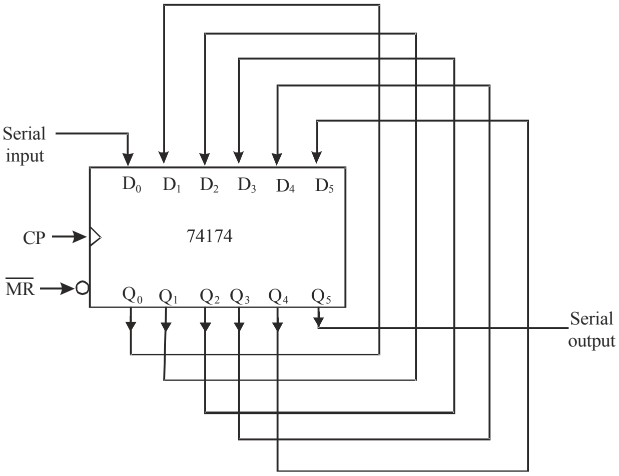


Fig. 7-72. Illustrating the connections for a 74174 as a serial shift register

Note. It is possible to connect two 74 ALS 174s to operate as a 12-bit shift register. This can be achieved by connecting a second 74 ALS 174 IC in such a way that Q_5 of the first IC is connected to D_0 of the second IC. Besides this, the clock pulse (CP) inputs of two ICs are also connected together so that they will be clocked from the same signal.

7-40. The 4731B 64-bit Serial In / Serial Out Shift Register

The IC 4731 B is a CMOS quad 64-bit serial in / serial out shift register. 'Quad' means the IC contains four identical 64-bit shift registers on one chip. Fig. 7-73 shows the logic diagram on one of the 64-bit registers. As seen from the figure, the shift register has a serial input (D_S), a clock input ($\overline{CP}$) that responds to the falling edge of clock pulses, and a serial output from the last flip-flop (Q_{63}).

It may be carefully noted that the output (Q_{63}) of the shift register shown in Fig. 7-73, goes through a buffer circuit. The buffer circuit is shown as a triangle symbol with no inversion bubble. A buffer does not change 0 signal's logic level but it is used to provide a greater current-output capability than normal.

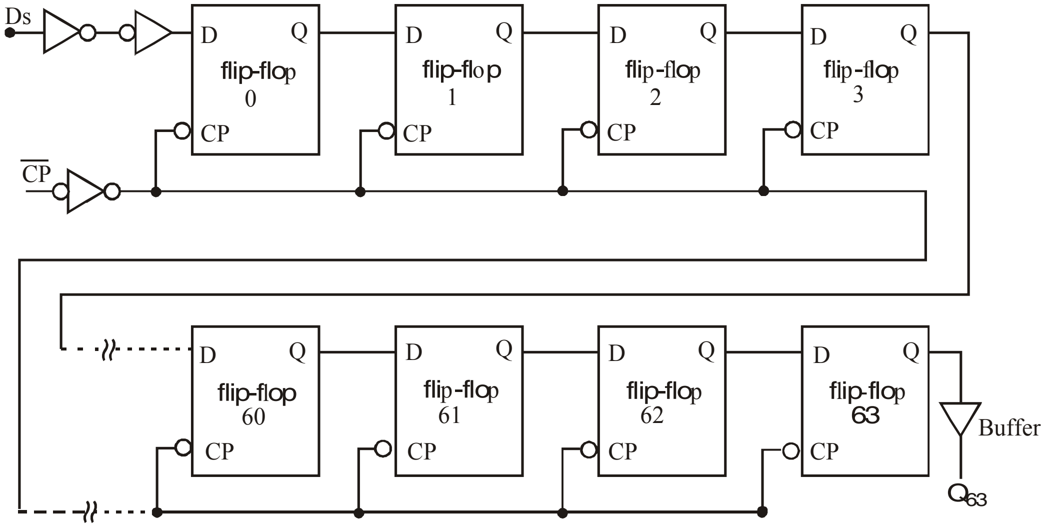
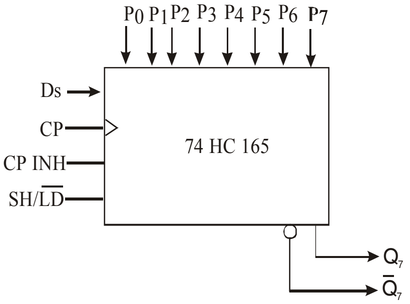


Fig. 7-73. Logic diagram of a 64-bit shift register

Note. A shift register is used to delay a digital signal by an integral number of clock cycles. The digital signal is applied to the shift register's serial input. It is shifted through the shift register by successive clock pulses until it reaches at the end of shift register where it appears at the output signal.

7-41. The 74165 8-bit Parallel In / Serial Out Shift Register

Fig. 7-74 (a) shows the logic symbol for the IC 74 HC 165. This IC is an 8-bit parallel in / serial out register. It has serial data entry via D_S and asynchronous parallel data entry via P_0 through P_7 . The register contains eight flip-flops with their outputs Q_0 through Q_7. All the flips are internally connected as a shift register but the only accessible flip-flop outputs are Q_7 and $\overline{Q_7}$. Further CP is the clock input used for the shifting operation. The clock inhibit input (CP INH) is used to inhibit the effect of CP input. The shift / load ($SH/\overline{LD}$) input controls the type of operation i.e. shifting or parallel loading.



Function Table

Inputs			
SH/$\overline{LD}$	***CP***	***CP INH***	***Operation***
L	X	X	Parallel load
H	H	X	No change
H	X	H	No change
H	↑	L	Shifting
H	L	↑	Shifting

H = High level
L = Low level
X = don't care
↑= Rising edge

(*a*) logic symbol (*b*) function table

Fig. 7-74. Parallel- in / Serial-out shift register

Fig. 7-74 (b) shows the list of various input combinations and the corresponding operation being performed. As seen from this table, when $SH/\overline{LD}$ is LOW, while CP and CP INH are X (*i.e.* does not matter whether HIGH or LOW), the device performs parallel load operation. On the other hand, if $SH/\overline{LD}$ is HIGH, CP INH is LOW, then with the arrival of rising edge of clock pulse at CP input, the register performs shifting operation.

Notice that shifting operation can also be performed if $SH/\overline{LD}$ is HIGH, CP is LOW. The clock input is applied at CP INH input.

7-42. The 74164 8-bit Serial In / Parallel Out Shift Register

Fig. 7-75 shows the logic diagram for the 74164. As seen from this figure, it is an eight-bit serial in / parallel out shift register with each flip-flop output externally accessible. Notice the presence of an AND gate just before the serial data input to the flip-flop 0. This AND gate combines inputs A and B to produce the serial data input to flip-flop 0.

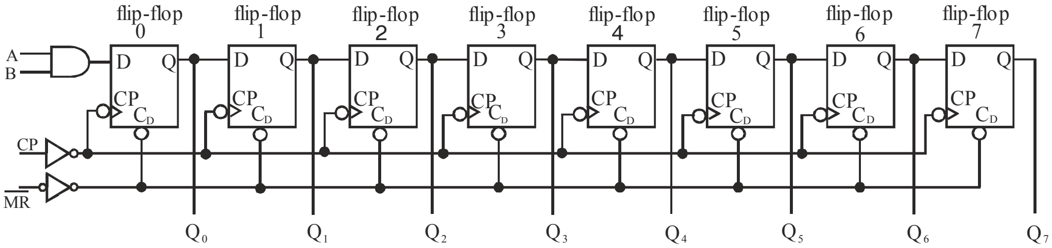


Fig. 7-75. Logic diagram of 74164 an eight-bit serial in/parallel out shift register

Notice that the shift operation occurs on the rising edge of the clock input (CP). The master reset ($\overline{MR}$) input provides asynchronous resetting of all flip-flops on a logic LOW level. Fig. 7-76 shows the logic symbol for the IC 74164. Notice the presence of '&' symbol within the block. This symbol is used to indicate that the A and B inputs are ANDed inside the IC and the result is applied to the D input of flip-flop '0'.

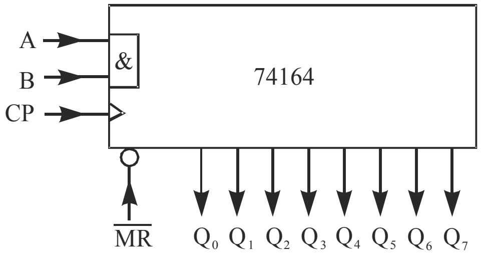
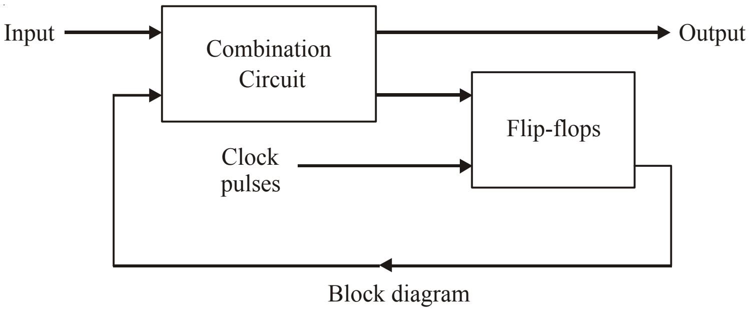


Fig. 7-76. Logic symbol of the 74 ALS 164-eight-bit serial in / parallel out shift register

7-43. Synchronous Sequential Circuit

The synchronous sequential circuit changes their states and output values at fixed points of time, which are specified by the rising or falling edge of a free-running clock signal. The synchronization of the sequential circuit is achieved by a timing device called a clock generator, it generates a periodic train of clock pulse. Fig. 7-77, shows the synchronous clocked sequential circuit.



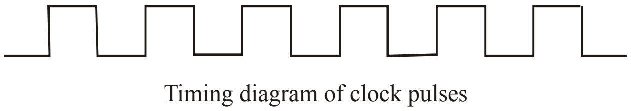


Fig. 7-77.

7-44. Synchronous Sequential Circuit Models

Synchronous sequential circuit is also known as a finite state machine (FSM). The most general model of a sequential circuit has input, output and internal states. Depending upon the way the external outputs are obtained from the circuit, there are two different models of sequential circuits.

1. Mealy Model
2. Moore Model

Mealy Model

In mealy model the next state is a function of the present state and the present inputs. Its output is also a function of the present state and the present inputs. Fig. 7-78 shows the mealy model. The next state and the output is defined as,

$$\text{Next state} = F_1\,(\text{Present state, inputs})$$
$$\text{Outputs} = F_2\,(\text{Present state, inputs})$$

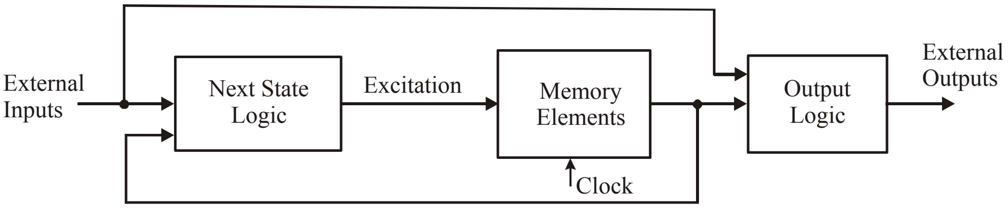


Fig. 7-78.

Moore Model

Fig. 7-79 shows the moore model. In this model the next state is the function of the present state and present inputs. The output function of the Moore model is only the functions of the present state and independent of the inputs. The next state and the output is defined as,

$$\text{Next state} = F_1\,(\text{Present state, inputs})$$
$$\text{Outputs} = F_2\,(\text{Present state})$$

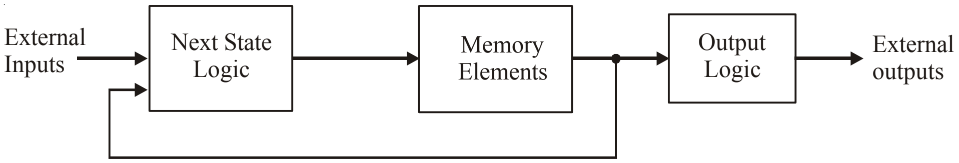


Fig. 7-79.

7-45. Analysis of Clocked Sequential Circuit

The behavior of a sequential circuit is determined from the inputs, the outputs, and the state of flip-flop. In the both models the memory element are flip-flop. The inputs are clock signal. The flip-flop are clocked simultaneously with the clock signal. The outputs and the next state of a clocked sequential circuit depend upon the external inputs and the present state of the circuit. For synchronous sequential circuits, the sequence of inputs, present and next states, and output can be represented by a state table or a state diagram.

The operation of a clocked sequential circuit can be described in the form of a diagram, known as state diagram, a table known as state table, or in the form of a flow chart, known as algorithmic state machine (ASM) chart.

7-46 State Diagram

It is a directed graph consisting of vertices (node) and directed areas between the nodes. Node represent the every state of the circuit. A node is represented as a circle with the name of the state written inside the circle. The directed arces represent the state transitions. With the circuit in any one state, at the occurrence of a clock pulse, there will be a state transition to the next state and there will be an output, both in accordance with the requirements of the circuit. This state transition is represented by a directed line from the node corresponding to the present state and terminating on the node corresponding to the next state. The labels are put on the directed arces specifying the inputs and outputs separated by a slash (/).

Let us consider a state diagram shown in Fig. 7-80. In this when the circuit is in state *A*, are input 1 causes the circuit to make a transition to the next state *B* and gives an output 0. A represent the present state and *B* represent the next state. This two state *A* and *B* are connected by an directed arc from *A* to *B*. The label put as 1/0, input and output separated by a slash (/).

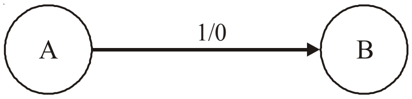


Fig. 7-80.

For a circuit with single input, when the circuit is in any state, the input can be 0 or 1. For each possibility of the input, there will be a directed arc. Thus two arcs emanate from each node, one each for a 0 and for a 1 input. Thus for a *n*-input machine, 2^n arcs will emanate from each node. This is shown in Fig. 7-81.

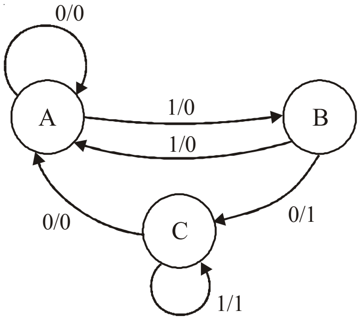
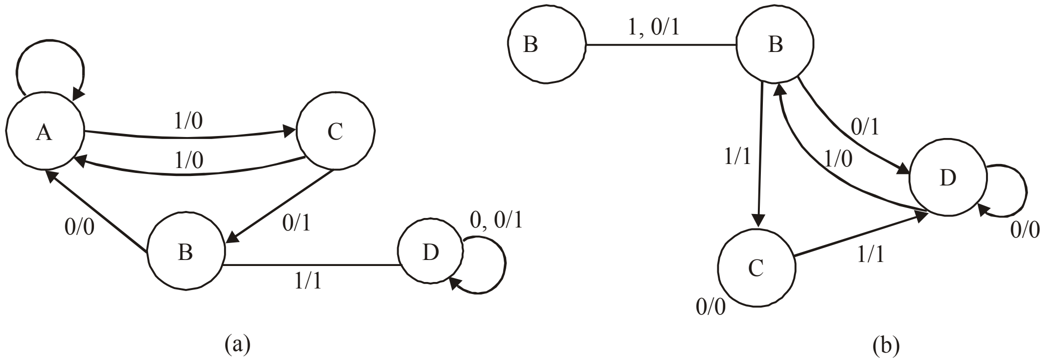


Fig. 7-81.

In some case a state may be terminal state. The corresponding node in the state diagram may be a sink node or a source node. A node is a sink node if there is no outgoing arc from it and terminate in other node. As shown in Fig. 7-82(a), node *D* is a sink node. Whatever may be the input (0 or 1), the arc emanating from it terminates on itself and there is no arc emanating from *D* that terminates in any other node. No state is accessible from a sink state.

The node is known as a source node if there are no arces which emanate from other node terminating in it as shown in Fig. 7-82, node A is a source node.



(a)

(b)

Fig. 7-82.

7-47. State Table of Synchronous Sequential Circuit

State table is a tabular form of describing the operation of a synchronous circuit. Each row of the state table represent the state of the circuit and each column represent the combination of external inputs. The table consists of four section labeled present state, input, next state and output. The present section shows the state of flip-flops at any given time *t*. The input section gives a value for each possible present state. The next selection shows the states of the flip-flops one clock period later at time $t + 1$. The output section gives the output values for each present state. The derivation of state table consists of first listing all possible binary combinations of present state and inputs.

Let us construct the table for the state diagram of Fig. 7-81. There are three states in the circuit *A*, *B* and *C*. The state will contain three rows. There is one input (*X*) and one output (*Y*) variable. There is one column for each value of *X*, i.e. $X = 0$ and $X = 1$. Each entry of table contains two pieces of information: next state and output state separated by a comma. The first entry is for the next state and after it is output.

Let us start making entries in the state table starting from the state *A*. When the circuit is in state *A*, the external input present is 0, in response to a clock pulse the state does not change, i.e., the next state is also *A* and the output in 0. This is written in the first row and first column a *A*, 0. If the present input is 1 while the circuit is in state *A*, a clock pulse will cause next state to be *B* and output = 0. The corresponding entry in the first row second column will be *B*, 0.

Similarly, take the state *B* and find out the next state and resulting output when $X = 0$ and when $X = 1$ and the corresponding entries are made in the second row under the appropriate columns. Sam procedure will give entries for the third row corresponding to the state *C*.

Table 7-4. State Table

Present State *PS*	*Next State, Output* *NS, Y*	
	X = 0	*X = 1*
A	A, 0	B, 0
B	C, 1	A, 0
C	A, 0	C, 1

7-48. State Reduction of Synchronous Sequential Circuit

A large portion of the digital electron subject is concerned with finding algorithms for minimizing the number of flip-flops and gates in sequential circuits.

The reduction of the number of flip-flops in a sequential circuit is referred to as the state-reduction problem. State reduction algorithms are concerned with procedure for reducing the number of states in a state table while keeping the external input-output requirements unchanged. Since m flip-flops produce 2^m states, a reduction in the number of states may result in a reduction in the number of flip-flops.

Two states are said to be equivalent if, for each input condition, they give exactly the same output and go to the same next state. In a state table, when two states are found to be equivalent, one of them is eliminated without altering the input-output relations. This process is referred to as state-reduction. Using the concept of equivalent states and state-reduction, all possible equivalent states are determined and the reduced state table is obtained which will contain the minimum number of states. This process minimizes the number of states.

Lets discuss the state reduction procedure with the help of table 7-5.

Table 7-5. State Table

Present State	*Next State*		*Output*	
	***X* = 0**	***X* = 1**	***X* = 0**	***X* = 1**
a	c	b	0	0
b	d	c	0	0
c	g	d	1	1
d	e	f	1	0
e	f	a	0	1
f	g	f	1	0
g	f	a	0	1

We see that from the initial state *e*, the next state is *f* for *X* = 0 and it is a for *X* = 1. The output is 0 and 1 for *X* = 0 and *X* = 1 respectively. Similarly, from the initial state *g* also, the next state is *f* for a for *X* = 0 and *X* = 1 respectively and the output is 0 and 1 for *X* = 0 and *X* = 1 respectively. From this we conclude that the two states *e* and *g* are equivalent, since for each input condition, they go to the same next state and give same output. We can eliminate one of them, say *g*. Thus *g* is replaced by *e* wherever it occurs.

After replacing *g* and *e* in the state table, we notice that the states *d* and *f* are equivalent. Thus one of them, say *d* can be eliminated.

The effect of eliminating state is shows in Table and the reduced state table is shown in Table 7-5. The reduced state table contains 5 state which is the minimum number of states required to implement the circuit represented by the given state Table 7-6.

Table 7-6. Effects of Eliminating States

Present State	***Next State***		***Output***	
	***X* = 0**	***X* = 1**	***X* = 0**	***X* = 1**
a	*c*	*b*	0	0
b	(*d*)*f*	*c*	0	0
c	(*g*)*e*	(*d*)*f*	1	1
d	*e*	*f*	1	0
e	*f*	*a*	0	1
f	(*g*)*e*	*f*	1	0
g	*f*	*a*	0	1

Table 7-7. Reduced State Table

Present State	***Next State***			
	***X* = 0**	***X* = 1**	***X* = 0**	***X* = 1**
a	*c*	*b*	0	0
b	*f*	*c*	0	0
c	*e*	*f*	1	1
e	*f*	*a*	0	1
f	*e*	*f*	1	0

7-49. State Assignment of Synchronous Sequential Circuit

For designing a clocked sequential circuit, the requirement may be specified as a set of statements. State diagram is constructed from the set of statements and state table can be prepared from the state diagram. We use some letter symbols such as A, B, $C\ldots$etc for the states. The exact number of states is also usually not known, but using the set of statements, arbitrary number of states may be chosen to satisfy the given requirements of the circuit. There are two issues involved in the design of clocked sequential circuits. There are given below:

1. The number of states must be minimum possible so as to be able to design a system with the minimum number of flip-flops.
2. Combinational circuits are required to generate the excitation functions and the output.

Each option will lead the different logic expressions for the flip-flop excitations and output, states.

7-50 Rules for State Assignment

There is no general procedure for the state assignment which produces the most cost effective design of the resulting combinational circuit, however, trial and error attempts at making state assignments are not practically feasible. The following thumb ruler will help in generating simple combinational circuits.

Rule 1

Adjacent codes should be assigned to the states having the same next state for:

1. each input combination
2. different input combinations, if the next state can also be assigned adjacent codes
3. some of the input combinations, not all

Rule 2

Adjacent codes should be assigned to the next state (*S*) of every present state.

Rule 3

Adjacent codes should be assigned to states that have the same outputs.

Lets take the Table 7-8 to explain the state assignment procedure in detail.

Table 7-8. State Table

Present State PS	***Next state, Output*** ***X* = 0**	***N, S, Y*** ***X* = 1**
A	*B*, 1	*B*, 0
B	*C*, 0	*D*, 1
C	*E*, 1	*F*, 0
D	*F*, 0	*E*, 1
E	*G*,0	*A*, 0
F	*A*, 0	*G*,0
G	*B*, 0	*B*, 0

There are seven states in this circuit. Number of required flip-flops are three. ($2^n = 7$, when *n* is the no. of flip-flops). Application of the rules of assignment is given below:

Rule 1

1. Next state from the present states *A* and *G* is same. It is *B*, the states *A* and *G* should be assigned adjacent codes.
2. From the present state *C*, the next state is *E* for *X* = 0 and *F* for *X* = 1, whereas from the present state *D*, the next states are *F* and *E* for *X* = 0 and *X* = 1 respectively. Therefore, *C* and *D* may be assigned adjacent codes.
3. Similarly *E* and *F* may be assigned adjacent codes.

Rule 2

From the state *B*, the next states are *C* and *D*, therefore, *C* and *D* may be assigned adjacent states.

Rule 3

The outputs are same for the present states *A* and *C*; *B* and *D*, may be assigned adjacent codes.

Now the state assignment is to be made so that as many as possible of these adjacencies can be accommodated. For this assignment map is prepared. Let us assign the state 000 to A, the G must be assigned an adjacent state. There are three possible adjacent states. Any one of them can be assigned to G, since G is not required to be adjacent to any other state. Similarly, all the other assignments are made and the resulting assignment map is show in Fig 7-83. The binary states are given below.

$A = 000$

$B = 111$

$C = 100$

$D = 101$

$E = 001$

$F = 011$

$G = 010$

$Q_3 \backslash Q_2Q_1$	00	01	11	10
0	A	E	F	G
1	C	D	B	

Fig. 7-83.

7-51. State Equivalence and Minimisation

The two states are said to be equivalent if, for each input condition, they give exactly the same output and go to the same next state. In a stable table, when two states are found to be equivalent, one of them is eliminated without altering the input-output relations. This process is referred to as state-reduction. With the process of equivalent states and state reduction, all possible equivalent states are determined and reduced form of state table is obtained. This process minimizes the number of states.

7-52. Design Procedure of Synchronous Sequential Circuit

Synchronous sequential circuit is made up of flip-flips and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding a combinational gate structure that, together with the flip-flops, produces a circuit that fulfills the stated specifications. The number of flip flops is determined from the number of states needed in the circuit. For the design of any clocked Sequential circuit, the following procedure is used.

1. The word description of the circuit behavior is stated. This may be accompanied by a state diagram, a timing diagram, or other information.
2. From the given information about the circuit, obtain the state table.
3. The number of state is reduced by state-reduction methods.
4. Assign binary values to each state if the state table obtained in step 2 or 3 contains lether symbols.
5. Determine the number of flip-flops needed and assign a letter symbol to each.
6. Chose the type of flip-flop to be used.
7. From the state table, derive the circuit excitation and output table.
8. Using the map or any other simplification method, derive the circuit output functions and the flip flop input functions.
9. Draw the logic diagram.

From the table 7-8, we explain the design procedure in more detail. The next step in the design is to assign binary values to the states. There are 5 states, so the required number of flip-flop is 3. Three flip-flops are FF2, FF1 and FF0. There outputs are Q_2, Q_1 and Q_0 respectively. When Q_2 value is the MSB and Q_0 as LSB.

From the state assignment, we find that,

1. The next states are same for the states *c* and *f* for $X = 0$ and 1, therefore states *c* and *f* should be assigned adjacent.

2. For states b and e, the next state is f for $X = 0$, therefore, the states b and e should be assigned adjacent codes.
3. Assume $a = 000$ as the initial state, the map is shown in Fig. 7-84.

Present state	***Next state***		***Output***	
PS	$X = 0$	$X = 1$	$X = 0$	$X = 1$
a	*c*	*b*	0	0
b	*f*	*c*	0	0
c	*e*	*f*	1	1
e	*f*	*a*	0	1
f	*e*	*f*	1	0

$a = 000$
$b = 011$
$c = 010$
$e = 111$
$f = 110$

Q_0 \ Q_2Q_1	00	01	11	10
0	a	c	f	
1		b	e	

Fig. 7-84.

The state transition and output table is constructed next. From the state a (000) the state is $c = 010$. When $X = 0$. The output corresponding to this is $Y = 0$. These entries are made in the first row. Similarly all the entries are made in the table.

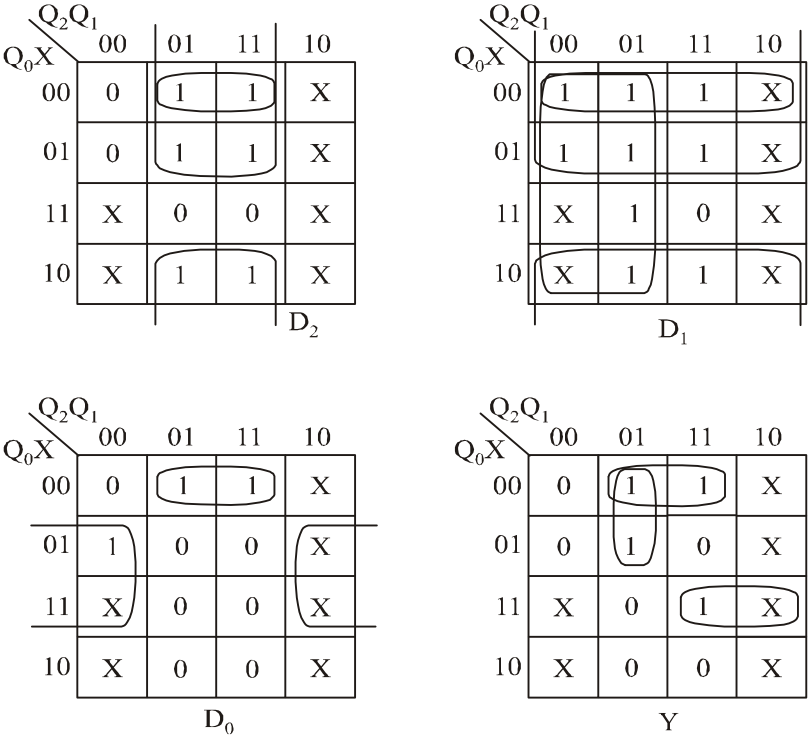
The D type of flip-flops have been chosen here, therefore, it is not necessary to construct excitation table. For D type flip-flops Q_2, Q_1 and Q_0 values are same as the excitation values of D_2, D_1 and D_0 respectively. The state transition and output table is given in Table 7-8..

Table 7-9. State Transition and Output Table

Present state			***Input***	***Next state***			***Output***
Q_2	Q_1	Q_0	X	Q^*_2	Q^*_1	Q^*_0	Y
0	0	0	0	0	1	0	0
0	1	1	0	1	1	0	0
0	1	0	0	1	1	1	1
1	1	1	0	1	1	0	0
1	1	0	0	1	1	1	1
0	0	0	1	0	1	1	0
0	1	1	1	0	1	0	0
0	1	0	1	1	1	0	1
1	1	1	1	0	0	0	1
1	1	0	1	1	1	0	0

Next we have to find out the minimum logic expressions for D_2, D_1, D_0 and Y in terms of Q_2, Q_1, Q_0 and X. The K-maps are constructed for D_2, D_1, D_2 and Y and are minimised. The K-maps are shown in Fig. 7-85.

There are eight states possible using three bits, out of which only five states are required for this circuit. The other three states do not exist are therefore, taken as don't cases (X's)



Q_0X \ Q_2Q_1	00	01	11	10
00	0	1	1	X
01	0	1	1	X
11	X	0	0	X
10	X	1	1	X

D_2

Q_0X \ Q_2Q_1	00	01	11	10
00	1	1	1	X
01	1	1	1	X
11	X	1	0	X
10	X	1	1	X

D_1

Q_0X \ Q_2Q_1	00	01	11	10
00	0	1	1	X
01	1	0	0	X
11	X	0	0	X
10	X	0	0	X

D_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	1	1	X
01	0	1	0	X
11	X	0	1	X
10	X	0	0	X

Y

Fig. 7-85. K-maps for D_2, D_1, D_0 and Y

The minimised expressions are

$$D_2 = Q_1\overline{Q}_2 + Q_1\overline{X}$$

$$D_1 = \overline{Q}_2 + \overline{Q}_0 + \overline{X}$$

$$D_0 = \overline{Q}_1 X + Q_1\overline{Q}_0\overline{X}$$

$$Y = Q_1\overline{Q}_0\overline{X} + \overline{Q}_2 Q_1\overline{Q}_0 + Q_2 Q_0 X$$

The complete circuit is shown in Fig. 7-86.

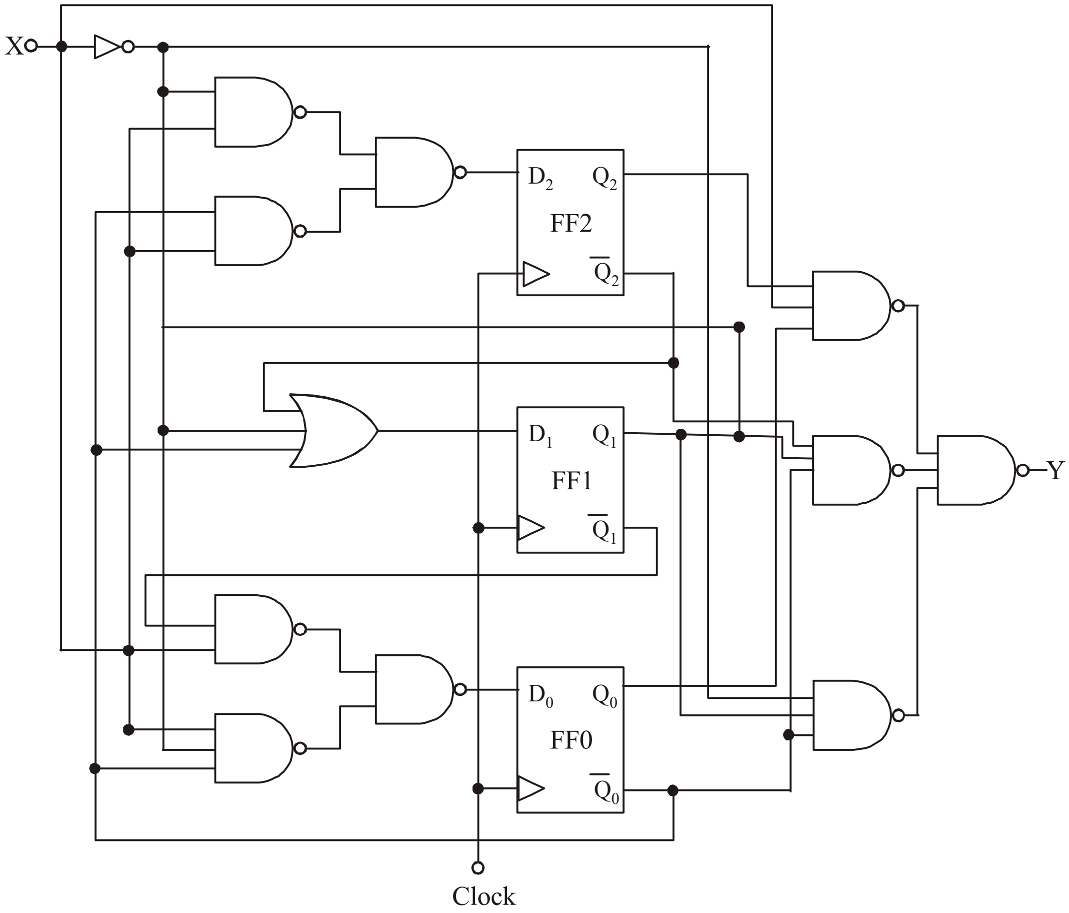


Fig. 7-86. Sequential Circuit

7-53. Asynchronous Sequential Circuit

A sequential circuit is specified by a time sequence of inputs, outputs and internet states. Asynchronous sequential circuit do not use clock pulses. The change in internal state occurs when there is a change in the input variables. The memory element in asynchronous sequential circuits are either unclocked flip-flops or time delay elements. The memory capability of a time-delay device depends on the finite amount of time it takes for the signal to propagate through digital gates. An asynchronous sequential circuit quite after resemble a combinational circuit with feedback.

This circuit are useful in a variety of applications. They are used when speed of operation is important, especially in those cases where the digital system must respond quickly without having to wait for a clock pulse. The communication between two units, each having it own independent clock, must be done with asynchronous circuits.

Block diagram of asynchronous circuit is shown in Fig. 7-87. It consists of a combinational circuit and delay elements connected to from feedback path. There are *n* input variables, *m* output variables and *K* internal states. The delay element are short-term memory for the sequential circuit. The present-state and next state variables in asynchronous sequential circuits are customarily called secondary variables and excitation variables, respectively. The excitation variables should not be confused with the excitable table used in the design of clocked sequential circuit.

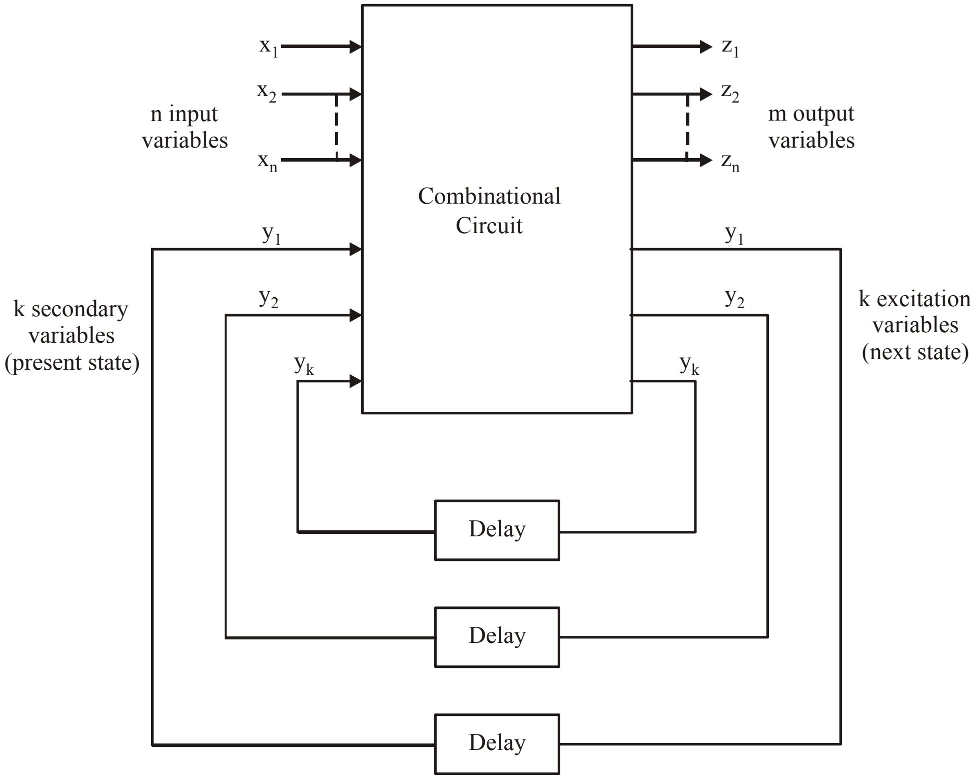


Fig. 7-87. Block diagram of an Asynchronous Sequential Circuit.

When an input variable changes in value, the *Y* secondary variables do not change instantaneously. It takes a certain amount of time for the signal to propagate from the input terminals, through the combinational circuit, to the *Y* excitation variables, which generate new values for the next state. These values propagate through the delay elements and becomes the new present state for the secondary variables.

In the steady state condition, *y's* and *Y's* are same but during transition they are not same. The system is stable if circuit reaches a steady-state condition with $y_i = Y_i$ for $i = 1, 2, ... k$. Otherwise, the circuit is in a continuous transition and is said to be unstable.

7-54. Asynchronous Sequential Machine Mode

There are two modes of operations of asynchronous sequential machines depending upon the type of input signals. These are given below:

1. Fundamental Mode

In this fundamental mode circuit all the input signals area considered to be levels. Fundamental mode operation assumes that the input signals will be changed only when the circuit is in a stable state and that only one variable can change at a given time.

2. Pulse Mode

In pulse mode circuits all input signal are considered to be pulses. In this mode of operation the width of the input pulses is critical to the circuit operation. The input pulse must be long enough for the circuit to respond to the input but it must be so long as to be present even after new state is reached. In such a situation the state of the circuit may make another transition.

Only one input is allowed to have pulse present at any time. This means that when pulse occurs on any one input, while the circuit is in stable state, pulse must not arrive at any other input.

7-55. Analysis of Asynchronous Sequential Machines

The analysis of asynchronous sequential circuits consists of obtaining a table or a diagram that describes the sequence of internal states and outputs as a function of changes in the input variables.

The analysis procedure will be presented by means of three specific examples. The first example introduces the transition table, the second defines the flow table, and the third investigate the stability of asynchronous sequential circuits.

7-56. Transition Table of Asynchronous Sequential Circuit

An example of an asynchronous sequential circuit with only gate is shown in Fig. 7-88. The diagram shows two feedback loops from the OR gate outputs back to the AND gate inputs.

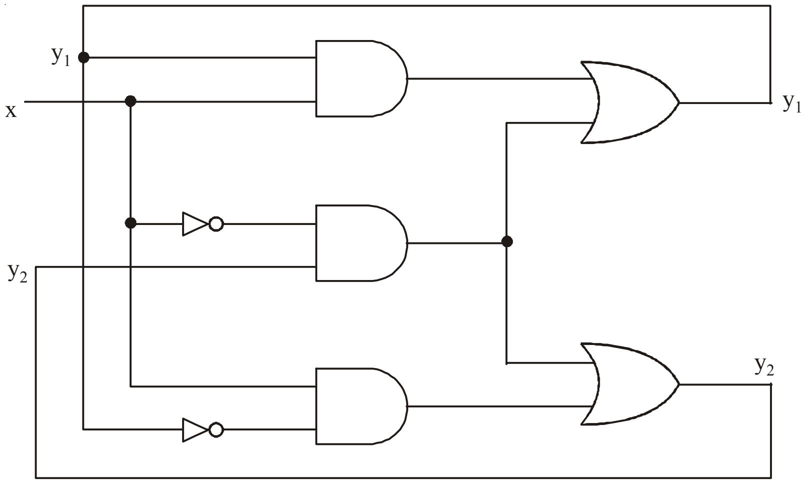


Fig. 7-88.

The circuit consists of one input variable x and two internal states. The internal states have two excitation variables, Y_1 and Y_2, and two secondary variables y_1 and y_2. The delay associated with each feedback loop is obtained from the propagation delay between each y input and its corresponding Y output. Each logic gate in the path introduces a propagation delay of about 2 to 10 ns.

The analysis of the circuit states with a consideration of the exciation variable as outputs and secondary variables as inputs. From the logic diagram the excitation variables are

$$Y_1 = xy_1 + x' y_2$$

$$Y_2 = xy'_1 + x'y_2$$

We plot the logic Y_1 and Y_2 in a Map, shown in Fig. 7-89.

y_1y_2 \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

(a) Map for $y_1 = xy_1 + x'y_2$

y_1y_2 \ x	0	1
00	0	1
01	1	1
11	1	0
10	0	0

(b) Map for $y_2 = xy'_1 + x'y_2$

Fig. 7-89.

The transition table is shown in Fig. 7-90, is obtained from the maps by combining the binary values in corresponding squares. The transition table shows the values of $Y = Y_1 Y_2$ inside each square. The first bit of Y is obtained from the value of Y_1, and the second bit is obtained from the value of Y_2 in the same square position. For a state to be stable, the secondary variable must match the excitation variables i.e., the value of Y must by the same as that of $y = y_1 y_2$. Those entries in the transition table where $Y = y$ are circled to indicate a stable condition. A uncircled entry represents an unstable state.

$y_1y_2 \backslash x$	0	1
00	$\enclose{circle}{00}$	01
01	11	$\enclose{circle}{01}$
11	$\enclose{circle}{11}$	10
10	00	$\enclose{circle}{10}$

Fig. 7-90. Transition Table

The square for $x = 0$ and $y = 00$ in the transition table shows that $Y = 00$. Since Y represents the next values of y, this is a stable condition. If x changes from 0 to 1 while $y = 00$, the circuit charges the value of Y to 01. This represents a temporary unstable condition because Y is not equal to the present value of y. What happens next is that as soon as the signal propagates to make $Y = 01$, the feedback path in the circuit causes a change in y to 01. This is manifested in the transition table by a transition from the first row ($y = 00$) to the second row, where $y = 01$. Now that $y = Y$, the circuit reaches a stable condition with an input of $x = 1$. In general, if a change in the input takes the circuit to an unstable state, the value of y will change (while x remains the same) until it reaches a stable (circled) state. Using this type of analysis for the remaining squares of the transition table, we find that the circuit repeats the sequence of states 00, 01, 11, 10 when the input repeatedly alternates between 0 and 1.

In a synchronous system, the present state is totally specified by the flip-flop values and does not change if the input changes while the clock pulse is inactive. In an asynchronous circuit, the internal state can change immediately after a change in the input. Because of this, it is sometimes convenient to combine the internal state with the input value together and call it the *total state* of the circuit. The circuit whose transition table is shown in Table.7-9 has four stable total states, $y_1 y_2 x = 000, 011, 110,$ and 101, and four unstable total states, 001, 010, 111, and 100.

We can obtain the state table with the transition table of asynchronous sequential circuits. We regard the secondary variables as the present state and the excitation variables as the next state. Table 7-10 provides the same information as the transition table. There is one restriction that applies to the asynchronous case but does not apply to the synchronous case. In the asynchronous transition table, there usually is at least one next state entry that is the same as the present-state value in each row. Otherwise, all the total states in that row will be unstable.

Table 7-10. State Table

Present State		***Next State***			
		***X* = 0**		***X* = 1**	
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	1	1	1	0

The procedure for obtaining a transition table is given below:

1. Determine all feedback loops in the circuit.
2. Designate the output of each feedback loop with variable Y_i and its corresponding input with Y_i for $i = 1, 2, \ldots\ldots, k$, where k is the number of feedback loops in the circuit.
3. Drive the Boolean functions of all Y's as a function of the external inputs and the y's.
4. Plot each Y function in a map, using the y variables for the rows and the external inputs for the columns.
5. Combine all the maps into one table showing the value of $Y = Y_1\ Y_2 \ldots . Y_k$ inside each square.
6. Circle those values of Y in each square that are equal to the value of $y = y_1 y_2 \ldots y_k$ in the same row.

Once the transition table is available, the behavior of the circuit can be analyzed by observing the state transition as a function of changes in the input variables.

7-57. Flow Table

In asynchronous sequential circuits, it is more convenient to name the states by letter symbols without making specific reference to their binary values. Such a table is called a flow table. A flow table is similar to a transition table except that the internal states are symbolized with letters rather than binary numbers. The flow table also includes the output values of the circuit for each stable state.

Flow table is shown in Fig. 7-91. The four states represent by the letters *a*, *b*, *c*, and *d*. it reduces to the transition table of Fig. 7-90 if we assign the following binary values to the states: *a* = 00, *b* = 01, *c* = 11, and *d* = 10. The table of Fig. 7-91 is called a ***primitive flow table*** because it has only stable state in each row.

y \ x		
a	ⓐ	b
b	c	ⓑ
c	ⓒ	d
d	a	ⓓ

Fig. 7-91. Four state with one input

Fig. 7-92 shows a flow table with more than one stable state in the same row. It has two states, *a* and *b*; two inputs, x_1 and x_2; and one output, *z*. The binary value of the output variable is indicated inside the square next to the state symbol and is separated by a comma. From the flow table, we observe the following behavior of the circuit. If $x_1 = 0$, the circuit is in state *a*. If x_1 goes to 1 while x_2, is 0, the circuit goes to state *b*. With inputs x_1x_2, = 11, the circuit may be either in state *a* or state *b*. If in state *a*, the output is 0, and if in state *b*, the output 1. State *b* is maintained if the inputs change from 10 to 11. The circuit stays in state *a* if the inputs change from 01 to 11. Remember that in fundamental mode, two input variables cannot change simultaneously, and therefore we do not allow a change of inputs from 00 to 11.

x_1x_2	00	01	11	10
a	ⓐ 0	ⓐ 0	ⓐ 0	b, 0
b	a, 0	a, 0	ⓑ 0	ⓑ 0

Fig. 7-92. Two state with two inputs and one output

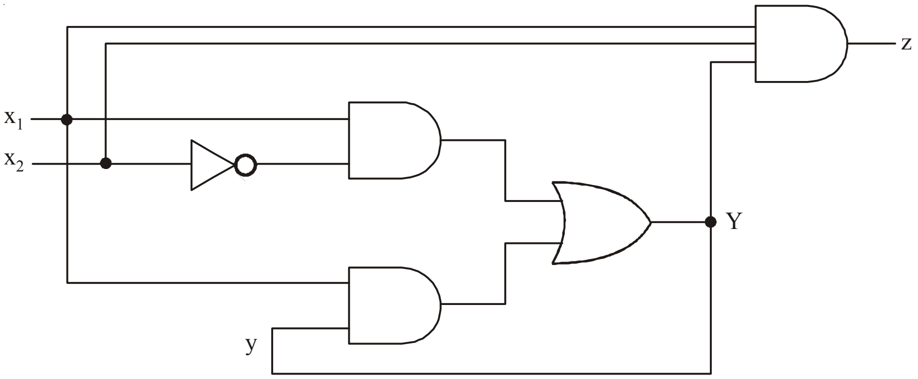
In order to obtain the circuit described by a flow table, it is necessary to assign to each state a distinct binary value. This assignment converts the flow table into a transition table from which we can derive the logic diagram. This is illustrated in Fig. 7-93 for the flow table of Fig. 7-92. We assign binary 0 to state *a* and binary 1 to state *b*. The result is the transition table of Fig. 7-92(a). The output map shown in Fig. 7-93 (b) is obtained directly from the output values in the flow table. The excitation function *Y* and the output function *z* are simplified by means of the two maps. The logic diagram of the circuit is shown in Fig. 7-93 (c).

$y \backslash X_1X_2$	00	01	11	10
0	⓪	⓪	⓪	1
1	0	0	①	①

(a) Transition Table $Y = x_1x'_2 + x_1y$

$y \backslash x_1x_2$	00	01	11	10
0	0	0	0	0
1	0	0	1	0

(b) Output $z = x_1x_2y$



(c) Logic Diagram

Fig. 7-93.

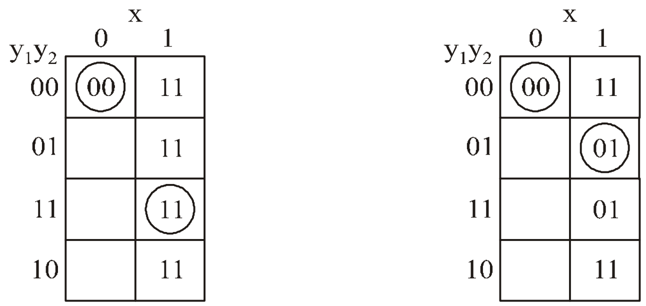
This example demonstrates the procedure for obtaining the logic diagram from a given flow table. This procedure is not always as simple as in this example. There are several difficulties associated with the binary state assignment and with the output assigned to the unstable states. These problem are discussed in detail in the following sections.

7-58. Race Conditions

A race condition is said to exist in an asynchronous sequential circuit when two or more binary state variables change value in response to a change in an input variable. When unequal delays are encountered, a race condition may cause the state variables to change in an unpredictable manner. For example, if the state variables must change from 00 to 11, the difference in delays may cause the first variable to change sooner than the second, with the result that the state variables change in sequence from 00 to 10 and then to 11. If the second variable changes faster than the first, the state variables will change from 00 to 01 and then to 11. Thus, the order by which the state variables change may not be known in advance. If the final stable state that the circuit reaches does not depend on the order in which the state variables change, the race is called a ***noncritical*** **race.** If it is possible to end up in two or more different stable states, depending on the order in which the state variables change, then it is a **critical race.** For proper operation, critical races must be avoided. We shall now discuss the races one by one.

1. Noncritical Race

The two examples in Fig. 7-94 illustrate noncritical races. We start with the total stable state y_1y_2x = 000 and then change the input from 0 to 1. The state variables must change from 00 to 11, which define a race condition. The listed transitions under each table show three possible ways that the state variables may change. They can either change simultaneously from 00 to 11, or they may change in sequence from 00 to 10 and then to 11, or they may change in sequence from 00 to 10 and then to 11. In all cases, the final stable state is the same, which results in a noncritical race condition. In (a), the final total state is y_1y_1x = 111, and in (b), it is 011.



y_1y_2 \ x	0	1
00	00	11
01		11
11		11
10		11

y_1y_2 \ x	0	1
00	00	11
01		01
11		01
10		11

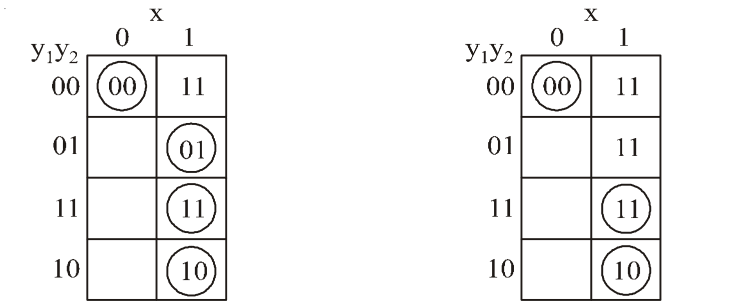
(a) Possible transitions:
00 → 11
00 → 01 → 11
00 → 10 → 11

(b) Possible transitions:
00 → 11 → 01
00 → 01
00 → 10 → 11 → 01

Fig. 7-94.

2. Critical Race

The transition tables of Fig. 7-95 (a) illustrate critical races. Here again we start with the total stable state y_1y_1x = 000 and then change the input from 0 to 1. The state variables must change from 00 to 11. If they change simultaneously, the final total stable is 111. In the transition table of part (a), if y_2 changes to 1 before y_1 because of unequal propagation delay, then the circuit goes to the total stable state 011 and remains there. On the other hand, if y_1 changes first, the internal state becomes 10 and the circuit will remain in the stable total state 101. Hence, the race is critical because the circuit goes to different stable states depending on the order in which the state variables change. The transition table of Fig. 7.95 (b) illustrates another critical race, where two possible transitions result in one final total state, but the third possible transition goes to a different total state.



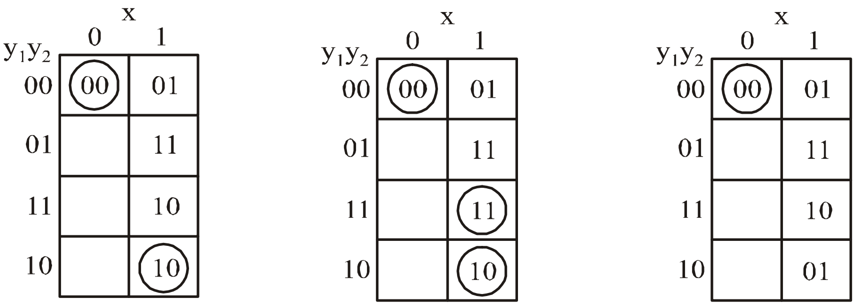
(a) Possible transitions:
00 → 11
00 → 01
00 → 10

(b) Possible transitions:
00 → 11
00 → 01 → 11
00 → 10

Fig. 7-95.

Races may be avoided by making a proper binary assignment to the state variables. The state variables must be assigned binary numbers in such a way that only one state variable can change at any one time when a state transition occurs in the flow table.

Races can be avoided by directing the circuit through intermediate unstable states with a unique state-variable change. When a circuit goes through a unique sequence of unstable states, it is said to have a *cycle*. Fig. 7-95 illustrates the occurrence of cycles. Again we start with y_1y_2 = 00 and then change the input from 0 to 1. The transition table of part (a) gives a *unique* sequence that terminates in a total stable state 101. The table in (b) shows that even though the state variables change from 00 to 11, the cycle provides a unique transition from 00 to 01 and then to 11. Care must be taken when using a cycle that it terminates with a stable state. If a cycle does not terminate with a stable state, the circuit will keep going from one unstable state to another, making the entire circuit unstable. This is demonstrated in Fig. 7-96 and also in the following example.



(a) State transitions
00 → 01 → 11 → 10

(b) State transitions
00 → 01 → 11

(b) Unstable
→ 01 → 11 → 10

Fig. 7-96.

7-59. Circuits with Latches

Asynchronous sequential circuits can be implemented by employing a basic flip-flop commonly referred to as an *SR latch*. The use of *SR* latches in asynchronous circuits produces a more orderly pattern, which may result in a reduction of the circuit complexity. An added advantage is that the circuit resembles the synchronous circuit in having distinct memory elements that store and specify the internal states. We discuss the following type of circuit with latches.

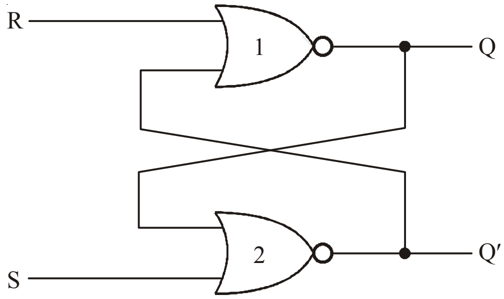
1. Circuit with SR NOR Latch
2. Circuit with SR NAND Latch

Now we shall discuss all the latch circuit in the following pages.

7-60. Circuit with S-R NOR Latch

The *SR* latch is a digital circuit with two inputs, *S* and *R*, and two cross-coupled NOR gates or two cross-coupled NAND gates. The cross-coupled NOR circuit is shown in Fig. 7-97 (a) and the truth table is shown in Fig. 7-97 (b). In order to analyze the circuit by the transition-table method, we redraw the circuit, as shown in Fig. 7-97 (c). Here we distinctly see a feedback path from the output of gate 1 to the input of gate 2. The output *Q* is identical to the excitation variable *Y* and the secondary variable *y*. The Boolean function for the output is

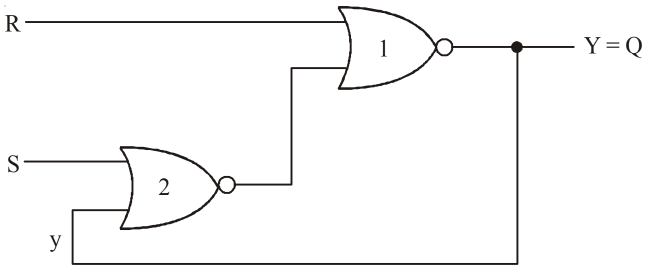
$$Y = \overline{[(\overline{S+y})+R]} = S\overline{R} + \overline{R}y$$



(a) Cross- Coupled NOR Circuit

S	*R*	*Q*	$\overline{Q}$	
1	0	1	0	
0	0	1	0	(After *SR* = 10)
0	1	0	1	
0	0	0	1	(After *SR* = 01)
1	1	0	0	

(b) Truth Table



(c) Circuit

Fig. 7-97.

We obtain the transition table for the circuit from *Y*. We can now investigate the behavior of the *S*-*R* latch from the transition table. With *SR* = 10, the output *Q* = *Y* = 1 and the latch is said to be set. Changing *S* to 0 leaves the circuit in the set state. With *SR* = 01, the output *Q* = *Y* = 0 and the latch is said to be reset. A change of *R* back to 0 leaves the circuit in the reset state. These conditions are also listed in the truth table. The circuit exhibits some difficulty when both *S* and *R* are equal to1. From the truth table, we see that both *Q* and $\overline{Q}$ are equal to 0, a condition that violates the requirement that these two outputs be the complement of each other. Moreover, from the transition table, we note that going from *S*-*R* = 11 to *S*-*R* = 00 produces an unpredictable result. If *S* goes to 0 first, the output remains at 0, but if *R* goes to 0 first, the output goes to 1. In normal operation, we must make sure that 1's are not applied to both the Sand *R* inputs simultaneously. This condition can be expressed by the Boolean function *SR* = 0, which states that the ANDing of Sand *R* must always result in a 0.

Coming back to the excitation function, we note that when we OR the Boolean expression *SR'* with *SR,* the result is the single variable *S.*

$$S\overline{R} + SR \;=\; S(\overline{R} + R) = S$$

From this we deduce that $S\overline{R} = S$ when *SR* = 0. Therefore, the excitation function,

$$Y \;=\; S\overline{R} + \overline{R}y \;=\; S + \overline{R}y \qquad \text{When } SR = 0$$

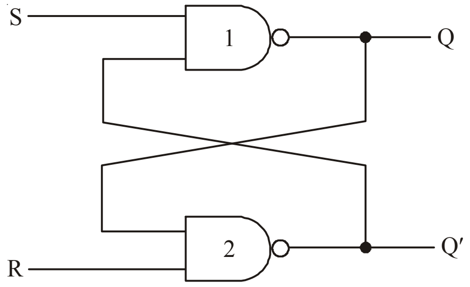
To analyze a circuit with an *SR* latch, we must first check that the Boolean condition *SR* = 0 holds at all times. We then use the reduced excitation function to analyze the circuit. However, if it is found that both *S* and *R* can be equal to 1 at the same time, then it is necessary to use the original excitation function.

7-61. Circuit with S-R NAND Latch

The analysis of the *S*-*R* latch with NAND gates is carried out in Fig. 7-98. The NAND latch operates with both inputs normally at 1 unless the state of the latch has to be charged. The application of 0 to *R* causes the output *Q* to go to 0, thus putting the latch in the reset state. After the *R* input returns to 1, a change of *S* to 0 causes a change to the set state. The condition to be avoided here is that both *S* and *R* not be 0 simultaneously. This condition is satisfied when $\overline{S}\,\overline{R} = 0$. The excitation function for the circuit is

$$Y \;=\; [\overline{S(\overline{Ry})}] = \overline{S} + Ry$$

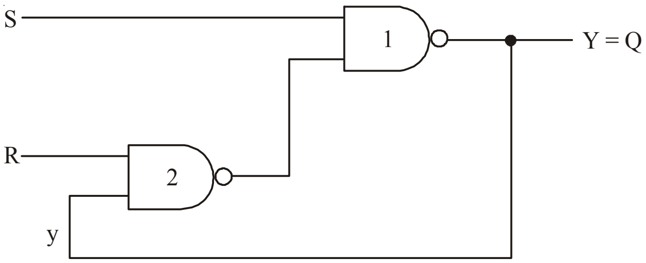
Comparing it with the excitation function of the NOR latch, we note that *S* has been replaced with $\overline{S}$ and $\overline{R}$ with *R*. Hence, the input variables for the NAND latch require the complemented values of those used in the NOR latch. For this reason, the NAND latch is sometimes referred to as an $\overline{S}\,\overline{R}$ latch (or $\overline{S} - \overline{R}$ latch).



(a) Crossed-coupled circuit

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(After $SR = 10$)
0	1	1	0	
1	1	1	0	(After $SR = 01$)
0	0	1	1	

(b) Truth Table



(c) Circuit showing feedback

y \ SR	00	01	11	10
0	1	1	0	0
1	1	1	1	0

$Y = S' + Ry$ with $S'R' = 0$

(d) Transition table

Fig. 7-98.

7-62. Analysis of Asynchronous Sequential

Asynchronous sequential circuits can be constructed with the use of SR latches with or without external feedback paths. There is always a feedback loop within the latch itself. The analysis of a circuit with latches will be demonstrated by means of a specific example.

The circuit shown in Fig. 7-99 has two *SR* latches with outputs Y_1 and Y_2. There are two inputs, x_1 and x_2 and two external feedback loops giving rise to the secondary variables y_1 and y_2. The analysis of the circuit requires that we first obtain the Boolean functions for the *S* and *R* inputs in each latch.

$$\begin{aligned} S_1 &= x_1 y_2 \\ S_2 &= x_1 x_2 \\ R_1 &= \bar{x}_1 \bar{x}_2 \\ R_2 &= \bar{x}_2 y_1 \end{aligned}$$

We then check whether the condition $SR = 0$ is satisfied to ensure proper operation.

$$\begin{aligned} S_1 R_1 &= x_1 y_2 \bar{x}_1 \bar{x}_2 = 0 \\ S_2 R_2 &= x_1 x_2 \bar{x}_2 y_1 = 0 \end{aligned}$$

The result is 0 because $x_1 \bar{x}_1 = x_2 \bar{x}_2 = 0$

The next step is to derive the transition state of the circuit. Remember that the transition table specifies the value of Y as a function of y and x. The excitation functions are derived from the relation $Y = S + \overline{R}\,y$.

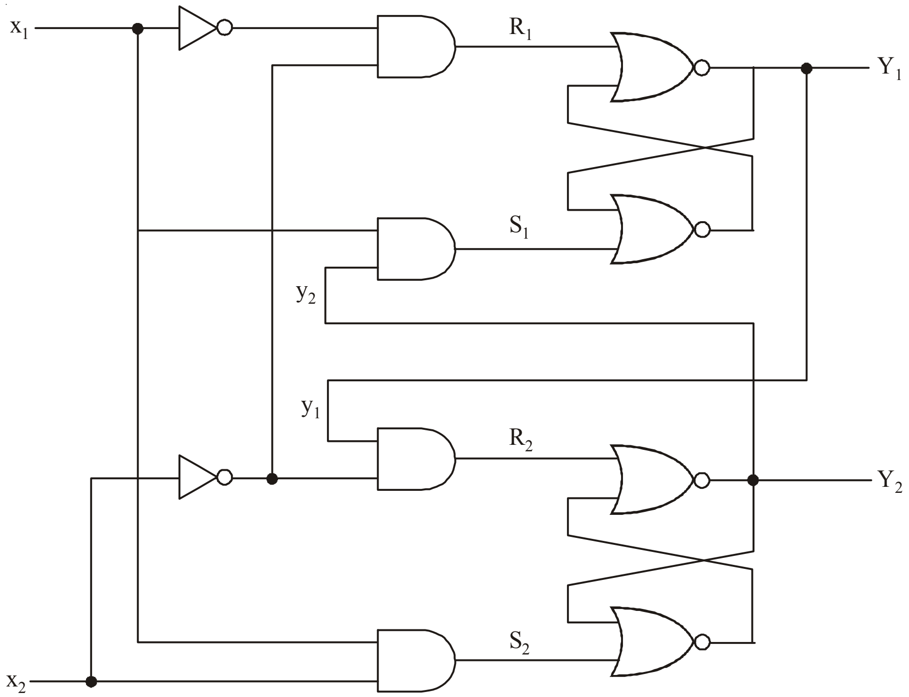


Fig. 7-99. Circuit with SR Latches

	x_1x_2			
y_1y_2	00	01	11	10
00	00	00	01	00
01	01	01	11	11
11	00	11	11	10
10	00	10	11	10

Fig. 7-100. Transition Table

$$Y_1 = S_1 + \overline{R}_1\,y_1 = x_1y_2 + (x_1 + x_2)\,y_1 = x_1y_2 + x_1y_1 + x_2y_1$$

$$Y_2 = S_2 + \overline{R}_2\,y_2 = x_1x_2 + (x_1 + \overline{y}_1)\,y_2 = x_1x_2 + x_1y_2 + \overline{y}_1y_2$$

Fig. 7-100 shows the composite map for $Y = Y_1Y_2$. The y variables are assigned to the rows in the map, and the x variables are assigned to the columns. The entries of Y in each row that have the same value as that given to Y are circled and represent stable states. The procedure for analyzing an asynchronous sequential circuit with SR latches is as follows:

1. Label each latch output with Y, and its external feedback path (if any) with y_1 for $i = 1, 2, \ldots. k$.
2. Derive the Boolean functions for the S_i and R_i inputs in each latch.

3. Check whether $SR = 0$ for each NOR latch or whether $\overline{S}\,\overline{R} = 0$ for each NAND latch. If this condition is not satisfied, there is a possibility that the circuit may not operate properly.
4. Evaluate $Y = S + \overline{R}\,y$ for each NOR latch or $Y = \overline{S} + R\,y$ for each NAND latch.
5. Construct a map with the y's representing the rows and the x inputs representing the columns.
6. Plot the value of $Y = Y_1 Y_2 \ldots Y_k$ in the map.
7. Circle all stable states where $Y = y$. The resulting map is then the transition table.

7-63. Design Procedure

The design of an asynchronous sequential circuit starts from the statement of the problem and logic diagram. There are a number of design steps that must be use to minimize the circuit complexity and to produce a stable circuit. The design step of asynchronous circuit is as follow:

1. First we obtain the primitive flow table from the design.
2. The flow table is reduced to a minimum number of states. This is done by merging rows in the primitive flow table.
3. The states are then given a binary assignment from which we obtain the transition table.
4. From the transition table, we drive the logic diagram as a combinational circuit with feedback or as a circuit with *SR* latches.

Let's take an example to understand the design process. It is necessary to design a gated latch circuit with two inputs, *G* (gate) and *D* (data), and one output, *Q*. binary information present at the *D* input is transferred to the *Q* output when *G* is equal to 1. The *Q* output will follow the *D* input as long as $G = 1$. When *G* goes to 0, the information that was present at the *D* input at the time the transition occurred is retained at the *Q* output. The gated latch is a memory element that accepts the value of *D* when $G = 1$ and retains this value after *G* goes to 0. Once $G = 0$, a change in *D* does not change the value of the output *Q*.

7-64. Primitive Flow Table

Primitive flow table is a flow table with only one stable total state in each row. The total state consists of the internal state combined with the input. The primitive flow table can be construct form a table with all possible total states in the system. This is shown in Table 7-11 for the gated latch. Each row in the table specifies a total state, which consists of a letter designation for the internal state and a possible input combination for *D* and *G*. The output *Q* is also shown for each total state. We start with the two total states that have $G = 1$. From the design specifications, we know that $Q = 0$ if $DG = 01$ and $Q = 1$ if $DG = 11$ because *D* must be equal to *Q* when $G = 1$. We assign these conditions to states *a* and *b*. When *G* goes to 0, the output depends on the last value of *D*. Thus, if the transition of *DG* is from 01 to 00 to 10, then *Q* must remain 0 because *D* is 0 at the time of the transition from 1 to 0 in *G*. If the transition of *DG* is from 11 to 10 to 00, then *Q* must remain 1.

Table 7-11.

State	***Inputs***		***Output***	***Comments***
	D	***G***	***Q***	
a	0	1	0	$D =$ because $G = 1$
b	1	1	1	$D =$ because $G = 1$
c	0	0	0	After state *a* or *d*
d	1	0	0	After state *c*
e	1	0	1	After state *b* or *f*
f	0	0	1	After state *e*

The primitive flow table for the gated latch is shown in Fig. 7-101. It has one row for each state and one column for each input combination.

We fill in one square in each row belonging to the stable state in that row. These entries are determined from Table 7-11. For example, state *a* is stable and the output is 0 when the input is 01. This information is entered in the flow table in the first row and second column. Similarly, the other five stable states together with their output are entered in the corresponding input columns. Both inputs are not allowed to change simultaneously, we can enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state.

For example, the first row in the flow table shows a stable state with an input of 01. Since only one input can change at any given time, it can change to 00 or 11, but not to 10. Therefore, we enter two dashes in the 10 column of row *a*. This will eventually result in a don't-care condition for the next state and output in this square. Following this procedure, we fill in a second square in each row of the primitive flow table.

	DG			
	00	01	11	10
a	*c*, –	ⓒ, 0	*b*, –	–, –
b	–, –	*a*, –	ⓑ, 1	*e*, –
c	ⓒ, 0	*a*, –	–, –	*d*, –
d	*c*, –	–, –	*b*, –	ⓓ, 0
e	*f*, –	–, –	*b*, –	ⓔ, 1
f	ⓕ, 1	*a*, –	–, –	*e*, –

Fig. 7-101. Primitive Flow Table

Next it is necessary to find values for two more squares in each row. The comments listed in Table 7-11 may help in deriving the necessary information. For example, state *c* is associated with input 00 and is reached after an input change from state *a* or *b*. Therefore, an unstable state *c* is shown in column 00 and rows *a* and *d* in the flow table. The output is marked with a dash to indicate a don't-care condition. The interpretation of this is that if the circuit is in stable state *a* and the input changes from 01 to 00, the circuit first goes to an unstable next state *c*, which changes the present state value from *a* to *c*, causing a transition to the third row and first column of the flow table. The unstable state values for the other squares are determined in a similar manner. All outputs associated with unstable states are marked with a dash to indicate don't-care conditions. The assignment of actual values to the outputs is discussed further after the design example is completed.

7-65. Reduction of the Primitive Flow Table

The primitive flow table has only one stable state in each row. The table can be reduced to a smaller number of rows if two or more states are placed in the same row of the flow table. The grouping of stable states from separate rows into one common row is called *merging*. Merging a number of stable states in the same row means that the binary state variable that is ultimately assigned to the merged row will not change when the input variable changes. This is because in a primitive flow table, the state variable changes every time the input changes, but in a reduced flow table, a change of input will not cause a change in the state variable if the next stable state is in the same row.

The primitive flow table is separated into two parts of three rows each, as shown in Fig. 7-102. Each part shows three stable states that can be merged because there are no conflicting entries in each of the four columns. The first column shows state *c* in all the rows and 0 or *a* dash for the output. Since *a* dash represents a don't-care condition, it can be associated with any state or output. The two dashes in the first column can be taken as 0 output to make all three rows identical to a stable state *c* with a 0 output. The second column shows that the dashes can be assigned to correspond to a stable state *a* with a 0 output. Note that if the state is circled in one of the rows, it is also circled in the merged row. Similarly, the third column can be merged into an unstable state *b* with a don't care output and the fourth column can be merged into stable state *d* and *a* 0 output. Thus, the three rows, *a*, *c*, and *d*, can be merged into one row with three stable states and one unstable state, as shown in the first row of Fig. 7-102. The second row of the reduced table results from the merging of rows *b*, *e*, and *f* of the primitive flow table.

	DG			
	00	01	11	10
a	*c*, –	ⓐ, 0	*b*, –	–, –
c	ⓒ, 0	*a*, –	–, –	*d*, –
d	*c*, –	–, –	*b*, –	ⓓ, 0

	DG			
	00	01	11	10
b	–, –	*a*, –	ⓑ, 1	*e*, –
e	*f*, –	–, –	*b*, –	ⓔ, 1
f	ⓕ, 1	*a*, –	–, –	*e*, –

Fig. 7-102. State for Merging

There are two ways that the reduced table can be drawn. The letter symbols for the states can be retained to show the relationship between the reduced and primitive flow tables. The other alternative is to define a common letter symbol for all the stable states of the merged rows. Thus, states *c* and *d* are replaced by state *a*, and states *e* and *f* are replaced by state *b*. Both alternatives are shown in Fig. 7-103.

	DG			
	00	01	11	10
a, c, d	ⓒ, 0	ⓐ, 0	*b*, –	ⓓ, 0
b, e, f	ⓕ, 1	*a*, –	ⓑ, 1	ⓔ, 1

(a)

	DG			
	00	01	11	10
a	ⓐ, 0	ⓐ, 0	*b*, –	ⓐ, 0
b	ⓑ, 1	*a*, –	ⓑ, 1	ⓑ, 1

(b)

Fig. 7-103. Reducing Table

7-66. Transition Table and Logic Diagram

In order to obtain the circuit described by the reduced flow table, it is necessary to assign to each state a distinct binary value. This assignment converts the flow table into a transition table.

Assigning 0 to state *a* and 1 to state *b* in the reduced flow table. We obtain the transition table of Fig. 7-104. The transition table is, in effect, a map for the excitation variable *Y*. The simplified Boolean function for *Y* is then obtained from the map.

$$Y = DG + G'y$$

y \ *DG*	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Fig. 7-104.

There are two don't care outputs in the final reduced flow table. If we assign values to the output, as shown in Fig. 7-105 it is possible to make output *Q* equal to the excitation function *Y*. if we assign the other possible values to the don't-care outputs, we can make output *Q* equal to *y*.

y \ *DG*	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Fig. 7-105.

The logic diagram of the gated latch is as shown in Fig. 7-106.

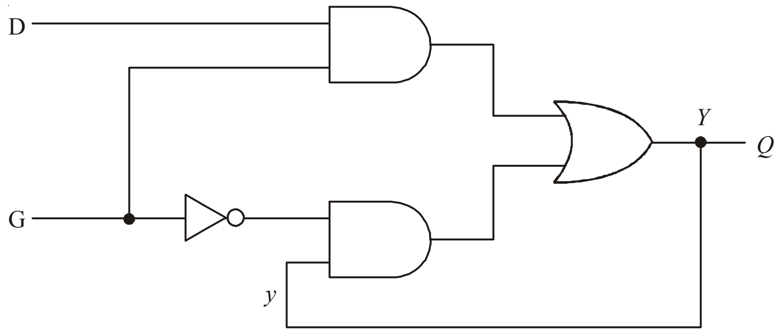


Fig. 7-106. Logic Diagram

7-67. Hazard

The unwanted switching transients (glitches) that may appear at the output of a circuit are called Hazards. The hazards cause the circuit to malfunction. The main cause of hazards is the different propagation delays at different paths. Hazards occur in the combinational circuits, where they may cause a temporary false output value. When such combinational circuits are used in the asynchronous sequential circuits, they may result in a transition to a wrong stable state. There are four types of hazard which can occur in digital systems they are given below:

1. Static Hazards

Static hazards are due to a momentary change in output caused by an input change that does not affect the steady-state output. They may be present in both combinational circuits and gate-implemented asynchronous circuits. Static hazards are two types they are given below:

(*i*) **Static 1- hazard:** When the circuit outputs momentarily go to 0 when it should remain a constant 1, we say the circuit has a static 1-hazard. Static 1-hazard is shown in Fig. 7-107. This hazard possible in AND-OR circuits.

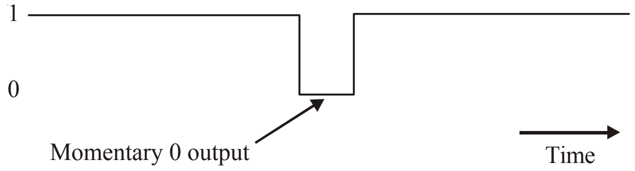


Fig. 7-107. Static 1- hazard

(*ii*) **Static 0-hazard:** When the circuit outputs momentarily go to 1 when it should remain a constant 0, we say the circuit has a static 0-hazard. Static 0-hazard is shown in Fig. 7-108. This hazard possible in OR-AND circuits.

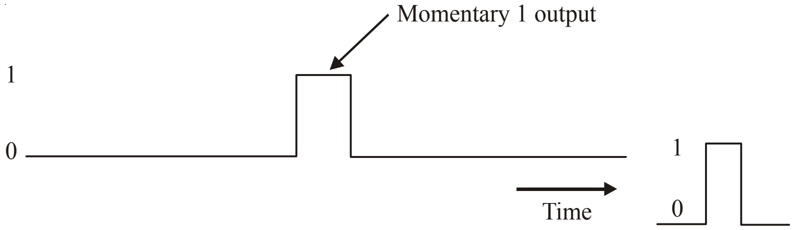


Fig. 7-108. Static 0-hazard

2. Dynamic Hazards

There is another type of hazard associated with combinational networks, called a dynamic hazard that is responsible for false outputs it is shown in Fig. 7-109. These hazards occur when the output of a network is to change between its two logic states, but a momentary false output signal occurs during the transient behavior. This hazard is define as a transient change occurring three or more times at an output terminal of a logic network when the output is supposed to change only once during a transition between two input states differing in the value of one variable. When output is supposed to change from 1 to 0 (or 0 to 1) the output may change three or more times, we say circuit has dynamic hazard.

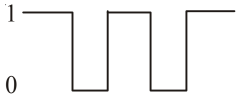


Fig. 7-109. Dynamic hazard

Let consider the circuit shown in Fig. 7-109. This circuit is free from static 0-hazards and static 1-hazards. Now consider the input states $x_1x_2x_3$ = 000 and $x_1x_2x_3$ = 100. For the first input state, the steady state output is 0; while for the second input state, the steady-state output is 1. Assume there are no propagation delays through gates G3 and G5 and that the propagation delays of the other three gates are such that G1 can switch faster than G2 and that G2 can switch faster than G4. When x_1

changes from 0 to 1, the change propagates through gate G1 before gate G2 with the net effect that the inputs to gate G3 are simultaneously 1 and, correspondingly, the network output changes from 0 to 1. Then, when the x_1 change propagates through gate G2, the lower input to gate G3 becomes 0 and the network output changes back to 0. Finally, when the x_1 = 1 signal propagates through gate G4, the lower input to gate G5 becomes 1 and the network output again changes to 1. It is therefore seen that during the change of the x_1 variable from 0 to 1. In this case, the output undergoes the sequence 0-1-0-1, which results in three changes when it should have undergone only a single change. Dynamic hazards are more difficult to detect than static hazards. They are capable of producing false outputs.

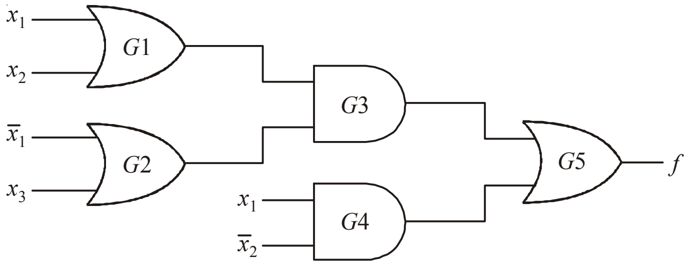


Fig. 7-110.

3. Function Hazards

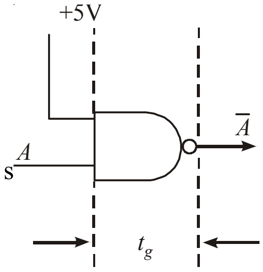
Function hazards are non-solvable hazards which occurs when more than one input variable changes at the same time. Function hazards occur when more than one input variable change takes place at same time.

4. Essential Hazards

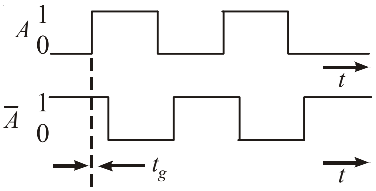
The essential hazard is a type of hazard that exists only in asynchronous sequential circuits with two or more feedback. An essential hazard is caused by unequal delays along two or more paths that originate from the same input. Same hazards can be eliminated by adjusting the amount of delays in the affected path. Such hazards can be eliminated by adjusting the amount of delays in the affected path.

7-68. Gate Delays

If a two input NAND gate is used as in inverter in a combinational network, as shown in Fig. 7-111(a) there will be a finite time delay t_g before any change at the input to the gate produces the required change at the output. This delay is demonstrated in the timing diagrams, where the change in A from 0 to 1 is followed by a change in $\overline{A}$ from 1 to 0, t_g seconds later. Similarly, when A changes from 1 to 0, the corresponding change in $\overline{A}$ from 0 to 1 also occurs later as shown in Fig. 7-111 (b).



(a)



(b) Timing Diagram

Fig. 7-111.

7-69. Generation of Spikes

If the signal A and its complement $\bar{A}$, generated by the NAND gate as shown in Fig. 7-112. Both A and $\bar{A}$ are fed to the inputs of a two input AND gate then according to the laws of Boolean algebra the output of the gate should be $A\bar{A} = 0$ at all times.

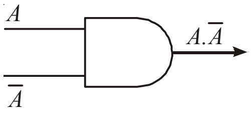


Fig. 7-112.

However, it will be observed from the timing diagram as shown in Fig. 7-112 that in the time periods t_g, A and $\bar{A}$ are simultaneously equal to 1, so that during these periods the gate output is $A\bar{A}$ = 1. The output of the gate $A\bar{A}$ consists of a series of positive going spikes which are imitated when A is changing from 0 to 1, each of time duration t_g, the gate delay of the inverter. The circuit used to generate the signal $A\bar{A}$ is said to exhibit a static 0-hazard because the output signal, which should be presently 0, goes to 1 for a short transient period.

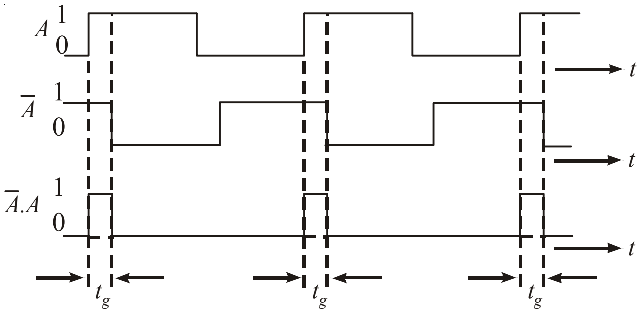


Fig. 7-113. Timing Diagram

Alternatively, if the signals A and $\bar{A}$ are applied to the inputs of a two-input OR gate as shown in Fig. 7-114 then the output of the gate is $A + \bar{A}$, which according to the laws of Boolean algebra, should be 1 at all instants of time.

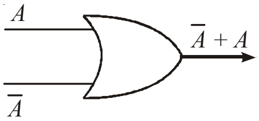


Fig. 7-114.

The waveforms of A and $\overline{A}$ shown in Fig. 7-115 that during the shaded time periods, they are both simultaneously equal to 0. In these shaded time periods, which are of short time duration, the output goes to 0. The circuit is said to exhibit a static 1-hazard because its output, which is normally 1, goes to 0 for short time periods. It will be observed that for the OG gate, the negative going spikes are initiated at the instant when A is changing from 1 to 0.

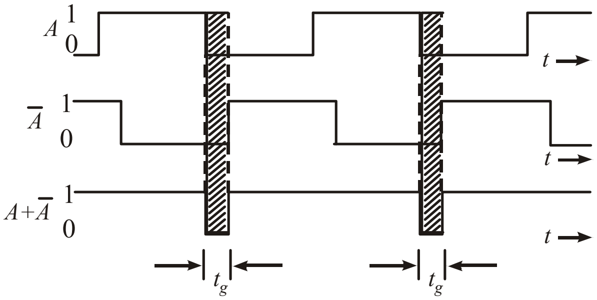


Fig. 7-115. Timing Diagram

The generation of spikes by NAND and NOR gates shown in Fig. 7-116. Negative going spikes are generated by a NAND gate at the instant when A is changing from 0 to 1. The circuit exhibits a static 1-hazard. In the NOR circuits, positive going spikes are generated at the instant when A is changing from 1 to 0. This circuit exhibits a static 0-hazard.

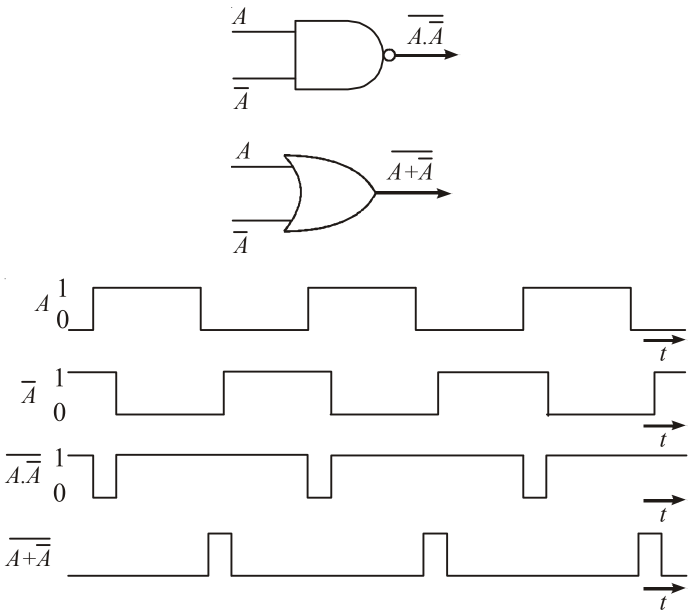

A
$\overline{A}$
$\overline{A.\overline{A}}$
A
$\overline{A}$
$\overline{A+\overline{A}}$
A
1
0
t
$\overline{A}$
1
0
t
$\overline{A.\overline{A}}$
1
0
t
$\overline{A+\overline{A}}$
t

Fig. 7-116.

7-70. Hazard in Combinational Circuits

A hazard is a condition where a single variable change produces a momentary output change when no output change should occur. To illustrate a ***static 1- hazard***, consider the gate network of Fig. 7-117. Assume the input state $x_1x_2x_3$ = 111 is initially applied to the network. Then the upper AND gate A has a 1 output, and the lower AND gate B has a 0 output and final output f = 1. If the input state is next changed to $x_1x_2x_3$ = 101. The value of x_2 changes, then the output of the upper AND gate A becomes 0 and the lower AND gate B becomes 1 output and final output is same f = 1.

For both input states, the output of the network is 1. However, if there is an appreciable propagation delay associated with the NOT-gate, then the output of upper AND gate A becomes 0 before the output of the lower AND gate B becomes 1. This result in a period of time in which neither input terminal to the OR-gate is a 1 and, consequently, the output *f* is 0 for that period of time. Thus, a static 1-hazard exits during the transition between the state $x_1x_2x_3 = 111$ and $x_1x_2x_3 = 101$.

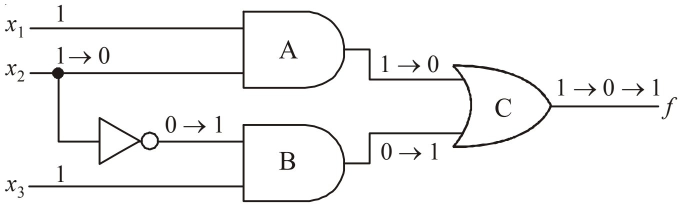


Fig. 7-117.

To understand a static 0-hazard, consider the network shown in Fig. 7-118 assume that the NOT gate has an appreciably greater propagation delay time than the other gates. In this case there is a ***static 0-hazard*** in the transition between the input states $x_1x_2x_3 = 000$ and $x_1x_2x_3 = 010$. Since it is possible for a logic-1 signal to appear at both input terminal of the AND gate for a short duration.

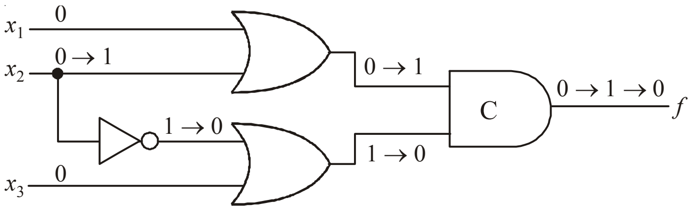


Fig. 7-118.

7-71. Detection of Static Hazards

The occurrence of the hazard can be detected by inspecting the map of the particular circuit. Consider the circuit shown in Fig. 7-119, the K-map of this circuit is shown in Fig. 7-120. The change in x_2 from 1 to 0 moves the circuit from minterm 111 to minterm 101. The hazard exists because the change of input results in a different product term covering the two minterms. Minterm 111 is covered by the product term implemented in AND gate 1, and interm 101 is covered by the product term implemented in AND gate 2. Whenever the circuit must move from one product term to another, there is a possibility of a momentary interval when neither term is equal to 1, giving rise to an undesirable 0 output. The output function of the circuit is,

$$y = x_1x_2 + \bar{x}_2x_3$$

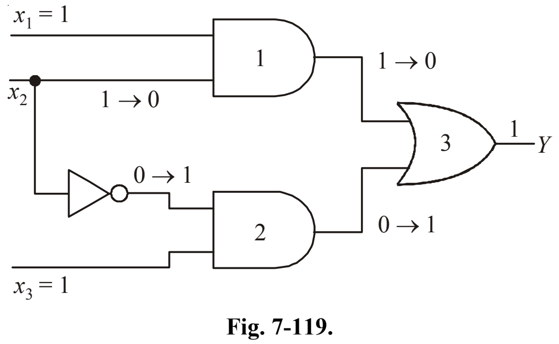
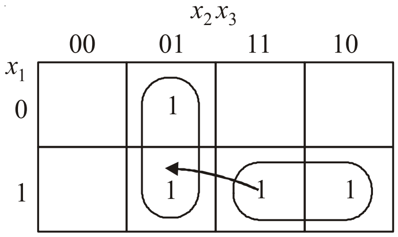


Fig. 7-119.

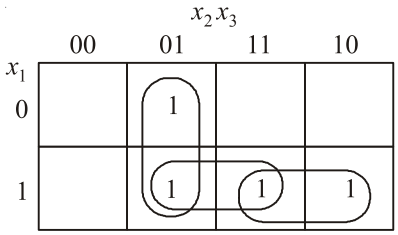


x_1 \ x_2x_3	00	01	11	10
0		1		
1		1	1	1

Fig. 7-120.

7-72. Elimination of Static Hazards

The above discussion on the detection of static hazards also suggests a method for their elimination. Additional logic is used to provide the holding action that is needed to eliminate the static hazards. The additional logic is determined by ensuring that any two adjacent cells on a Karnaugh map with the same logic-value entry are included in a common square. This is shown in the map of Fig. 7-121. Where the two minterms that cause the hazard are combined into one product term.



x_1 \ x_2x_3	00	01	11	10
0		1		
1		1	1	1

Fig. 7-121.

The hazard-free circuit obtained by this configuration is shown Fig. 7-122. The extra gate in the circuit generates the product term x_1, x_3. In general, hazards in combinational circuits can be removed by covering any two min terms that may produce a hazard with a product term common to both. The removal of hazards requires the addition of redundant gates to the circuit. The output of the circuit

$$y = x_1x_2 + \overline{x}_2x_3 + x_1x_3$$

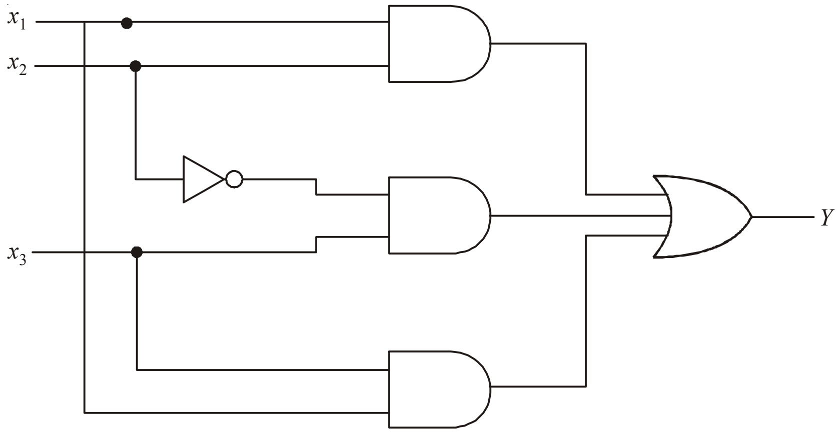


Fig. 7-122.

7-73. Hazards in Sequential Circuits

We known that, in sequential circuit, the combinational circuits are associated with them to drive flip-flop inputs. In synchronous sequential circuits, the hazards due to combinational circuits associated with then are not of concern. This is because momentary erroneous signals are not generally troublesome in synchronous circuits. However, if a momentary incorrect signal is feedback in an asynchronous sequential circuit, it may cause the circuit to go to the wrong stable state. Let us consider the logic diagram shown in Fig. 7-123. The output function of this circuit is

$$Y = x_1x_2 + \overline{x}_2 y$$

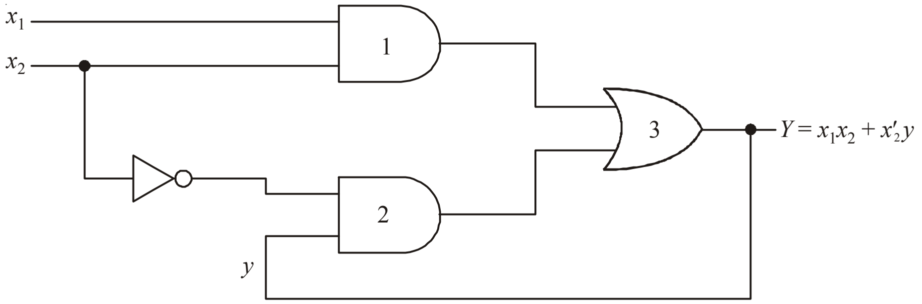
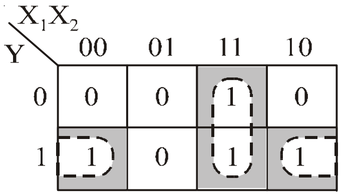


Fig. 7-123.

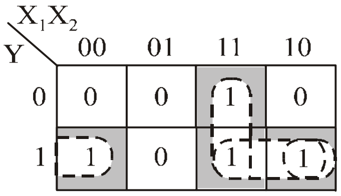
If the circuit is in total stable state $yx_1x_2 = 111$ and input x_2 changes from 1 to 0, the next total stable state should be 110. However, because of the hazard, output Y may go to 0 momentarily. If this false signal feeds back into gate 2 before the output of the inverter goes to 1, the output of gate 2 will remain at 0 and the circuit will switch to the incorrect total stable state 010. Such a hazard can be eliminated by enclosing two minterms by another minterm as shown in Fig. 7-124 and Fig. 7-125.



Y \ X1X2	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$Y = x_1x_2 + \overline{x}_2 y$

Fig. 7-124.



Y \ X1X2	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$Y = x_1x_2 + \overline{x}_2 y + x_1 y$

Fig. 7-125

Therefore, the hazard free asynchronous sequential circuit will be as shown in Fig. 7-126. The output function of this circuit is

$$Y = x_1x_2 + \overline{x}_2 y + x_1 y$$

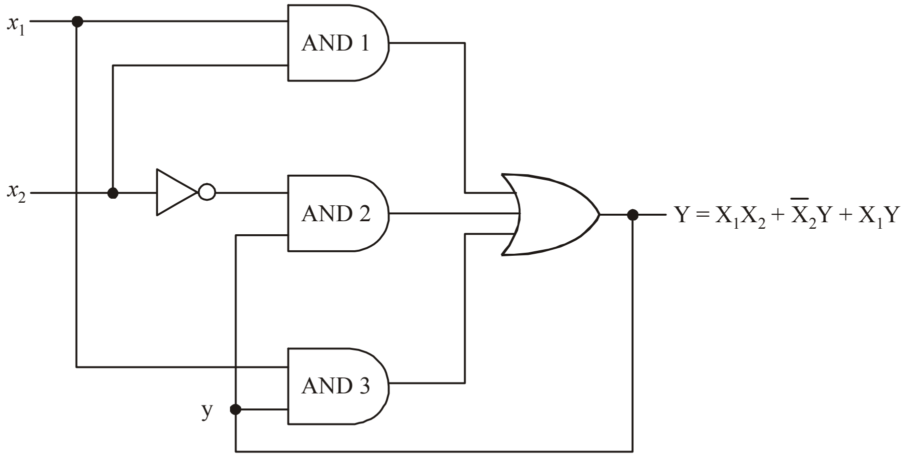


Fig. 7-126.

7-74. Faults in Combinational Logic Gates

Fig. 7-127 shown the basic two-input AND gate. Its inputs are *A* and *B* and output is *C*. The wires connecting input points *A* and *B* to input pins of the gate are designated as *a* and *b*, respectively, while the output wire is designated as *C*.

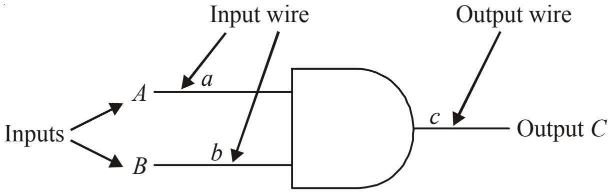


Fig. 7-127.

The truth table is shown in Fig. 7-128.

A	*B*	*C*
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 7-128.

Let us assume that due to some fault connection, line *b* is shorted to ground. This makes line *b* to remain permanently at logic 0 as shown in Fig. 7-129. We notice that, when *b* is permanently connected though some fault, the output so obtained is different from the normal output of the AND gate. This then can be used as a test for detecting a 0 fault in one of the input lines of an AND gate.

A	*B*	*C*
0	0	0
0	0	0
1	0	0
1	0	0

Fig. 7-129.

There are two types of faults they are given below:

1. Stuck-at-Zero (S-A-0)

Where an input line is grounded by a faulty connection so that the line is permanently at logic 0 is called a stuck-at-zero (A-A-0) fault. Fig. 7-130 shows the S-A-0 fault.

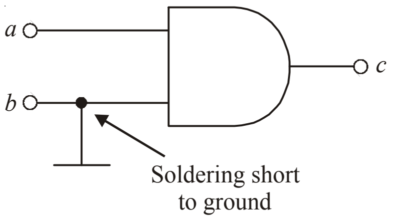


Fig. 7-130.

2. Stuck-at-Zero (S-A-1)

When an input line shorted to V_{CC} by fault, we say that line is Stuck-at-1 (S-A-1). Usually in digital circuits, majority of the faults that occur fall under these two categories. Fig. 7-131 shows the S-A-1 fault.

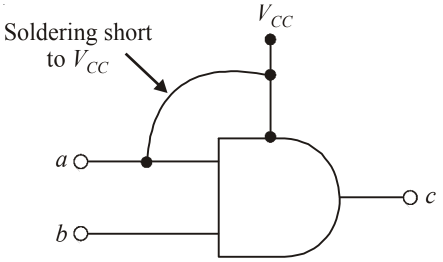


Fig. 7-131.

7-75. Fault-Detection Methods

The following two methods are generally used for detecting faults in combinational logic circuits:

1. Fault detection by fault tables
2. Fault detection by Path-sensitizing method

We will discuss each method one by one in the following pages.

7-76. Fault Detection by Fault Tables

A fault table is a truth table that shows all possible faults and the minimum set of tests to be performed on a combinational logic circuit to detect these faults. Consider the circuit shown in Fig. 7-132.

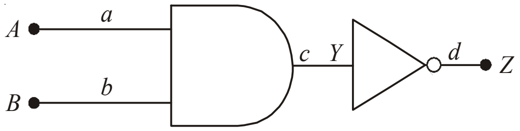


Fig. 7-132.

In the circuit shown, *A* and *B* are inputs to the AND gate, with *Y* its output. *Y* also forms the input to the NOT gate, and *Z* is the final output. As before *a*, *b*, *c* and *d* represent various connecting wires. The truth table for a fault-free output from this circuit is given in Table 7-12.

Table 7-12

A	B	$Y = AB$	$Z = \overline{Y}$
(line *a*)	(line *b*)	(line *c*)	(line *d*)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Let as now assume that line *a* is S-A-0. The new truth table with this fault is shown in Table 7-13.

Table 7-13. Line *a* as S-A-0

A	B	$Y = AB$	$Z = \overline{Y}$
(line *a*)	(line *b*)	(line *c*)	(line *d*)
0	0	0	1
0	1	0	1
0	0	0	1
0	1	0	1 ← Fault

Comparing table 7-11 and 7-12 we notice that they differ in the last roe only, as far as the output *Z* is concerned. The output *Z*- 0 for a fault-free circuit, and *Z* – 1 when line *a* is S-A-0. This is shown by a circle in table. Consider the second situation where *a* is now S-A-1. The truth table for this situation is shown in Table 7-14.

Table 7-14. Line *a* is S-A-0

$A(S\text{-}A\text{-}1)$	B	$Y = AB$	$Z = \overline{Y}$
1	0	0	1
1	1	1	0 ← Fault
1	0	0	1
1	1	1	0

In this case, we find that the fault occurs 1 row 2 where we get a 0 output, which actually should have been *a* 1 for fault-free condition. We can similarly draw truth tables for *b*, *c* and *d* lines, S-A-0 and S-A-1. The truth table showing all these possible faults is known as the fault table. This is shown in Table 7-14. Symbol Z_{a0}, Z_{b0} and Z_{c0} represents S-A-0 and Z_{a1}, Z_{b1} and Z_{c1} represents S-A-1.

Table 7-15

Fault-free Inputs and outputs				***Faulty outputs***															
A	B	Y	Z	a_0	Y_{a0}	Z_{a0}	a_1	Y_{a1}	Z_{a1}	b_0	Y_{b0}	Z_{b0}	b_1	Y_{b1}	Z_{b1}	c_0	Z_{c0}	c_1	Z_{c1}
0	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	0	1	1	0
0	1	0	1	0	0	1	1	1	0	0	0	1	1	0	1	0	1	1	0
1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	0	0	1	1	0
1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	1	0

Circles represent respective fault in the circuit. The fault table may be simplified considerably by removing the columns labeled as Y_a, Y_b etc. The reduced fault-table is shown in Table 7-15. The Column of the reduced fault table are modified by taking the modulo-2 sum of the fault free output Z and the outputs under faulty conditions Z_{a0}, Z_{b0}, Z_{c0}, Z_{a1}, Z_{b1}, and Z_{c1} respectively. The modulo-2 sum will be 0, if the outputs are identical; it will be 1, if the outputs differ.

Table 7-16.

Input lines a, b	$Z_{a0} = Z_{b0} = Z_{c0}$	$Z_{a1} \oplus Z$	$Z_{b1} \oplus Z$	$Z_{c1} \oplus Z$
00	0	0	0	1
01	0	1	0	1
10	0	0	1	1
11	1	0	0	0

We find that there is only a simply 1 entry under each of the columns Z_{a0}, Z_{a1} and Z_{b1}. However, in the last column Z_{c1} there are three 1 entries. Thus all the columns show faulty situations. Then the tests to be performed to detect these faults are designated as 00, 11, 10 and 11, respectively. These are the logic inputs to the gates, we have to find a minimum set of tests to be performed on the circuit so that all possible faults can be detected. The tests to be performed in the case of the circuit under consideration may be stated as follows:

1. Test 1: Apply inputs $AB = 11$. Then the condition $Z = 1$ indicates a S-A-0 fault on lines a, b or c. If $Z = 0$, we go for test 2.
2. Test 2: Apply $AB = 01$. If $Z = 1$, a S-A-1 fault exists on line a, $Z = 0$ indicates fault-free situation. So, go to test 3.
3. Test 3: Apply $AB = 10$. With $Z = 1$, fault S-A-1 exists on line b. $Z = 0$ indicates the need for test 4.
4. Test 4: The final test is to apply $AB = 00$. This gives the Z_{c1} faults.

In a digital circuit, positions where faults have occurred must be located for repairing. This can be accomplished by means if fault-location tables. This is a fault modified to include columns that show pairs of faults in modulo-2 addition. This process is repeated for all pairs of faults. The fault location table corresponding to our fault table is given in Table 7-16. We notice that the minimum number of tests to be conducted is three and they are 11, 10 and 01, respectively, for covering all the faults.

Table 7-17. Fault Location Table

Tests	Z_{a0}	Z_{a1}	Z_{b1}	Z_{c1}	$Z_{a0} \oplus Z_{a1}$	$Z_{a0} \oplus Z_{b1}$	$Z_{a0} \oplus Z_{c1}$	$Z_{a1} \oplus Z_{b1}$	$Z_{a1} \oplus Z_{a0}$	$Z_{b1} \oplus Z_{c1}$
00	0	0	0	1	0	0	1	0	1	1
01	0	0	1	1	0	1	1	1	1	0
10	0	1	0	1	1	0	0	1	0	1
11	1	0	0	1	1	1	1	0	0	0

|← Old entries →|← New entries →|

Advantages

(*i*) The method is quite simple and straightforward.

(*ii*) Very easy to test simple logic circuits

(*iii*) Valid even for different faults and tests

Disadvantages

(*i*) With larger and more complex circuits, the testing procedure is quite lengthy, as it requires formulation of elaborate fault tables.

(*ii*) The problem of finding minimal set of tests is highly complicated for larger number of faults.

7-77. Fault Detection by Path-Sensitizing Method

The main idea behind the path-sensitizing procedure will explain by devising a test which detects a S-A-1 fault at input A of Fig. 7-131. Suppose that this path is the only one from A to the circuit output. In order to test a S-A-1 fault at input A, it is necessary to apply a 0 to a and 1's to all remaining inputs to the OR and NOR gates in the path. This ensures that all the gates will allow the propagation of the signal from A to the circuit outputs, and that only this signal will reach the circuit output. This assignment of values is shown in Fig. 7-133. The path is now said to be sensitized.

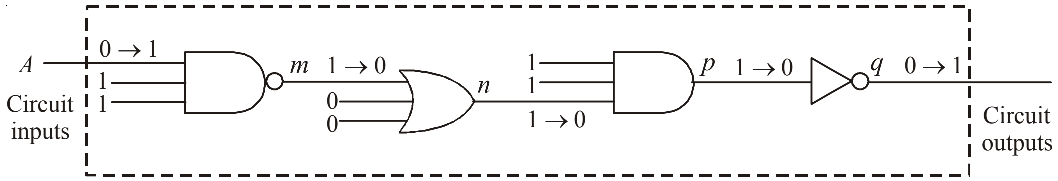


Fig. 7-133.

If input A is S-A-1, then *m* changes from 1 to 0, and this change propagates through connections *n* and *p* and causes *q* to change from 0 to 1. Clearly, in addition to detecting a S-A-1 fault at A, this test also detects S-A-0 faults at *m*, *n* and *p* and a S-A-1 fault at *q*. A S-A-0 fault at A is detected in a similar manner. A 1 is applied to A, while other gate inputs remain as before, this second test would also detect the complementary set of faults to those detected by the previous test on this path. Thus the two tests together are sufficient to detect all S-A-0 and S-A-1 faults on this path.

The basic principle of the path-sensitization method, which is also known as the one-dimensional path sensitization, can thus be summarized as follows:

1. At the site of the fault assign a logical value complementary to the fault being tested.
2. Select a path from the circuit inputs through the site of the fault to a circuit output. The path is said to be sensitized if the inputs to the gates along the path are assigned values so as to propagate to the path output any fault on the wires along the path. This process is called the forward-drive phase of the method.
3. Determine the primary inputs that will produce all the necessary signal values specified in the preceding steps. This is accomplished by tracing the signals backward from each of the gates along the path to the primary inputs. This process is called the backward-trace phase of the method.

SUMMARY

In this chapter, you have learned that

1. A counter is a circuit that uses flip-flops to divide the frequency of the input signal.
2. A counter circuit could be an asynchronous or synchronous depending upon whether the clock input pulses feed all the flip-flops or just one flip-flop in the circuit.
3. The MOD number of a counter (called modulus number) is equal to the number of states that the counter goes through in each complete cycle.
4. A counter with any desired MOD number can be obtained by using an appropriate binary counter and a NAND gate.
5. A digital clock circuit uses binary counters to keep track of and display the time of the day.
6. Several ICs are available that can be used as asynchronous (or ripple) and synchronous counters. Some of them can count up/down and are presettable.
7. The flip-flops can be connected in a shift-register arangement to transfer data to left-to-right or right-to-left.
8. Several IC registers are available which can be classified according to wheter their inputs are parallel, serial or both, and outputs that are parallel or serial.

GLOSSARY

A Synchronous Counter. A counter circuit in which each flip-flop is clocked from its preceeding flip-flop.

BCD Counter. A binary counter that counts from 0000 to 1001 before it recycles.

Decade Counter. A counter that has 10 states.

Frequency division. The use of flip-flop circuits to produce an output waveform whose frequency is equal to the input clock frequency divided by some integer value.

Johnson Counter. A shift register in which the inverted output of the last flip-flop is connected to the input of the first flip-flop.

MOD number. The number of different states that a counter can sequence through. It is also equal to counter's frequency division ratio.

Parallel Counter. See synchronous counter

Parallel data transfer. It is an operation by which several bits of data are transferred simultaneously into a counter or register.

Parallel-in/Parallel-out register. A type of register that can be loaded with parallel data and has parallel outputs available.

Parallel-in/Serial out register. A type of register that can be loaded with parallel data and has one serial output.

Register. A group of flip-flops that are capable of storing data.

Ripple Counter. An asynchronous counter.

Serial in/Parallel out register. A type of register that can be loaded with data serially and has parallel outputs available.

Serial in/Serial out register. A type of register that can be loaded with data serially and has only one serial output.

DESCRIPTIVE QUESTIONS

1. What does the term "asynchronous" mean in relation to counters ? Explain with an example.
2. What is a binary counter? Discuss how a basic four-bit a synchronous binary can be constructed by using four flip-flops. Draw the waveforms associated with a four-bit counter.
3. What do you understand by Module-N counter? Briefly explain with suitable schematic supported by truth table (function table).
4. Design a MOD-8 ripple counter using J-K flip-flops. Draw a timing waveform of the counter.
5. How many states does a MOD-12 counter have? What is the minimum number of J-K flip-flops required to implement the circuitry.
6. What is the difference between the counting sequence of an UP counter and a DOWN counter.
7. Describe briefly, how an asynchronous down-counter circuitry differs from an up-counter circuitry.
8. Explain why a ripple counter's maximum frequency limitation decreases as more number of flip-flops are added to the counter circuit. Illustrate with an example.
9. List the advantages/disadvantages of a synchronous counter over an asynchronous counter.
10. Design a 4-bit synchronous binary counter based on J-K flip-flops.
11. What do we mean by "a presettable counter". Illustrate with an example. Describe the difference between asynchronous and synchronous presetting.
12. Explain briefly, why the decoding gates for an asynchronous counter may have glitches on their outputs. What method is used to eliminate these glitches.
13. How can a ring counter be converted to a Johnson counter.
14. Write the sequence of states for a 3-bit Johnson counter starting with 000.
15. Design a 4-bit ring counter using J-K flip-flops. Explain its operation along with possible states the sesired ring counter is going to produce.
16. Draw a 3-bit Johnson counter (or ring counter) circuit using J-K flip-flops. The input to the circuit is a pulse train of square wave shape. Clearly show the circuit connections and the output waveforms.
17. What is a shift register? Draw the circuit diagram of a four-bit shift register using J-K flip-flops.
18. What are qualitative differences between parallel loading and serial loading in shift registers?
19. Draw the schematic of a 4-bit left shift register with parallel loading using D-type flip-flops.

20. How are shift-right and shift-left registers implemented using D flip-flops. Explain brienfly with suitable diagrams.
21. How can a serial in / parallel out register be used as a serial in / serial out register.
22. How the classification of counters has been done? Design a modulus 16 synchronous UP counter. (*Gauhati Univrsity, 2006*)
23. Describe the operation of a shift register. Also explain the function of a ring counter. (*Gauhati Univrsity, 2006*)
24. Explain the operation of Twisted Ring Counter and give its state diagram with four flipflops. (*Nagpur University, 2008*)
25. Design a 3-bit up-down counter with a direction control M using JK flip flops. (*Nagpur University, 2008*)
26. What is the modulus of the counter? Give difference between Synchronous and Asynchronous Counters. (*Nagpur University, 2008*)
27. Design a sequence detector using Mealy circuit which detects sequence 10010. Assume overlapp-ing permitted. Use D-flip flop. (*Nagpur University, 2008*)
28. Write a short note on Johnson's counter. Explain its operation with waveforms. (*Nagpur University, 2008*)
29. What are the different types of shift resisters? Explain the PIPO and PISO shift registers. (*Nagpur University, 2008*)
30. Differentiate between synchronous and asynchronous counters. (*Nagpur University, 2008*)
31. Describe Universal Shift Registers. (*Mahatma Gandhi University, Jan. 2007*)
32. Give applications of up-down counter. (*Mahatma Gandhi University, Jan. 2007*)
33. Differentiate between Ring Counter and Johnson Counter. (*Mahatma Gandhi University, Jan. 2007*)
34. Give applications for each type of counters. (*Mahatma Gandhi University, Jan. 2007*)
35. Write note on counter ICs. (*Mahatma Gandhi University, Jan. 2007*)
36. Design mod-11 synchronous counter. Implement it using T-flip flops. (*Mahatma Gandhi University, Jan. 2007*)
37. What are advantages and disadvantages of asynchronous counter over synchronous? (*Mahatma Gandhi University, Dec. 2007*)
38. Draw the circuit diagram, timing diagram and output table for a 4 bit Johnson counter. (*Mahatma Gandhi University, Dec. 2007*)
39. What is an excitation table? Design a mode 5 synchronous binary counter having the following repeated binary sequences. Use K–maps. 0, 4, 2, 1, 0, 4, 2, 1, 0 (*Mahatma Gandhi University, Dec. 2007*)
40. Design a mod-12 ripple counter using JK flip-flops. Using timing diagrams explain its working. (*Mahatma Gandhi University, Dec. 2007, RGTU., Dec. 2009*)

41. What are the differences between Counter and Shift register? Give two main functions performed by a shift register.

(*Mahatma Gandhi University, May/Jun. 2006*)

42. Show how you can use a ring counter to get a gate waveform.

(*Mahatma Gandhi University, May/Jun. 2006*)

43. Explain any two distinct applications of shift register.

(*Mahatma Gandhi University, Nov. 2005*)

44. Design and explain a modulo-8 binary up/down ripple counter constructed using JK flip flops and NAND, gates.

(*Mahatma Gandhi University, Nov. 2005*)

45. With necessary diagram, explain the action of a 4-bit serial in, serial-out shift resister.

(*Mahatma Gandhi University, Nov. 2005*)

46. Draw and explain the circuit of a twisted ring counters constructed using JK flip flops.

(*Mahatma Gandhi University, Nov. 2005*)

47. Draw a Mod-3 counter and with the help of waveforms explain how it functionality?

(*Mahatma Gandhi University, May 2005*)

48. Bring out the advantages and disadvantages of the synchronous counter compared with asynchronous counter.

(*Mahatma Gandhi University, May 2005*)

49. Design a 4 bit synchronous counter to generate the output sequence 1, 5, 7, 9, 12 and then this sequence repeats. Realise the counter using JK flip-flops and gates.

(*Mahatma Gandhi University, May 2005*)

50. Using D-flip flops, design a modulo 10 gray code up counter.

(*Mahatma Gandhi University, May 2005*)

51. Darw a 4 bit shift register which is fed serially and read parallelly. How can this be mod to be fad parallelly and read serially?

(*Mahatma Gandhi University, May 2005*)

52. State and explain two applications of shift register.

(*VTU, Jul. /Aug. 2005*)

53. Design mod-4 ripple counter with initial state is $(011)_2$. Draw timing diagram for the same.

(*VTU, Jul. /Aug. 2005*)

54. Design a synchronous mod – 3 counter with the following binary sequence using clocked condition JK flipflops. Count Sequence: 0. 1. 2. 0. 1. 2...........

(*VTU, Jan. /Feb. 2006*)

55. Differentiate between ripple and synchronous counter.

(*VTU, Dec. /Jan. 2008*)

56. Design a synchronous module N counter and sketch the output waveform.

(*VTU, Dec. /Jan. 2008*)

57. With a block diagram describe a 3-bit Johnson twisted ring counter. Draw the sequence diagram and indicate the valid and invalid states.

(*VTU, Dec. /Jan. 2008*)

58. Give expression for maximum frequency of operation of n-bit asynchronous and synchronous counters.

(*PTU, May 2007*)

59. Compare binary counters with non-binary counters.

(*PTU, May 2007*)

60. Draw logic circuit diagram for 3-bit synchronous up-down counter with clear input, start input and done output. The counter should produce done output after completion of counter in either direction.

(*PTU, May 2007*)

61. Design a mod 30 Synchronous up counter.

(*PTU, May 2007*)

62. Explain why there may be a race condition in a shift register.

(*PTU, May 2008*)

63. What is a ripple counter?

(*PTU, May 2008*)

64. Write the short note on the following

(*i*) Counter design with state equation and state diagrams

(*ii*) Classification and characteristics of memories

(*PTU, May 2008*)

65. Differentiate between synchronous and asynchronous counters.

(*PTU, Dec 2008*)

66. What is buffer register?

(*PTU, Dec 2008*)

67. Suggest four applications of shift registers.

(*PTU, Dec 2008*)

68. Explain what is universal shift register? Explain its working.

(*PTU, Dec 2008*)

69. What is universal shift register?

(*PTU, May 2009*)

70. Discuss the operation of twisted ring counter.

(*PTU, Dec 2009*)

71. Draw the two bit ripple counter and convert this into a 2 bit ring counter.

(*Anna University, Nov. /Dec. 2005*)

72. Draw a six stage ring counter and explain its operation. Mention about the use of presetting the counter.

(*Anna University, Nov. /Dec. 2005*)

73. Draw a 5 flip flop shift counter, its truth table and waveform. Explain its operation as a decade counter.

(*Anna University, Nov. /Dec. 2005*)

74. Draw a scale of 8 ripple counter.

(*Anna University, May /Jun. 2006*)

75. Draw an asynchronous decade counter and explain its operation drawing neat waveforms.

(*Anna University, May /Jun. 2006*)

76. Draw a 3 bit reversible counter and explain its operation.

(*Anna University, May /Jun. 2006*)

77. Draw a 4 bit serial-in serial-out shift register and draw its waveforms.

(*Anna University, May /Jun. 2006*)

78. Define synchronous counter.

(*Anna University, Nov. /Dec. 2006*)

79. Draw the four bit Johnson counter and explain the operation.

(*Anna University, Nov. /Dec. 2006*)

80. Draw a four bit serial in serial out shift register and explain.

(*Anna University, Nov. /Dec. 2006*)

81. Draw the eight bit serial in parallel out shift register and explain its operation.

(*Anna University, Nov. /Dec. 2006*)

82. Classify the registers with respect to serial and parallel input output.

(*Anna University, May /Jun. 2006*)

83. Draw the four bit Johnson counter and explain the operation.

(*Anna University, May /Jun. 2006*)

84. Draw the logic diagram for a 5 bit serial load shift register using D flip-flop and explain.

(*Anna University, May /Jun. 2006*)

85. Draw the logic diagram of a 4 bit parallel load recalculating shift register and explain.

(*Anna University, May /Jun. 2006*)

86. Design a 5 bit shift register using 5 master-slave SR flip.

(*Anna University, Apr. /May 2008*)

87. Design a 3 bit synchronous counter using JK flip flop.

(*Anna University, Apr. /May 2008*)

88. Design an asynchronous circuit using JK flip flop that will O/P only the first pulse received and will ignore and other pulses.

(*Anna University, Apr. /May. 2008*)

89. Discuss the difference between synchronous circuits with example.

(*GBTU/MTU, 2006-07*)

90. What are the synchronous and asynchronous sequential circuit ? Write the procedure for the analysis of a sequential circuit.

(*GBTU/MTU, 2005-06*)

91. Explain state table with respect to sequential circuit with suitable example.

(*GBTU/MTU, 2006-07*)

92. What are the uses of asynchronous input of flip-flop ?

(*GBTU/MTU, 2004-05*)

93. What are synchronous and asynchronous sequential circuit? Write the procedure for the analysis of a sequential circuit. Draw the state diagram for a sequence generator which produce an output '1' every time the sequence 0101 is detected and output '0' at all other times. Design the circuit also.

(*GBTU/MTU, 2005-06*)

94. Draw the block diagram of asynchronous segmental circuit and explain its working. Also list the different techniques used for state assignment.

(*GBTU/MTU, 2009-10*)

95. Define critical race and non-critical race. What is hazard? Also explain the elimination of hazard.

(*GBTU/MTU, 2009-10*)

96. What are the steps for analysis of synchronous sequential circuit?

(*GBTU/MTU, 2006-07*)

97. Write a short note for the analysis of blocked sequential circuit.

(*GBTU/MTU, 2006-07*)

98. Write the procedure for design of sequential circuits.
(*GBTU/MTU, 2006-07*)

99. Design a 3-bit binary UP/DOWN counter with a direction control M. Use J-K flip- flops.
(*GBTU/MTU, 2002*)

100. Using any type of flip-flops, design a 4-bit synchronous counter that upcounts in gray code.
(*GBTU/MTU, 2003*)

101. Differentiate between synchronous and asynchronous counters.
(*Jamia Milla Islamia University, 2007*)

102. What is race around condition and how it can be eliminated ?
(*Nagpur University, 2004*)

103. Compare combinational and sequential circuits.
(*Nagpur University, 2004*)

104. Define and distinguish between
Synchronous and Asynchronous systems
(*Gauhati University, 2003*)

105. Design a 4 bit up-down binary ripple counter with a control line for up and down counting. Explain the operation with truth table.
(*Gauhati University, 2003*)

106. Write a short notes on IC-7495, 4 bit shift register.
(*Nagpur University, 2004*)

107. Write short notes in Ring Counter and Twistes Ring Counter.
(*Nagpur University, 2004*)

108. What in the significance of combinational counters?
(*Jamia Millia Islamia University, 2007*)

109. Design a modulo-8 up counter with positive trigger using T-design procedure.
(*Jamia Millia Islamia University, 2007*)

110. Design a modulo-6 counter using J-K flip-flop. Explain its action by writing a truth table and draw waveform for the outputs of the flip-flops.
(*Nagpur University, 2004*)

111. Write short notes on parallel in serial out shift register.
(*Gujarat Technological University, Dec. 2009*)

112. Define and distinguish between
Combinational and sequential circuit
(*Gauhati University, 2003*)

113. Explain working of 4-bit binary ripple counter.
(*GTU., Dec. 2011*)

114. Draw and explain block diagram of 4-bit bidirectional shift register with parallel load.
(*GTU., Dec. 2011*)

115. Explain the procedure followed to analyze a clocked sequential circuit with suitable example.
(*GTU., Dec. 2011*)

116. Give classification of counters and explain asynchronous.
(*GTU., May 2011*)

117. Draw the diagram of a 4-bit left right shift register.
(*WBUT., 2011*)

118. Design a mod 5 synchronous counter using J-K flip-flops.

(*WBUT., 2011*)

119. Design a mod-7 ripple counter using C-R lines of J-K flip-flops.

(*WBUT., 2011*)

120. Draw the timing diagram of a 3-bit ring counter.

(*WBUT., 2011*)

121. Design a 4-bit up/down synchronous serial counter using J-K flip-flops and other necessary logic gates.

(*WBUT., 2011*)

122. Draw the circuit diagram of a mod-8 ripple counter using J-K flip-flops. Draw the output waveforms also. Obtain the state table and hence show the corresponding state diagram.

(*WBUT., 2011*)

123. Design a mod-11 ripple counter using J-K flip-flop.

(*RGTU., June 2010*)

124. Design a mod-12 binary counter using J-K flip-flop.

(*RGTU., June 2009*)

125. Explain the following terms :

(*i*) Synchronous and asynchronous counters

(*ii*) Johnson and ring counter

(*iii*) Binary ripple counter

(*RGTU., Dec. 2010*)

126. Design a BCD Ripple Counter.

(*RGTU., June 2011*)

127. Differentiate between synchronous and asynchronous counters. Design a 4-bit up/down counter.

(*RGTU., June 2011*)

TUTORIAL PROBLEMS

1. A 4-bit ripple counter shown in Fig. 7-134 starts off in the 0000 and then clock pulses are applied. Sometime later the clock pulses are removed and the counter flip-flops read 0011. How many clock pulses have occurred.

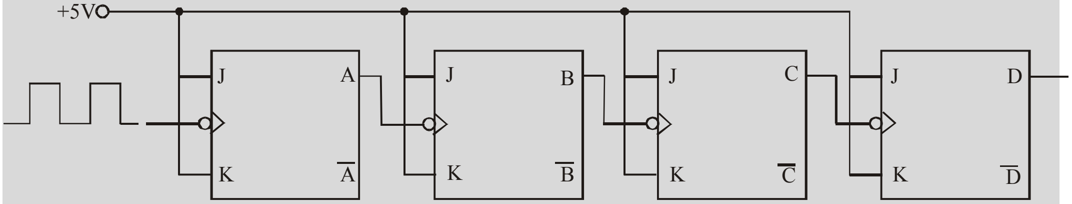


Fig. 7-134

(**Ans.** 3, 19, 37, 53, ……………..)

2. A 5-bit binary counter starts in the 00000 state. Determine the count after 144 pulses. (Hint : The count will recycle every 32 pulses.)

(**Ans** : $16_{10} = 10000_2$)

3. A binary counter is being pulsed by a 100 kHz clock signal. The output frequency from the last flip-flop is 3.125 kHz. Determine (*a*) the MOD number (*b*) the number of flip-flops used in the counter and (*c*) the counting range.

(**Ans.** (a) 32, (b) 5 and (c) – 32 to 31)

4. Use J-K flip-flops and any other necessary logic to construct a MOD-28 ripple counter.

5. Design a MOD-3 and a MOD-2 counter and use it to construct a MOD-6 counter.

(MOD-3 counter must progress through 00 $\rightarrow$ 01 $\rightarrow$ 10 $\rightarrow$ 11)

(*a*) Assume that all flip-flops are initially LOW and sketch the waveforms at each flip-flop output for 12 clock cycles of the input

(*b*) Construct the state transition diagram and show that it is not a normal binary sequence.

6. Using the logic symbol of 7493 shown in Fig. 7-135, show how it can be used as a MOD-12 counter.

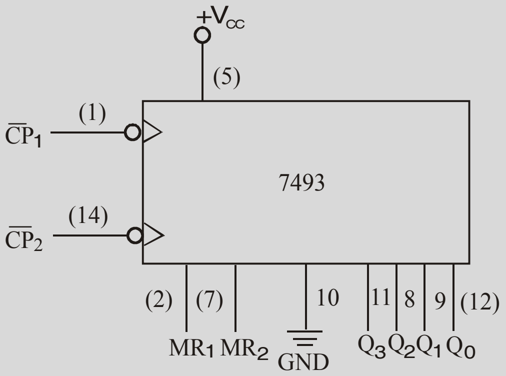


Fig. 7-135.

7. Using the logic symbol of 7493 shown in Fig 7-135, show how it can be used as a MOD-13 counter.

8. Show how 7493 can be used to produce a 1.2 kHz output from a 18 kHz input signal.

9. Fig. 7-136 shows two 74293 's combined to provide frequency division by a certain factor. Determine the frequency output at Z.

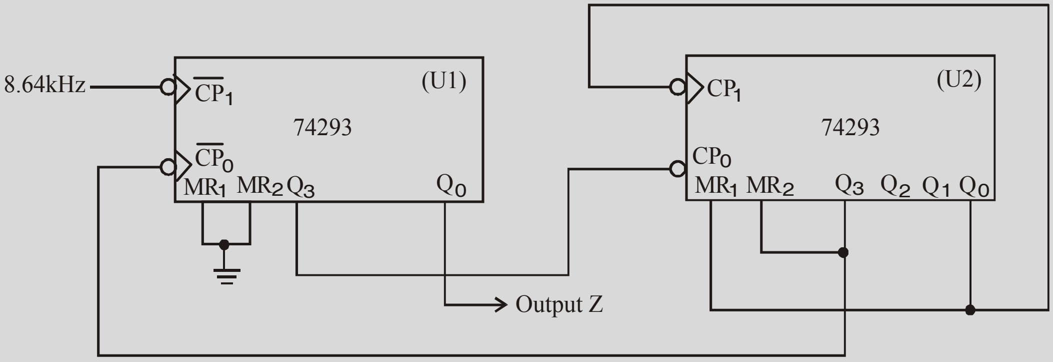


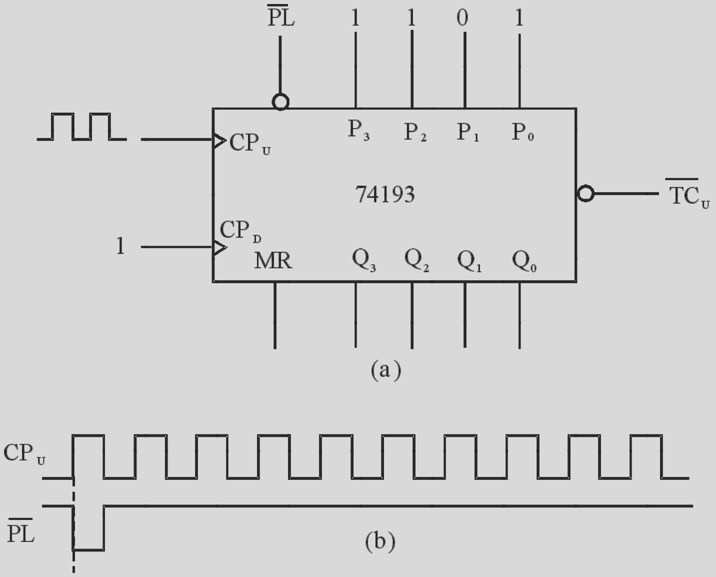
Fig. 7-136.

(**Hint :** The flip-flop outputs of Q_1-Q_3 of 74293 (U1) are used as MOD-8. Thus the frequency at its Q_3 output is 8.64 kHz / 8 = 1.08 kHz. The second 74293 is wired as MOD-9 counter. So its Q_3output will be 1.08 kHz / 9 = 120 Hz. This signal is connected to the Q_0 flip-flop of the first 74293 which divides the frequency in half to produce a frequency of 60 Hz).

10. How many cascaded BCD counters are needed to be able to count up to 6000? How many flip-flops does this require? Compare this with the number of flip-flops required for a normal binary counter to count up to 6000.

[**Hint :** One BCD stage per decimal digit is required. Thus four BCD counters are required which means total of 16 flip-flops. A normal binary counter would require 13 flip-flops to count to 6000 because $2^{12} = 4096$ and $2^{13} = 8192$].

11. Fig. 7-137 (a) shows the IC 74193 wired as an up counter. The parallel data inputs are permanently connected as 1101 and the CP_U, $\overline{PL}$ and MR input waveforms are as shown in Fig. 7-137 (b), Assuming the counter initially in the 0000 state, sketch the counter output waveforms.



(a)

(b)

Fig. 7-137.

12. Sketch the diagram for a five-bit ring counter using J-K flip-flops.

MULTIPLE CHOICE QUESTIONS

1. Asynchronous counters are known as

(*a*) ripple counters (*b*) multiple clock counters

(*c*) decade counters (*d*) modulus counters

2. An asynchronous counter differs from a synchronous counter in

(*a*) the MOD number (*b*) the method of clocking

(*c*) the type of flip-flops used (*d*) the number of states in a sequence

3. The MOD number of a counter is

(*a*) the maximum possible number of states (*b*) the actual number of states in a sequence

(*c*) the number of flip-flops (*d*) the number of times it recycles in a second

4. A 3-bit binary country has a maximum MOD number of

(*a*) 3 (*b*) 6 (*c*) 8 (*d*) 16

5. A MOD – 13 counter must have

(*a*) 13 flip-flops (*b*) 3 flip-flops

(*c*) 4 flip-flops (*d*) synchronous clocking

6. A N-stage ripple counter will count up to

(*a*) 2^N (*b*) 2^{N-1} (*c*) N (*d*) 2^{N+1}

7. In a 4-bit ripple counter, the external clock is applied to the

(*a*) clock input of first-stage flip-flop (*b*) clock input of second-stage flip-flop
(*c*) clock input of third-stage flip-flip (*d*) clock input of fourth-stage flip-flop

8. In a ripple counter made of positive edge-triggered flip-flop, if output of flip-flop is fed to clock input of next stage flip-flop, this counter results in,

(*a*) up-counter (*b*) down-counter (*c*) no-count (*d*) none-of these

9. Which of the following is an invalid state of an 8421 BCD counter?

(*a*) 0010 (*b*) 0101 (*c*) 1000 (*d*) 1100

10. A 10 MHz clock frequency is applied to a cascaded counter consisting of a MOD-5 counter, a MOD-8 counter and two MOD-10 counters. The lowest output frequency possible is,

(*a*) 10 kHz (*b*) 2.5 kHz (*c*) 5 kHz (*d*) 25 kHz

11. A synchronous binary counter

(*a*) recieves clock at flip-flop representing LSB
(*b*) recieves clock at flip-flop representing MSB
(*c*) recieves clock at all flip-flops
(*d*) none of the above.

12. The IC 7490 is a

(*a*) synchronous decade counter (*b*) synchronous div-by-16 counter
(*c*) asynchronous decade counter (*d*) asynchronous div-by-16 counter.

13. A divide – by – 10 ring counter requires a minimum of

(*a*) ten flip-flops (*b*) five flip-flops
(*c*) four flip-flops (*d*) twelve flip-flops

14. A divide–by–10 Johnson counter requires

(*a*) ten flip-flops (*b*) four flip-flops
(*c*) five flip-flops (*d*) twelve flip-flops

15. A shift register consists of

(*a*) a latch (*b*) a flip-flop
(*c*) a byte of storage (*d*) four bits of storage

16. To serially shift a byte of data into a shift register, there must be

(*a*) one clock pulse (*b*) one load pulse
(*c*) eight clock pulses (*d*) one clock pulse for each 1 in the data.

17. To parallel load a byte of data into a shift register, there must be

(a) one clock pulse (*b*) one clock pulse for each 1 in the data
(*c*) eight clock pulses (*d*) one clock pulse for each 0 in the data

18. With a 100 kHz clock frequency, eight bits can be serially entered into a shift register in

(*a*) 80 µs (*b*) 8 µs (*c*) 80 µs (*d*) 10 µs

ANSWERS

1. (*a*)	**2.** (*b*)	**3.** (*b*)	**4.** (*c*)	**5.** (*c*)	**6.** (*b*)
7. (*a*)	**8.** (*a*)	**9.** (*d*)	**10.** (*b*)	**11.** (*c*)	**12.** (*a*)
13. (*a*)	**14.** (*c*)	**15.** (*b*)	**16.** (*c*)	**17.** (*a*)	**18.** (*a*)

8

IC LOGIC FAMILIES AND THEIR CHARACTERISTICS

OUTLINE

Objectives

Upon completion of this chapter, you should be able to:

- Know the different types of IC logic families
- Explain fan-in, fan-out, noise-margin, propagation delay, TTL and CMOS logic levels and power dissipation
- Analyse the internal circuitry of TTL, ECL, I^2L and CMOS logic gates
- Discuss the differences of various subfamilies within both TTL and CMOS ICs.
- Understand the interfacing digital devices from different logic families.

8-1. Introduction

The IC (integrated-circuit) technology has advanced in the last five decades from small-scale integration (SSI) through medium-scale integration (MSI), large-scale integration (LSI), very large-scale integration (VLSI and ULSI) to giga-scale integration (GSI). At present it is possible to integrate more than one million logic gate on a small silicon chip.

There are several advantages of using ICs in digital systems. Some of the important ones are:

1. Reduction of the overall size of the digital circuit.
2. Reduction of the cost because of mass-producing volume of similar ICs.
3. Lower power consumption because of extremely small size of logic circuits.
4. Increased reliability because of reduced external interconnections from one device to another.

However you must know that there are some issues where ICs have not been able to do/handle. These are:

1. ICs cannot handle very large currents or voltages. This is due to the reason that the heat generated in a small space would cause temperatures to rise beyond acceptable limits.
2. ICs cannot easily implement electrical devices such as inductors, transformers, large-size capacitors.

As a result, ICs are more useful in low-power circuit operations. Most of the digital systems in our day-to-day life like personal computers, mobile phones, personal digital assistants (PDAs) etc are all low-power digital systems. It may be carefully noted that operations that require large-power levels or devices that cannot be integrated on a silicon chip are still handled by discrete components.

It is evident from the above discussion that the usage of ICs is widespread in all digital systems. In order to understand the operation of such digital systems, it is important to know the electrical characteristics of the some of the main IC logic families.

8-2. Main IC Logic Families

Most digital systems are designed by combining various logic functions discussed in Chapter 3. All these logic circuits are available in IC modules and are divided into many 'families'. Each family is classified by abbreviations which indicate the type of logic circuit used. For example, RTL means resistors-transistor logic. We will discuss the following seven transistor logic families although the first two are, at present, of historic interest only.

1. **Resistance-transistor logic (*RTL*).** It was the first family group of logic circuits to be developed and packaged in *IC* form in *early* 1960*s*;
2. **Diode-transistor logic (*DLT*).** It followed *RTL* in *late* 1960*s*;
3. **Transistor-transistor logic (*TTL*) OR (*T^2L*).** Was introduced in the early 1970*s*;
4. **Schottky *TTL*.** Was introduced to improve the speed of *TTL*;
5. **Emitter-coupled logic (*ECL*).** It is fastest logic family currently available;
6. **Integrated-injection logic (*I^2L*).** It is one of the latest of the bipolar types of logic;
7. **Complementary metal-oxide semiconductor (*CMOS*).** It has the lowest power dissipation of the currently-available IC logic devices.

The various logic families mentioned above posses different characteristics as discussed below.

8-3. Saturated and Non-saturated Logic Circuits

Those logic circuits in which transistors are driven into saturation are called **saturated** logic circuits or simply **saturated logic.** Those circuits which avoid saturation of their transistors are called **non-saturated logic.**

The disadvantage of saturated logic is the delay that occurs when the transistors is brought out of saturation. When a transistor is saturated, its base is flooded with carriers. Even when base voltage is switched off, the base remains flooded for some time till all carriers leave it. The time required by the carriers to leave the base is called **saturation delay time** (t_s). Obviously, saturated logic circuits have low switching speeds whereas non-saturated type are much faster. *TTL* is the example of a saturated logic whereas *ECL* represents a non-saturated logic.

8-4. Basic Operating Characteristics and Parameters of Logic Families

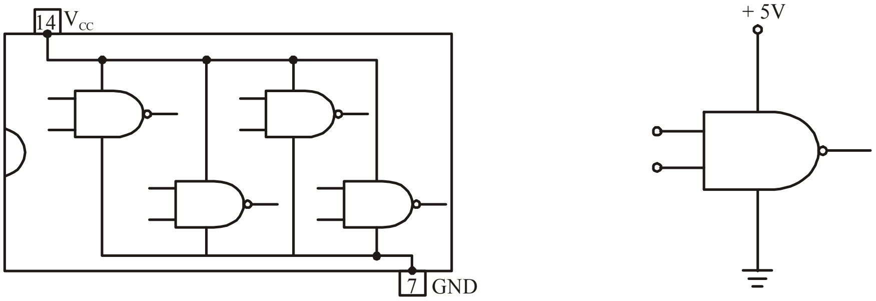
When we work with digital ICs from different logic families, we should be familiar with, not only their logical operation but also with the basic operational properties. Following are the important basic operational properties important from the subject point of view.

1. *DC* supply voltage.
2. *TTL* and *CMOS* logic levels.
3. Noise immunity.
4. Noise margin.
5. Power dissipation.
6. Propagation delay.
7. Speed-power product.
8. Loading and fan-out.

Now we will describe all the above operational characteristics one by one in the following pages.

8-5. DC Supply Voltage

The standard value of the dc supply voltage for *TTL* (*i.e.,* transistor-transistor logic) and CMOS (*i.e.,* complementary metal-oxide semiconductor) device is + 5*V*. For simplicity, the dc supply voltage is usually omitted from the logic circuits. But in practice, it is connected to the V_{CC} or V_{DD} pin of an *IC* package and the ground is connected to the *GND* pin of an *IC* package. Both the voltage and ground are distributed internally to all the logic gates with the package as shown in Fig. 8-1 (*a*). The connections for the single logic gate are as shown in Fig. 8-1 (*b*).



(*a*) (*b*)

Fig. 8-1. Illustrating supply voltage and ground in an IC package

8-6. TTL and CMOS Logic Levels

For TTL circuits, the range of input voltages, that can represent a valid *LOW* (logic 0) and a valid *HIGH* (logic 1) is as shown in Fig. 8-2 (*a*). As seen from this diagram, the range of input voltages that can represent a valid LOW (logic 0) is from 0 to 0.8 V. The LOW input voltage is indicated by the symbol V_{IL}. The lower limit for V_{IL} is represented by $V_{IL(min)}$ and the higher limit for V_{IL}, by $V_{IL(max)}$. The range of input voltages that can represent a valid HIGH (logic 1) is from 2 *V* to V_{CC} (usually 5 *V*). The HIGH input voltage is indicated by smbol V_{IH}. The lower limit for V_{IH} is represented by $V_{IH(min)}$ and the higher limit for V_{IH} by $V_{IH(max)}$. Note that the range of values between 0.8 *V* and 2 *V* is called *indeterminated*. This means that when a input voltage is in this range, it can be interpreted as a HIGH or LOW by the logic circuit. Therefore TTL logic gates cannot be operated reliably when input voltages are in this range.

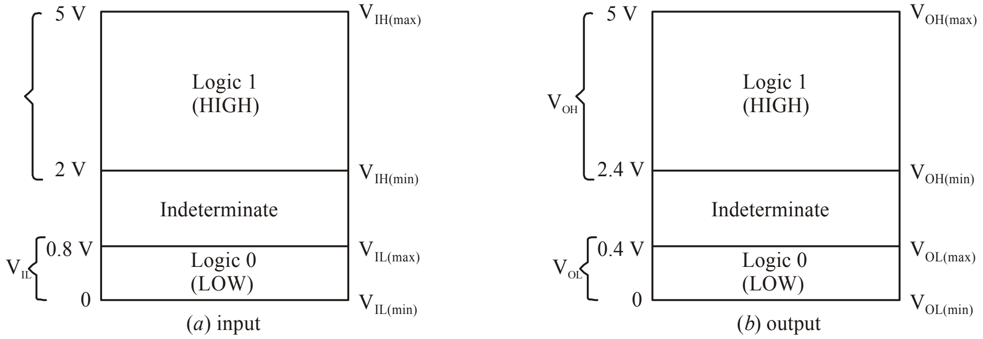


Fig. 8-2. Illustrating TTL logic levels

Fig. 8-2 (*b*) shows the range of *TTL* output voltages that can represent valid HIGH (logic 1) and valid *LOW* (logic 0). As seen the range for logic 1 output is from 2.4 *V* to 5 *V* and logic 0 output is from 0 to 0.4 *V*. Note again that the range of values between 0.4 *V* and 2.4 *V* is indeterminate. Also note that the output voltage for logic 1 is indicated by the symbol V_{OH}. The lower limit for V_{OH} is represented by $V_{OH(min)}$ and the higher limit for V_{OH} by $V_{OH(max)}$. Similarly the output voltage for logic 0 is indicated by the symbol V_{OL}. The lower limit for V_{OL} is represented by $V_{OL(min)}$ and the higher limit for V_{OL} by $V_{OL(max)}$.

It may be noted from Fig. 8-2 (*a*) and (*b*) that the minimum HIGH output voltage, $V_{OH(min)}$ is greater than the minimum HIGH input voltage $V_{IH(min)}$. On the other hand, the maximum LOW output voltage, $V_{OL(min)}$ is less than the maximum LOW input voltage, $V_{IL(max)}$.

The input and output voltages for a device from HCMOS (*i.e.* High-speed CMOS) logic family for V_{DD} = 5 *V* as shown in Fig. 8-3 (*a*) and (*b*) respectively.

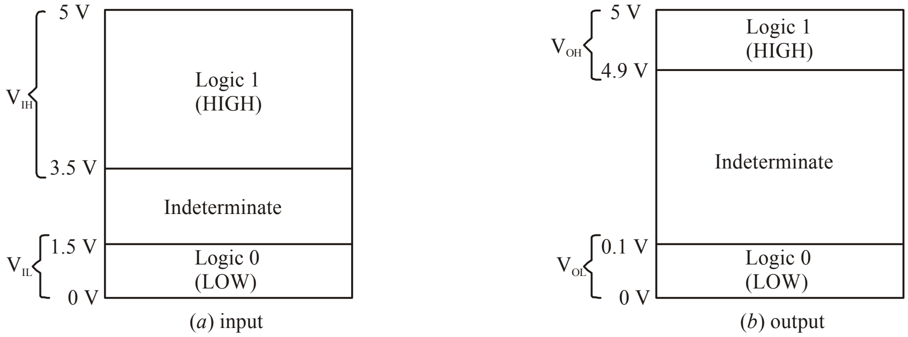


Fig. 8-3. Illustrating CMOS logic levels

Notice that the input and output voltage values in the HCMOS device are different from that of a TTL device. In a HCMOS device, $V_{IL(max)}$ is 1.5 *V* (its value is 0.8 *V* in TTL), $V_{IH(min)}$ is 3.5 *V* (its value is 2 *V* in TTL), $V_{OL(max)}$ is 0.1 *V* (its value is 0.4 *V* is TTL), $V_{OH(min)}$ is 4.9 *V* (its value is 2.4 *V* in TTL).

8-7. Noise Immunity

The noise immunity of a logic circuit refers to the circuit's ability to tolerate noise without causing a false change in its output voltage. The noise voltage is produced by stray electric and magnetic fields on the connecting wires between logic circuits. Sometimes, too much noise voltage cause the voltage at the input of the logic circuit to drop below $V_{IH(min)}$ or rise above $V_{IL(max)}$. This could produce unpredictable operation in a logic circuit.

8-8. Noise Margin

A quantitative measure of a circuit's noise immunity is called noise margin. It is expressed in volts. There are two values of noise margin specified for a given logic circuit as described below.

1. The high level noise margin (V_{NH})
2. The low level noise margin (V_{NL})

These parameters are shown in Fig. 8-4 and are given by the equations,

$$V_{NH} = V_{OH(min)} - V_{IH(min)}$$

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

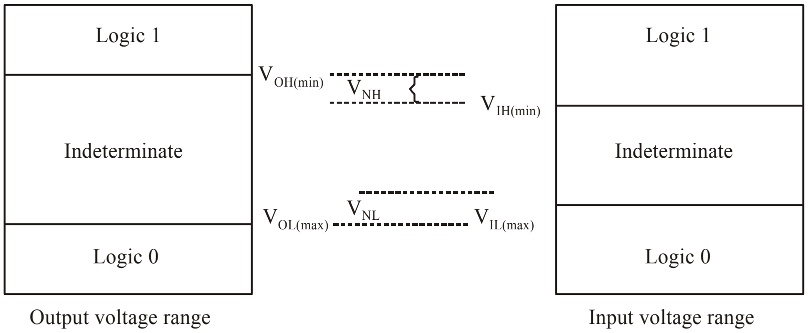


Fig. 8-4. Illustrating high-level and low-level noise margin

Example 8-1. *Table shows the input/output voltage specifications for the standard TTL family.*

Parameter	*Min* (*V*)	*Typical* (*V*)	*Max* (*V*)
V_{OH}	2.4	3.4	—
V_{OL}	—	0.2	0.4
V_{IH}	2.0	—	—
V_{IL}	—	—	0.8

Using these values, find (*a*) *the maximum value of noise spike that can be tolerated when a HIGH output is driving an input,* (*b*) *the maximum value of noise spike when a LOW output is driving an input.*

Solution. The maximum value of the noise spike, when driven by a HIGH output,

$$V_{NH} = V_{OH(min)} - V_{IH(min)} = 2.4 - 2.0 = 0.4\,\text{V}$$

and the maximum value of the noise spike, when driven by a LOW output,

$$V_{NL} = V_{IL(max)} - V_{OL(max)} = 0.8 - 0.4 = 0.4\,V$$

It is observed that a TTL gate is immunate to 0.4 V of poise for both the HIGH and LOW input states.

8-9. Power Dissipation

As a matter of fact, all logic gates draw current from the dc supply voltage for its normal operation. When the logic gate is in the HIGH output state, it draws an amount of current, I_{CCH}, as shown in Fig. 8-5 (*a*) and when in LOW output state, it draws an amount of current I_{CCL} as shown in Fig. 8-5 (*b*).

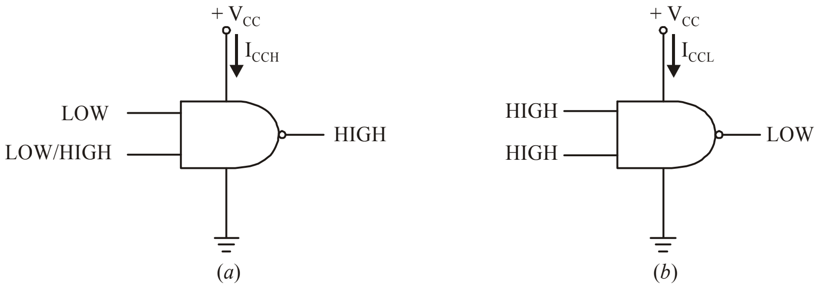


Fig. 8-5. Illustrating I_{CCH} and I_{CCL} in a logic gate

The power dissipation of a logic gate is given by the product of the dc supply voltage (V_{CC}) and the amount of current drawn from the supply (*i.e.,* I_{CCH} or I_{CCL}). Thus power dissipation is given by :

$$P_{\text{D}} = V_{CC} \cdot I_{\text{CCH}} \text{ or } V_{\text{CC}} \cdot I_{\text{CCL}}$$

For example, if I_{CCH} is 2.5 *mA* when V_{CC} is 5 *V*, the power dissipation,

$$P_{\text{D}} = V_{\text{CC}} \cdot I_{\text{CCH}} = 5V \times 2.5\,mA = 12.5\,mW.$$

Usually, the logic gate operates with the inputs, which keep on changing with time (*i.e.,* the input is pulsed). Accordingly the output of a logic gate also switches back and forth between HIGH and LOW. Because of this, the amount of current drawn from the dc supply also varies between I_{CCH} and I_{CCL}. In such a situation, we calculate the average power dissipation. The average power dissipation depends upon the duty cycle and is usually specified for a duty cycle of 50%, the output is HIGH half the time and LOW the other half. Therefore average supply current is :

$$I_{\text{CC}} = \frac{I_{\text{CC}} + I_{\text{CC}}}{2}$$

and the average power dissipation is,

$$P_{\text{D}} = V_{\text{CC}} \cdot I_{\text{CC}}$$

Example 8-2. *A TTL logic gate draws* 2 *mA when its output is HIGH and* 3.5 *mA when its output is LOW. Calculate the average power dissipation if the supply voltage is* 5 *V and the logic gate is operated on* 50% *duty cycle.*

Solution. The average supply current,

$$I_{\text{CC}} = \frac{I_{\text{CCH}} + I_{\text{CCL}}}{2} = \frac{2mA + 3.5\,mA}{2} = 2.75\,mA$$

∴ The average power dissipation,

$$P_{\text{D}} = V_{\text{CC}} . I_{\text{CC}} = 5r \times 2.75\,mA = \mathbf{13.7\,mW}$$

8-10. Power Dissipation versus Frequency

Fig. 8-6 shows a graph of power dissipation versus frequency for a *TTL* and a *CMOS* logic gate. As seen from this graph, the power dissipation in a *TTL* circuit is essentially constant over its range of operating frequencies. However, the power dissipation in a *CMOS* circuit is frequency dependent. That is, it is extremely low under zero frequency (or dc) conditions. But the power dissipation increase as the frequency increase. For example, the power dissipation of a typical *TTL* logic gate is a constant 2 *mW*. On the other hand, power dissipation of typical *CMOS* logic gate is 0.0025 *mW* under static (or dc) conditions and 0.17 *mW* at 100 *kHz*.

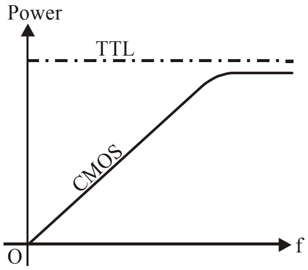


Fig. 8-6. Power dissipation versus frequency for TTL and CMOS logic gate

8-11. Propagation Delay

When a signal passes (*i.e.,* propagates) through a logic circuit, it always experiences a finite time delay as shown in Fig. 8-7. It shows that the change in output level occurs after a short time, (called the propagation delay time), later than the change in input level that caused it. There are two propagation delay times specified for logic gates.

1. **t_{PLH}.** It is the time interval between a designated point on the input pulse and the corresponding point on the output pulse when the output is changing from LOW to HIGH (or 0 to 1) as shown in Fig. 8-8.
2. **t_{PHL}.** It is the time interval between a designated point on the input pulse and the corresponding point on the output pulse when the output is changing from HIGH to LOW (or 1 to 0) as shown in Fig. 8-8.

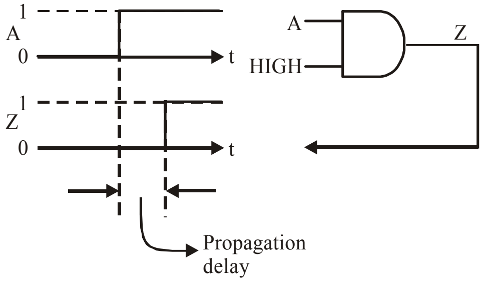


Fig. 8-7. Propagation delay

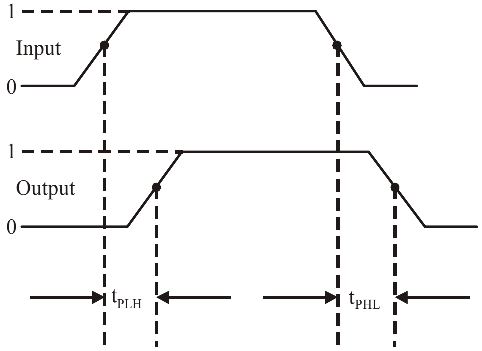


Fig. 8-8. Illustrating t_{PLH} and t_{PHL}

It may be noted that the propagation delay times are indicated in Fig. 8-8 with 50% points on the pulse edges used as references.

The propagation delay time of a logic gate limits its maximum operating frequency. The greater the propagation delay time of a logic gate, the lower is its maximum operating frequency. This means a high speed logic gate is a one that has a small propagation delay time. For example, a logic gate with a delay time of 2 ns is faster than a logic gate that has a delay time of 10 ns.

8-12. Speed-Power Product

It provides a basis for comparison of logic circuits when the propagation delay and power dissipation are important considerations in the selection of the type of logic family to be used in certain application. The speed power product is expressed in picojule (pJ). It may be noted carefully that the lower the speed-product, the better is the value. Typically the *CMOS* family has much lower value of speed-power product as compared to the *TTL* logic family. For example, a typical *CMOS* logic family has a speed-power product of 1.5 pJ at 100 Hz while a typical *TTL* has speed-power product of 20 pJ.

8-13. Loading and Fan-out

When the output of any logic gate is connected to one or more inputs of other logic gates, a load on the driving gate is created. This is shown in Fig. 8-9. Here the output of a logic gate (labeled as 1) is connected to the inputs of 3 other logic gates (labeled as 2, 3 and 4). Note that the logic gate labeled 1 is called a *driving gate* while the logic gates labeled as 2, 3 and 4 are called *load gates.*

In any logic family, there is a limit to the number of load gate inputs that a given logic gate can drive. This limit is called the *fan-out* of the logic gate. Now we will study the loading and fan-out in *TTL* and *CMOS* logic families in more detail.

Loading and fan-out in TTL family : A *TTL* gate that acts as a driving gate *sources* (*i.e.,* supplies) current to a load gate input in the logic HIGH state and *sinks* (*i.e.,* recieves) current from the load gate in the logic LOW state. Fig. 8-10 (*a*) illustrates the current sourcing and (*b*) shows current sinking in the logic gates.

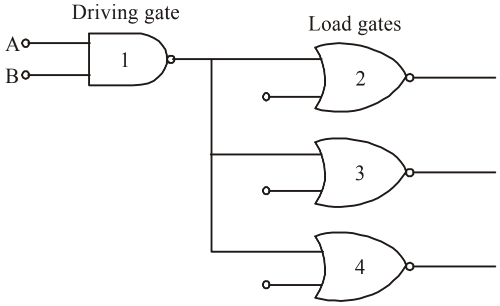


Fig. 8-9. Driving gate and load gates

Note that I_{IH} is the current supplied by the driving gate to the load gate when the input of the load gate is HIGH similarly. I_{IL} is the current recieved by the driving gate from the load gate when the input of the load gate is LOW.

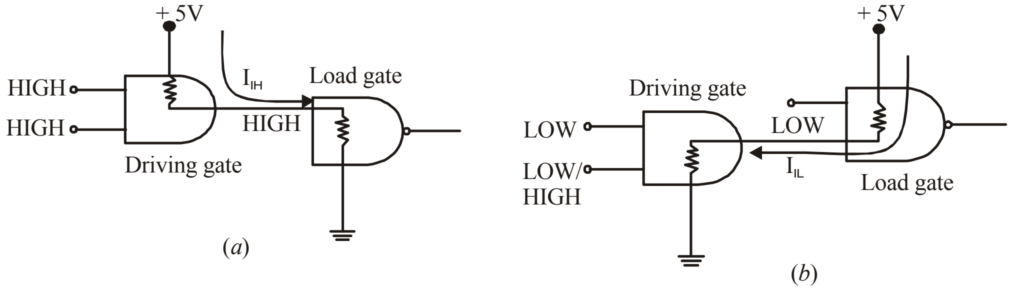


Fig. 8-10. Illustrating I_{IH} and I_{IL} currents

As more and more number of load gates are connected to the driving gate, the loading on the driving gate increases. The total source current increases with each load gate input that is added as illustrated in Fig. 8-11. As the source current increase, the internal voltage drop of the driving gate increases V_{OH}. This causes the voltage drop V_{OH} to decreases. If a large number of load gate inputs are connected, drops below $V_{OH(min)}$. As a result of this, HIGH level noise margin is reduced, thus affecting the specified operating characteristics of the gate. Moreover, as the total source current increases, the power dissipation of the driving gate increases.

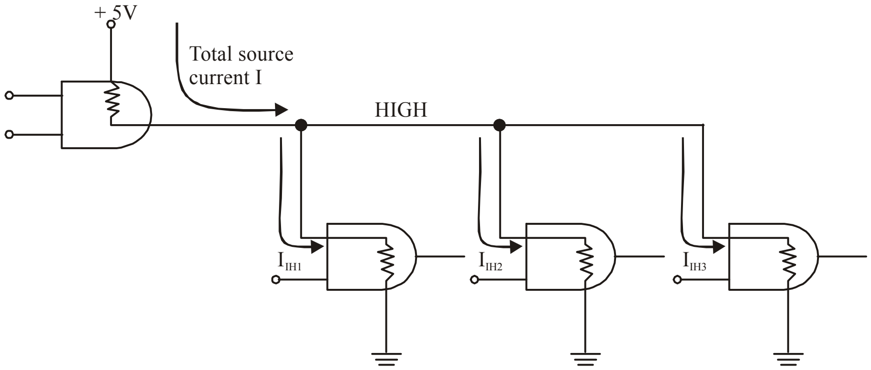


Fig. 8-11.

The maximum number of load gate inputs that can be connected without affecting the specified operational characteristics of the driving gate is called *fan-out*. Its value is important for designing logic circuits. For example, the standard *TTL* has a fan-out of 10. One input of the same logic family as the driving gate is referred to as a *unit load*. This we can also say that a standard *TTL* has a fan-out of 10 unit loads.

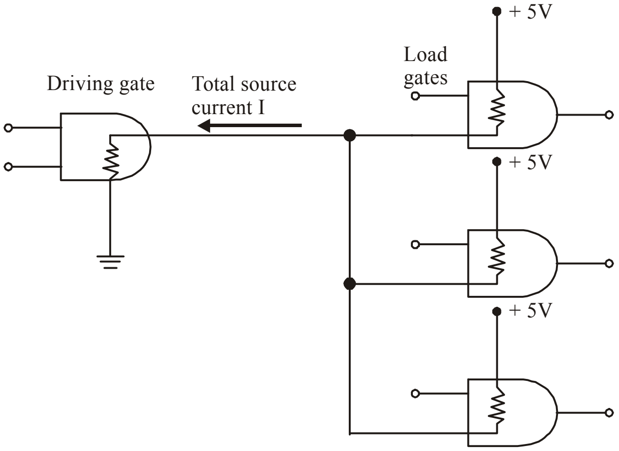


Fig. 8-12.

The total sink current also increases with each load gate input that is added as shown in Fig. 8-12. As this current increases, the internal voltage drop of the driving gate increases, causing V_{OL} to increase. If a large number of loads are added, V_{OL} exceeds $V_{OL(max)}$ and the LOW-level margin is reduced.

As a matter of fact, in *TTL*, the current sinking capability (in the *LOW* state) is the limiting factor in determining the fan-out.

Loading and fan-out in CMOS. Loading in *CMOS* logic family differs from that in *TTL*. It is because of the fact that the field-effect transistors used in *CMOS* logic family present a predominantly capacitive load to the driving gate as shown in Fig. 8-13. In this case, the limitations are the charging and discharging times associated with the output resistance of the driving gate and the input capacitance of the load gates.

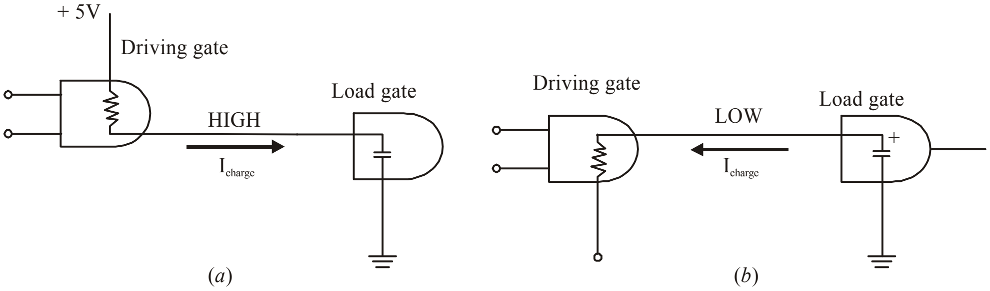


Fig. 8-13.

When the output of the driving gate is HIGH, the input capacitance of the laod gate is charging through the output resistance of the driving gate. When the output of the driving gate is LOW, the capacitance is discharging. When more and more number of load gate inputs are added to the driving gate output, the total capacitance increases because the input capacitances effectively appear in parallel. This increase in capacitance increases the charging and discharging times. As a result, the maximum operating frequency. The smaller the number of load gate inputs, the greater the maximum operating frequency.

8-14. RTL Circuit

It is a *saturated* logic. It uses only transistors and resistors as circuit elements and also resistances in the input to each base. This family is based on the *NOR* circuit shown in Fig. 8-14. All other members of the family are made up of *NOR* cells or variations on them.

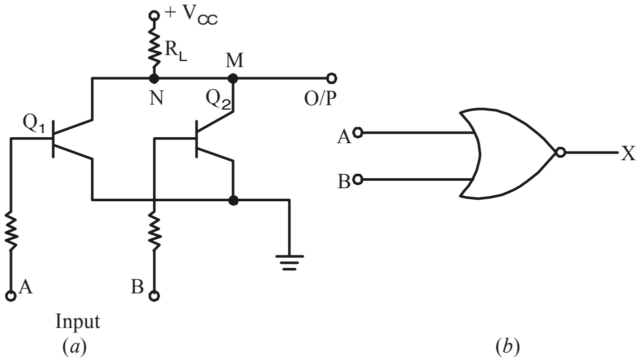


Fig. 8-14. RTL NOR Circuit

Circuit Operation

We will assume ideal transistors. When both inputs *A* and *B* are 0 *V* (or logic 0) both transistors are turned *OFF*, hence point *M* goes to + V_{CC} so that output is logic 1.

If either or both input terminals are at + V_{CC} *i.e.* are high (or logic 1), one or both transistors would be fully turned *ON* (*i.e.* saturate) thereby reducing the voltage of point *N* to almost 0 *V*. Hence, output would be at logic 0.

It is seen that the output is at logic 1 only when *both inputs are at logic* 0 — the *NOR* logic functions as shown in Fig. 8-14 (*b*).

The *RTL* family has the following characteristics :

1. relatively slow speed,
2. low fan-out of 6 and a fan-in of 4,
3. poor noise immunity,
4. expensive since resistors are required to be fabricated,
5. cannot operate at speed above 4 MHz.

8-15. DTL Circuit

It is a *saturated* logic because transistors between cut-off and saturation. It was the next family to be introduced after *RTL*. It consists of diodes, resistors and transistors. The basic gate of this family performs *NAND* function. As shown in Fig. 8-15 (*a*) the circuit basically consists of a diode *AND* gate followed by a transistor inverter which leads to a *NAND* gate.

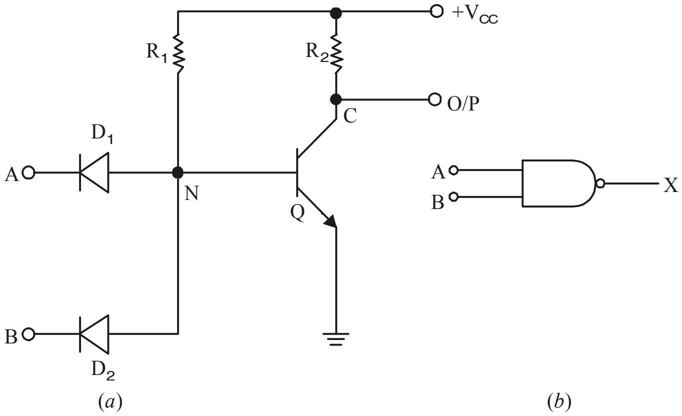


Fig. 8-15. DTL NAND Circuit

Circuit Operation

1. When both D_1 and D_2 have positive voltage applied to them (logic 1), neither conducts and *Q* is turned *ON* by the current provided by V_{CC} through R_1. Since *Q* becomes saturated, point *C* is brought 0*V* (logic 0). Hence output goes logic 0.

2. If either or both inputs are at 0*V* (logic 0), the associated diode will conduct driving point *N* to ground *i.e.* 0*V*. Since there is no base voltage for *Q*, it will be cut *OFF* thereby driving point *C* and hence output to V_{CC} i.e. logic 1.

It is seen that output is low (a logic 0) only when all inputs are high — the condition for a *NAND* gate.

The *DTL* family is characterised by

1. relatively lower speed,
2. Comparatively better noise immunity,
3. propagation delay of 30 ns,
4. a fan-out of 5.

8-16. TTL Circuit

It is a *saturated* logic. It is the most widely used circuit line since early 1970*s* because of its speed, good fan-out and easy interface with other digital circuitry. The unique feature of this circuit is that it used *multiple-emitter transistor* at input which replaces the input diodes of the *DTL.* The number of emitters is equal to the number of inputs of the logic circuit (limited to 8). Since a multi-emitter transistor is small in area than the diodes it replaces, the yield from a wafer is increased. Moreover, smaller area resultes in *lower capacitance* to the substrate, thereby wide selection of circuit modules ranging from simple gates and flip-flops in *SSI* circuit series through various registers in computers in *MSI* circuit series to micro-processor bit-slice chips in the *LSI* series.

Basic Circuit

The basic circuit of the *TTL* family is the *NAND* gate cell shown in Fig. 8-16. However, at present *NOR*, *OR* and *AND* gate configurations have also been added to the series.

Circuit Operation

1. If both inputs *A* and *B* are HIGH (logic 1), *E/B* junction of Q_1 is *reverse-biased* so that it has no emitter current. Hence Q_1 is *OFF*. However, its *C/B* junction is *forward-baised* supplying base current to Q_2 from V_{CC} via R_1. As a result transistor Q_2 is turned fully *ON* (*i.e.* it becomes saturated) driving point *N* to 0*V*. Hence, output is a logic 0.

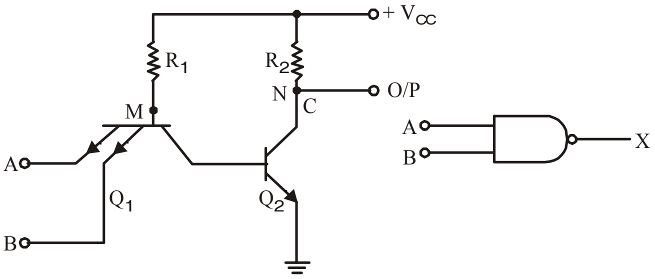


Fig. 8-16. TTL NAND Circuit

2. When either or both inputs are at 0 *V* (logic 0), the associated *E/B* junction becomes forward-biased. The value of R_1 is so selected as to ensure that Q_1 is turned fully *ON*. The voltage at point M falls to 0*V* with the result the base current for Q_2 is reduced to zero. Hence, Q_2 is cut *OFF* driving point *N* and the output to logic 1.

Totem Pole Output

The basic circuited of Fig. 8-16 is never used in practice. Its modified version with an added output stage is in common use (Fig. 8-17). This extra output stage is often known as *totem-pole* stage because the three components Q_3, Q_4 and *D* are stacked one on top of the other in the manner of a totem-pole. The circuit action is as follows :

1. When Input is HIGH

In this case, the two input terminals have *positive* voltage (logic 1). The *E/B* junction is reverse-biased because of which there is no emitter current. Hence, Q_1 is *OFF*. Since *C/B* junction of Q_1 is forward-biased, base current of Q_2 flows from V_{CC} through R_1. Hence, Q is turned *ON*. As a rusult, potential of point *N* falls so much that Q_2 is turned *OFF*. At the same time Q_4 is turned *ON* by the voltage drop across R_3. Now, when Q_4 is *ON*, its collector potential (*i.e.* potential of point *C*) is nearly that of its emitter. Hence, output is *LOW* i.e. at logic 0.

In short, when inputs are at logic 1, Q_1 is *OFF*, Q_2 is *ON*, Q_3 is *OFF* and Q_4 is *ON* because of which output is logic 0.

2. When Input is LOW

If any of the two inputs or both *LOW* (logic 0), Q_1 turns *ON* and potential of its collector (point M) falls. Hence, Q_2 is turned *OFF*, grounding its emitter and the base of Q_4 so that Q_4 is also turned OFF.

Since *N* is at V_{CC}, it turns Q_3 *ON*. The potential of point *C* is V_{CC}, minus drop in R_4, Q_3 and *D*. Since these drops do not amount to much, output is at logic 1.

It may be noted that when *even-numbered transistors are ON, the odd-numbered ones are OFF and vice-versa.*

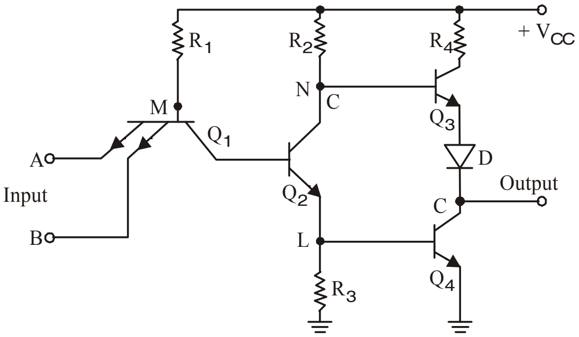


Fig. 8-17. TTL NAND Circuit with Totem pole output

The function of diode *D* in Fig. 8-17 is to prevent both Q_3 and Q_4 from being turned ON simultaneously. If both were to be *ON* at the same time, they would offer low immedance to the supply which will draw excessive current and produce large noise 'spikes' in the output. It may also be noted that the addition of a pair of totem pole transistor increases the operating speed and output current capability of this circuit. The standard *TTL*-family has

1. greater speed then *DTL*,
2. less noise immunity (0.4 *V*),
3. average propagation delay per gate of 9 *ns*,
4. average power dissipation of 10 *mW*,
5. a fan-out of 10 meaning one output can drive 10 other *TTL* inputs,

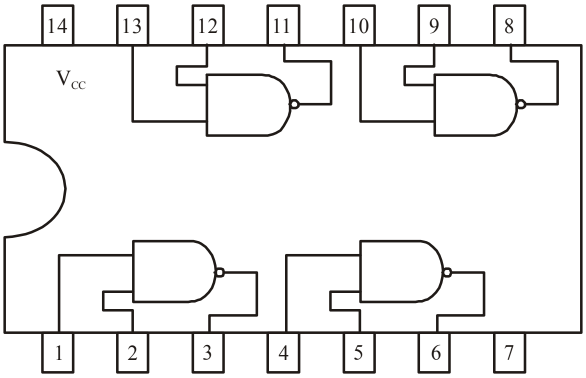


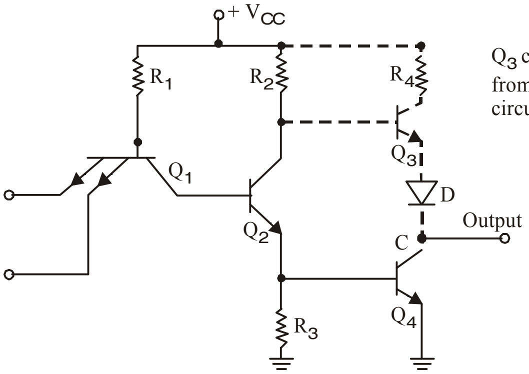
Fig. 8-18. Quad 2-input NAND gate

Fig. 8-18 shows a pin-out of a *BEL* 7400 *IC* referred to as 'quad (quadrupole). 2-input *NAND* gate chip'. As seen, there are four separate 2-input NAND, gates of which any or all may be used at any point in time. It is manufactured by Bharat Electronics LTD. (BEL) Bangalore, India.

8-17. TTL Circuit with Open-collector Output

Instead of using a totem-pole arrangement in the output stage of a *TTL* gate, we can use another arrangement called open-collector output. Recall that with the totem-pole output stage, for a LOW output, the lower transistor (*i.e.* Q_4 in Fig. 8-17) is *ON* and the upper transistor (*i.e.* Q_3 in Fig. 8-17) is *OFF*. On the other hand for *HIGH* output, lower transistor is *OFF* and the upper transistor is *ON*.

Fig. 8-19 shows open-collector arrangement for a TTL gate. Notice that the upper transistor is removed. Now the output will be *LOW* when Q_4 is *ON* and the output will float (*i.e. HIGH* or *LOW*) when Q_4 is *OFF*. This means that open-collector output can sink current, but it cannot source current.



Q_3 circuitry removed from original NAND gate circuit.

Fig. 8-19. TTL Circuit with open-collector output

In order to get an open-collector output to produce a *HIGH*, an external resistor (called a pull-up resistor) must be used as shown in Fig. 8-20. Now when Q_4 is *OFF*, the output is *HIGH* (i.e. + 5 V) and when Q_4 is *ON*, the output is *LOW* (i.e. 0 *V*). The optimum value of the pull-up resistor depends on the value of load current required at the output. A typical value of pull-up resistor is 10 $k\Omega$.

It will be interesting to know that open-collector buffer/driver ICs are available for output loads requiring large sink currents. The examples of such loads are LED display, relays and motors. The term buffer/driver means the device has an ability to provide high output currents to drive heavy loads. For example, IC 7406 and 7407 are open collector inverter buffers/drivers. These ICs can sink current up to 40 *mA* which is 10 times more than the normal (4 *mA*) capability of the standard 7404 inverter.

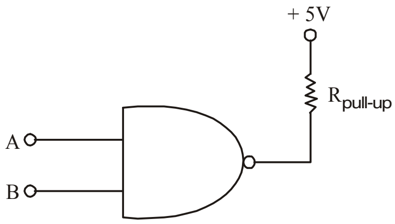


Fig. 8-20. Using pull-up resistor with a TTL gate with open-collector output

8-18. TTL Circuit with Three-State Output

TTL gates in all the available series come in three different types of output configuration:

1. Open-collector output
2. Totem-pole output
3. Three-state (or tristate) output

These three types of outputs will be considered in conjunction with the circuit description of the basic TTL gate. We have already discussed the Totem pole and Open-collector output in the previous article. We will know discuss the Three-state output configuration.

The outputs of two TTL gates with totem-pole structures cannont be connected together as in open-collector outputs. There is, however, a special type of totem-pole gate that allows the wired connection of outputs for the purpose of forming a common-bus system. When a number of gate outputs are connected to the bus, we encounter some difficulties. These are:

1. Totem pole outputs cannot be connected together because of very large current gain from the supply and consequent heating of the ICs which get damaged.
2. Open-Collector outputs can be connected together with a common collector-resistor connected externally. This causes the problems of loading and speed of operation.

To overcome these difficulties, special circuits have been developed in which there is one more state of the output, referred to as the third state or high-impedance state, in addition to the LOW and HIGH states. When a totem-pole output TTL gate has this property, it is called a three-state (or tristate) gate.

A three-state gate exhibits three output states:

1. Low level state: When the lower transistor in the totem-pole is on and the upper transistor is off,
2. High-level state: When the upper transistor in the totem-pole is on and the lower transistor is off,
3. Third state: When both transistors in the totem-pole are off.

The third state provides an open circuit or high-impedance state that allows a direct wire connection of many outputs to a common line. Three-state gates eliminate the need for open-collector gates in bus configurations. We have already study the tristate gate in chapter 3.

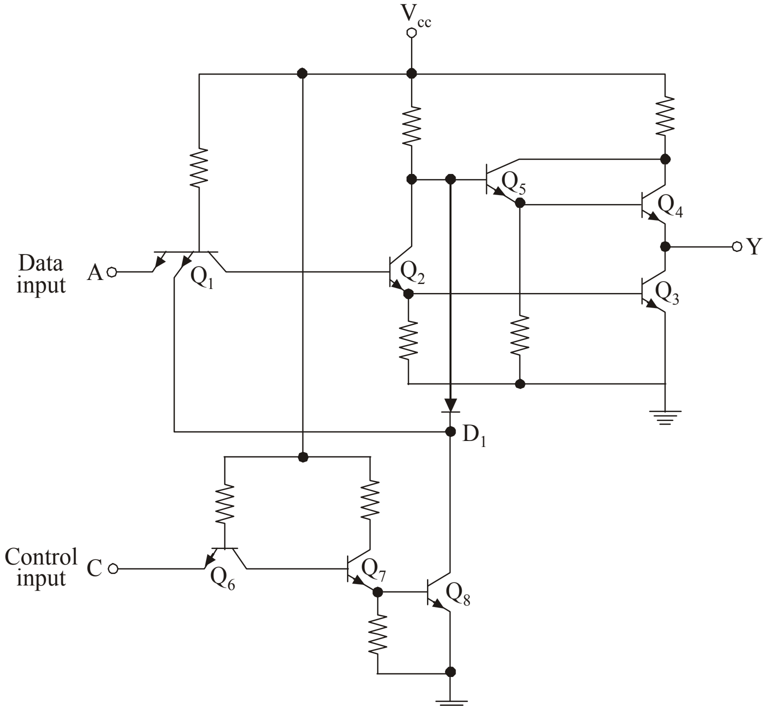


Fig. 8-21. Circuit Diagram for the three-state inverter

The circuit diagram of the ***three-state inverter*** in shown in Fig. 8-21. Transistors Q_6, Q_7, and Q_8 associated with the control input from a circuit similar to the open collector gate. Transistors Q_1, Q_2, Q_3, Q_4 and Q_5, associated with the data input, from a totem-pole TTL circuit. The two circuits are connected together through diode D_1. As in an open collector circuit, transistor Q_8 turns off when the control input is in the low-level state. This prevents diode D_1 from conducting, and also, the emitter in Q_1 connected to Q_8 has no conduction path. Under this condition, transistor Q_8 has no effect on the operation of the gate and the output in Y depends only on the data input at A.

When the control input is high, transistor Q_8 turns on, and the current flowing from V_{cc} through diode D_1 causes transistor Q_8 to saturate. The voltage at the base of Q_5 is now equal to the voltage across the saturated transistor, Q_8, plus one diode drop, or 0.9 V. This voltage turns off Q_5 and Q_4 since it is less than two V_{BE} drops. At the same time, the low input to one of the emitters of Q_1 forces transistor Q_3 (and Q_2) to turn off. Thus, both Q_3 and Q_4 in the totem-pole are turned off and the output of the circuit behaves like an open circuit with very high output impedance.

A three-state bus is created by wiring several three-state outputs together. At any given time, only one control input is enabled while all other outputs are in the high impedance state. The single gate not in a high-impedance state can transmit binary information through the common bus. Extreme care must be taken that all expect one of the outputs are in the third state; otherwise, we have the undesirable condition of having two active totem-pole outputs connected together.

An important feature of most three-state gates is that the output enable delay is longer than the output disable delay. If a control circuit enables one gate and disables another at the same time, the disabled gate enters the high-impedance state before the other gate is enabled. This eliminates the situation of both gates being active at the same time.

There is a very small leakage current associated with the high-impedance condition in a tree-state gate. Nevertheless, this current is so small that as many as 100 three state outputs can be connected together to form a common-bus line.

8-19. TTL Sub-families

TTL has several *sub-families* having different speed and lower dissipation characteristics as detailed below :

1. **74L00** series — the letter *L* standing for low power consumption. It has an average power dissipation of 1 *mW* per gate but an average propagation delay of 33 *ns.*
2. **74H00** series — the letter *H* standing for higher speed. It has a propagation delay of 6 ns but average power dissipation of 23 mW/gate.
3. **74S00** — the letter *S* representing Schottky. It has the highest speed because its average propagation delay is just 3 ns per gate. However, its average power dissipation is 23 mW/ gate.
4. **74LS00** — It is called low-power Schottky *TTL*. It has an average propagation delay of 9.5 ns and an average power dissipation of 2 mW.
5. **74AS00** series — The letter *A* representing Advanced and *S* standing for schottky. It is called advanced schottky *TTL* series. It is the fastest *TTL* series.
6. **74ALS00** series — It is called Advanced Low-power Schottky *TTL* series. The 74*ALS* series has the lower speed-power product and the lowest gate power dissipation of all the *TTL* series.
7. **74F00** series — The letter *F* standing for fast. This logic family uses a new *IC* fabrication technique to reduce interdevice capacitances to achieve reduced propagation delays. It has a propagation delay of 3 ns and a power consumption of 6 *mW.*

Table 8-1 summarizes the important characteristics of the each of the *TTL* sub-families.

Table 8-1.

Characteristic	*TTL sub-family*					
	74	74*S*	74*LS*	74*AS*	74*ALS*	74*F*
Propagation delay (ns)	9	3	9.5	1.7	4	3
Power Dissipation (mW)	10	20	2	8	1.2	6
Speed-power product (pJ)	90	60	19	13.6	4.8	18
Fan-out	10	20	20	40	20	33

8-20. ECL Circuit

The *ECL* also called current-mode logic (*CML*), has the **highest speed** of any of the currently-available logic circuits. It is primarily due to the fact that transistors never operate fully saturated or cut-off. That is why *ECL* is known as *non-saturated* logic. The latest *ECL* series has propagation delay time varying from 0.1 *ns* to 0.8 *ns*. However, power dissipation is increased since one transistor is always in the active region.

Another feature fo *ECL* is that it provides two outputs which are always complement of each other (Fig. 8-21). It is so because the circuit operation is based on a differential amplifier.

This family is particularly suited to monolithic fabrication techniques because logic levels are function of resistor ratios.

Circuit Operation

The basic circuit shown in Fig. 8-22 is combined *OR/NOR* circuit and is operated from a V_{EE} = – 5.2 *V* supply. A built-in constant-current source provides current to the emitters. Strictly speaking, logic 1 is represented by – 0.9 *V* (less negative) and logic 0 by – 1.75 *V* (more negative). Please note that it is a *positive* logic. In *negative* logic, the functions would be *AND/NAND*. A reference voltage of – 1.29 *V* is applied to the base of Q_3 from a built-in temperature-compensated reference voltage source (*RVS*).

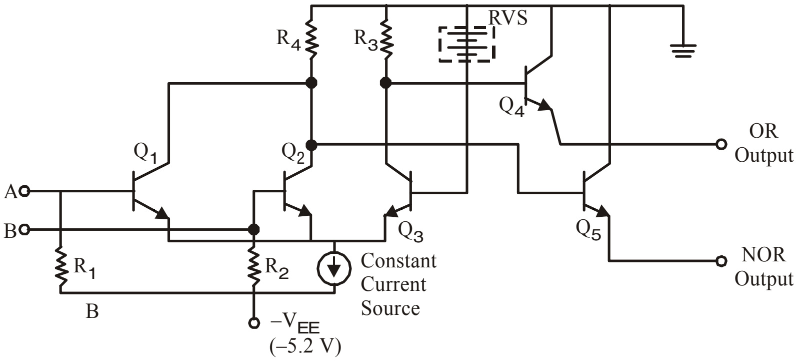


Fig. 8-22. ECL OR/NOR gate Circuit

1. When both inputs are logical 0 *i.e.* – 1.75 V

In this case, base potential of Q_3 is less negative (or more positive) than the base potential of either Q_1 or Q_2. Hence, Q_3 conducts whilst Q_1 and Q_2 do not. Only enough base current is drawn by Q_3 from RVS so as to remain out of saturation. The collector current of Q_3 develops a voltage of – 1.0 *V* across R_3 which makes Q_4 to conduct. Transistor Q_4 gives an output voltage at the emitter of about

– 1.0 – 0.7 = – 1.7 *V* which represents logic 0. Since collector potentials of Q_1 and Q_2 are nearly zero (because they are cut off), the output voltage at the emitter of Q_5 is 0 – 0.7 = – 0.7 *V* which is a logic 1. Obviously, the two outputs are complements of each other.

2. When either input A or B is at logical 1 i.e. – 0.9 V

In that case, the associated transistor (either Q_1 or Q_2) is turned *ON* while Q_3 is turned OFF. The collector potentials of Q_1/Q_2 and are opposite of the previous case. Hence, now Q_5 output is at logical 1 and Q_4 output is at logical 0.

Typical characteristics of an *ECL* family are :

1. propagation delay time per gate of 0.3 ns (meaning extremely fast speed),
2. power dissipation of 25 *mW*,
3. fan-out of 25 to 50,
4. noise margin from about 0.2 to 0.25 *V*.

It will be interesting to know that *ECL* family of *ICs* does not include a wide range of general purpose logic devices as do the *TTL* and *CMOS* logic families. *ECL* does include complex, special purpose *ICs* used in applications such as high-speed data transmission, high-speed memories and high-speed arithmetic units. The relatively low noise margins and high power drain of *ECL* are disadvantages compared with *TTL* and *CMOS.* Another drawback is its negative power supply voltage and logic levels which are not compatible with those of other logic families. This makes it difficult to use *ECL* devices in conjunction with *TTL* and/or *CMOS ICs*. Special level-shifting (also called interface) circuits must be connected between *ECL* devices and the *TTL* (or *CMOS*) devices on both input and output.

8-21. I^2L Circuit

It is latest entry into the bipolar *saturated* logic field. It uses no biasing and loading resistors at all! Resistors require lot of power and space on an *IC* chip. Hence, their elimination results in *higher density circuits operating at much reduce power.* Because of its **high speed** and **less power dissipation,** it is used in large computers. Such circuits are also used where high packing density is of prime consideration as in digital wrist watches. I^2L chips are capable of microwatt power dissipation yet can provide high currents when necessary to drive *LED* displays. Another feature of I^2L (integrated injection logic) is that it is easy to fabricate.

Circuit Operation

I^2L *NOR* logic circuit is shown in Fig. 8-23. Here, transistors Q_1 and Q_2 act as current sources to the bases of Q_3 and Q_4 respectively.

If *A* input goes *LOW*, the current to the base of Q_2 will be shorted to ground which will result in Q_2 being turned *OFF*. The input *B* controls Q_4 in a similar way.

If *A* is *HIGH*, base current flowing to Q_2 will turn in *ON*, making output *Z LOW*. Same would be the case when *B* is high.

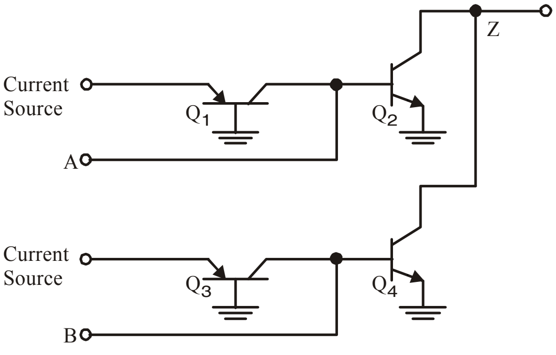


Fig. 8-23. I^2L NOR gate Circuit

It is obvious that output would be high only when *both inputs A* and *B* are *LOW.*

The output would be *LOW* when either *A* or *B* or both are *HIGH i.e. NOR* logic function.

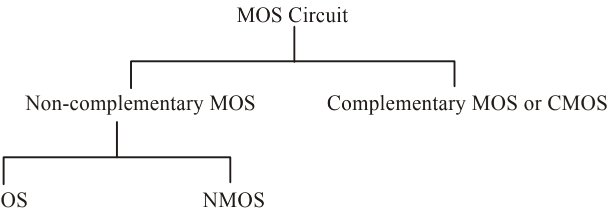
8-22. MOS Family

It does not use bipolar transistors but just Enhancement-only *MOSFETs*. There are two kinds of digital *MOS* circuits.

(*a*) one which used *MOSFETs* of one polarity either all of *N*-type called *NMOS* or all of *P-type* called *PMOS* but not both on the same chip,

(*b*) the other which employs both *N*-type and *P*-type *MOSFETs* on the same chip. It is called complementary *MOS* (*CMOS*).

The *MOS* family may be subdivided as under :



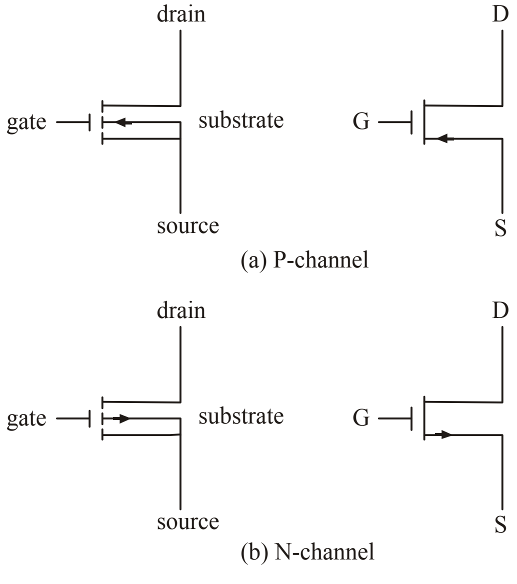
Since a *FET* requires small area, it is possible to fabricate a large number of *MOS* circuits on a single small chip. Gating arrays with thousands of gates and flip-flops are manufactured in standard containers and are often used in *IC* memories and microprocessors.

Following are same of the advantages of *MOS ICs* over the bipolar *ICs* (i.e. *TTL*, *ECL etc.*)

1. The *MOS IC* is relatively simple and inexpensive to fabricate.
2. The *MOS* device size is small and it consumers less power. Because of the small size, the *MOS ICs* can accommodate a much larger number of circuit elements on a single chip than bipolar *IC* in the area of large scale integration. This makes them especially well-suited for complex *ICs* such as microprocessor, memory chips etc.
3. *MOS* digital *ICs* normally do not use the *IC* resistor elements that take up so much of the chip area or bipolar *ICs*.

There are four basic types of MOS structures. The channel can be a P- or N-type, depending on whether the majority carriers are holes or electrons. The mode of operation can be enhancement or depletion, depending on the state of the channel region at zero gate voltage. If the channel is initially doped lightly with P-type impurity (diffused channel), a conducting channel exists at zero gate voltage and the device is said to operate in the *depletion mode*. In this mode, current flows unless the channel is depleted by an applied gate field. If the region beneath the gate is left initially uncharged, a channel must be induced by the gate field before current can flow. Thus, the channel current is enhanced by the gate voltage and such a device is said to operate in the *enhancement mode*.

The graphic symbols for the MOS transistors are shown in Fig. 8-24. The accepted symbol for the enhancement type is the one with the broken-line connection between source and drain. In this symbol, the substrate can be identified and is shown connected to the source. We will use an alternative symbol that omits the substrate; in this symbol, the arrow is placed in the source terminal to show the direction of *positive* current flow (from source to drain in the P-channel and from drain to source in the N-channel).



(a) P-channel

(b) N-channel

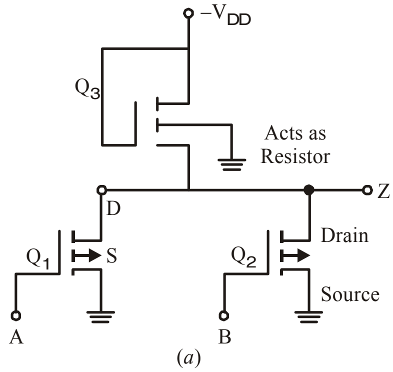
Fig. 8-24.

The continuous improvement in *MOS IC* technology has led to the device that are faster than *TTL* devices. Consequently *MOS* devices (especially *CMOS*) have become dominant in the *SSI* (small-scale integration) and *MSI* (medium-scale integration) market.

The major disadvantage of *MOS* devices is that they are susceptible to static-electricity damage. Although we can minimize it by adopting proper handling procedures, yet *TTL* is still more durable for laboratory experimentation. As a result, we are likely to see *TTL* devices used in education as long as they are available.

8-23. PMOS Circuit

A *PMOS NAND* gate is shown in Fig. 8-23 (*a*). As seen, there are no resistors in the circuit. The gate consists of two *E* only *MOSFETs* Q_1 and Q_2 as logic elements and the third one Q_3 as load resistors. When $-V_{DD}$ (say –12 *V*) is applied, *MOSFETs* will be turned *ON* and when 0 *V* is applied they would be turned *OFF.* Hence, with positive logic, 0 *V* would be 1 and –12 *V* would be 0 since 1 is assigned to the most positive voltage.



(*a*)

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

(*b*)

Fig. 8-25. PMOS NAND gate

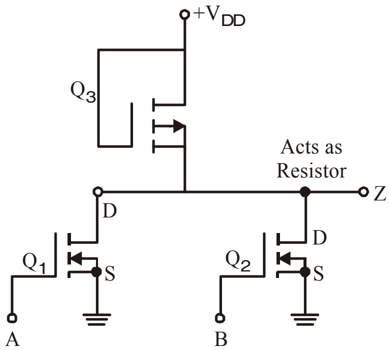
Circuit Operation

1. If any of the two inputs *A* or*B* is at logic 0 (*i.e.* – 12 *V*), the concerned *MOSFET* would turn *ON* thereby offering a low resistance from drain to source thus causing the output to be nearly 0*V i.e.* a logic 1.
2. The output can be at – 12 *V i.e.* logic 0 only when both inputs *A* and *B* are at 0 *V i.e.* logic 1. This is the *NAND* function as shown by the truth table of Fig. 8-25 (*b*).

Incidentally, the positive logic *NAND* gate of Fig. 8-25 (*a*) would be the *negative logic NOR* gate since the two are identical.

8-24. NMOS Circuit

In Fig. 8-26 (*a*) is shown a two-input *NOR* gate circuit consisting of two *MOSFETs* Q_1 and Q_2 acting as logic elements and Q_3 as a load resistor. For positive logic, 0 *V* will be logic 0 and positive voltage (+ V_{DD}) will be logic 1.



(*a*)

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

(*b*)

Fig. 8-26. NMOS NOR circuit

Circuit Operation

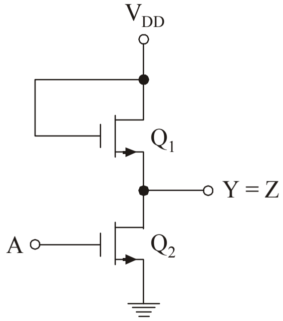
1. If any of the inputs *A* or *B* is at logic 1, the corresponding *MOSFET* will conduct causing the output to go *LOW i.e.* logic 0.
2. If both inputs *A* and *B* are at logic 0, then both *MOSFETs* will be *OFF* driving the output to logic 1.

This is *NOR* function as shown by the truth table of Fig. 8-26 (*b*).

For an N-channel MOS, supply voltage V_{DD} is positive (about 5 V) to allow positive current flow from drain to source. The two voltage levels are a function of the threshold voltage V_T. The low level is anywhere from zero to V_T, and the high level ranges from V_T to V_{DD}. The N-channel gates usually employ positive logic. The P-channel MOS circuits use a negative voltage for V_{DD} to allow positive current flow from source to drain.

8-25. NMOS INVERTER Circuit

The INVERTER circuit is shown in Fig. 8-27 uses two MOS devices. Q_1 acts as the load resistor and Q_2 as the active device. The load resistor MOS has its gate connected to V_{DD} , thus maintaining it always in the conduction state. When the input voltage is low (below V_T), Q_2 turns off. Since Q_1 is always on, the output voltage is at about V_{DD}. When the input voltage is high (above V_T), Q_2 turns on. Current flows from V_{DD} through the load resistor Q_1 and into Q_2. The geometry of the two MOS devices must be such that the resistance of Q_2, when conducting, is much less than the resistance of Q_1 to maintain the output Y at a voltage below V_T.

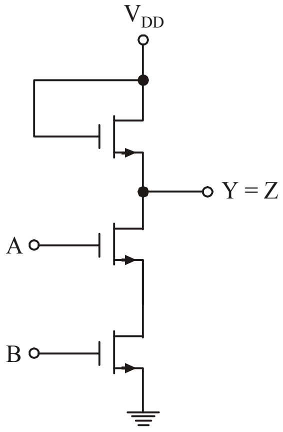


A	Z
0	1
1	0

Fig. 8-27. NMOS INVERTER Circuit

8-26. NMOS NAND Circuit

The NAND gate shown in Fig. 8-28 uses transistors in series. Inputs A and B must both be high for all transistors to conduct and cause the output to go low. If either input the corresponding transistor is turned off and the output is high. Again, the series resistance formed by the two active MOS devices must be much less than the resistance of the load-resistor MOS.



A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 8-28. NMOS NAND Circuit

Circuit Operation

1. If any of the inputs A or B is at logic 0, the corresponding MOSFET will be OFF and cause the output to go HIGH *i.e.* logic 1.
2. If both inputs A and B are at logic 1, then both MOSFETs will be ON driving the output to logic 0.

This is NAND function as shown by the truth table of Fig. 8-28.

8-27. CMOS Circuit

CMOS logic circuits use both *PMOS* and *NMOS* devices in the same circuit. It gives the advantage of *drastic decrease in power dissipation* (12 *nW* per gate) and *increase in speed of operation.* In fact, it has the **lowest power dissipation** amongest different logic families. It has very high packing density *i.e.* larger number of circuits can be placed on a single chip. As a result, it is extensively used in *VLSI* circuits such as on-chip computers and memory systems. The recent silicon-on-saphire *MOS* (*SOMOS*) is 2 to 4 times faster than the standard *CMOS*. Hence, they are being widely used for everything from electronic watches and calculators to mirco processors.

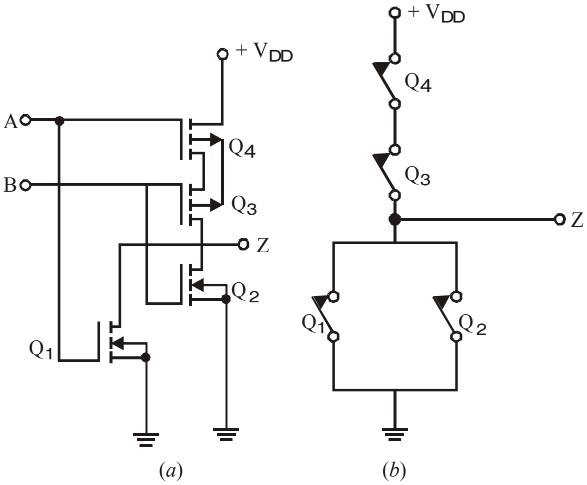


Fig. 8-29. CMOS NOR gate

Fig. 8-29 (*a*) shows a *CMOS NOR* circuit which has two *N*-channel *MOSFETs* Q_1 and Q_2 and two *P*-channel *MOSFETs* Q_3 and Q_4. The two inputs *A* and *B* switch between + V_{DD} (logic 1) and ground (logic 0).

Circuit Operation

1. Let $A = B$ = logic 1 *i.e.* have positive voltage. In that case, Q_1 and Q_2 are *ON* (closed switches) but Q_3 and Q_4 are *OFF* and act as open switches as shown in Fig. 8-29 (*b*). Hence, output *Z* is logic 0.
2. If either *A* or *B* is at logic 1, then the associated *N*-channel *MOSFET* (Q_1 or Q_2) is turned *ON* but the associated *P*-channel *MOSFET* (Q_3 or Q_4) is turned *OFF*. Since either Q_3 or Q_4 would be *OFF* [Fig. 8-29 (*b*)], output *Z* would be at logic 0.
3. When both *A* and *B* are at logic 0, Q_3 and Q_4 would be *ON* but Q_1 and Q_2 would be *OFF* — just the opposite of that in Fig. 8-29 (*b*). Hence output Z would be at logic 1.

The above logic represents a *NOR* function as shown in the truth table of Fig. 8-30. It would be observed from Fig. 8-29 that in each combination of *A* and *B*, there is at least *one open switch* between + V_{DD} and ground. Hence, the gate draws *only leakage current from supply for any static state.* However, when the gate switches from one level to another, some power is consumed because two *MOSFETs* are *partly ON* at the same time. Because of this reason, power dissipated by *CMOS* circuit *is a function of input signal frequency.* Higher the frequency, greater the power dissipation.

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 8-30.

8-28. CMOS INVERTER Circuit

The basic circuit is the inverter, which consists of one p-channel transistor and one n-channel transistor, as shown in Fig. 8-31. The source terminal of the p-channel device is at V_{DD}, and the source terminal of the N-channel device is at ground. The value of V_{DD} may be anywhere from + 3 to + 18 V. The two voltage levels are 0 V for the low level and V_{DD} for the high level.

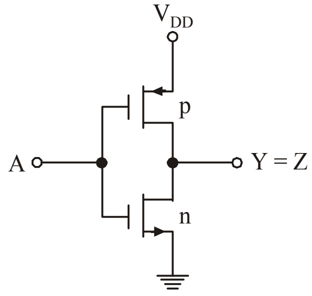


Fig. 8-31.

8-29. CMOS NAND Circuit

Two other CMOS basic gates are shown in Fig. 8-32. A two-input NAND gate consists of two P-type units in parallel and two N-type units in series. If all inputs are high, both P-channel transistors turn off and both N-channel transistors turn on. The output has low impedance to ground and produces a low state. If any input is low, the associated n-channel transistor is turned off and the associated p-channel transistor is turned on. The output is coupled to V_{DD} and goes to the high state. Multiple-input NAND gates may be formed by placing equal numbers of P-type and N-type transistors in parallel and series, respectively.

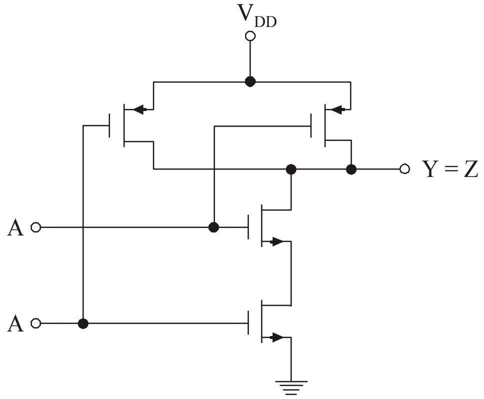


Fig. 8-32.

8-30. CMOS Sub-families and their Characteristics

The *CMOS* family of *ICs* competes directly with *TTL* in the small and medium-scale integration. As *CMOS* technology has produced better and better performance characteristics, it has gradually taken over the field that has been dominated by *TTL* for so long. As a matter of fact, *TTL* devices will be around for a long time, but more and more new equipment is using *CMOS* logic circuits. These days, the *CMOS ICs* provide all the logic-functions that are available in *TTL*. Some special-purpose functions provided in *CMOS ICs* are not provided even by *TTL*. Before we look at the various *CMOS* subfamilies, let us define few terms that are used when *ICs* from different families or series are to be used together or as replacements for one another.

(*a*) **Pin-to-pin Compatible.** Two *ICs* are said to be pin-to-pin compatible when three pin configuration are the same. For example, pin 14 on both *ICs* is V_{CC} supply. Pin 7 on both is ground etc.

(*b*) **Functionally Equivalent.** Two *ICs* are said to be functionally equivalent when the logic function they perform are exactly the same. For example, both contain four two-input *AND* gates, or two-input *NAND* gates etc.

(*c*) **Electrically compatible.** Two *ICs* are said to be electrically compatible when they can be connected directly to each other without taking any special measures to ensure proper operation.

Let us now study the different *CMOS* subfamilies.

1. **4000/14000 series.** The *CMOS* 4000 series or the improved 4000 *B* is the oldest series and was first introduced by *RCA*. This series is functionally equivalent to 14000 series from Motorola. The *CMOS* devices in 4000/14000 series have a very low power dissipation and can operate over a wide range of supply voltage (*i.e.* from 3 to 15 *V*). These devices are very slow as compared to *TTL* and other *CMOS* subfamilies. The 40H00 series was designed to be faster than 4000 series. It did overcome some of the speed limitations, but it is still much slower than *LSTTL* series . The 4000 series have very low output current capability. The devices in 4000 series are not pin-compatible or electrically compatible with any *TTL* series.
2. **74C00 series.** This is pin-compatible as well as functionally equivalent to *TTL* devices having the same device number. For example, 74C04 is a HEX inverter that has the same pin configuration as the *TTL* 7404 HEX inverter *IC*. The performance characteristics of 74C00 series are the same as those of 4000 series.
3. **74HC/HCT00 series.** The letter *H/HC* is standing for high-speed *CMOS* series. This series has a speed which is 10 times faster than of 74LS00 series devices. It has also a higher current capability than that of 74C00 series. The 74HC/HCT00 series *ICs* are pin-compatible with and functionally equivalent to *TTL ICs* with the same device number. This series has become the most widely used series.
4. **74AC/ACT00 series.** This series is referred as advanced *CMOS* logic. It is functionally equivalent to the *TTL* series but not pin-compatible.
5. **74AHC00.** This series is referred to as advanced high-speed *CMOS* logic. It is faster, and has lower-power dissipation. The devices in this series are 3 times faster and can be used as direct replacements for *HC* series devices.
6. **74-BiCMOS series.** These days, the *IC* manufacturers have developed a new logic series-called *BiCMOS* logic. This series combines the best features of bipolar and *CMOS* logic *i.e.* low power characteristics of *CMOS* and high speed characteristics of bipolar circuits. *BiCMOS ICs* are available only in those functions that are used in microprocessor interfacing and memory applications such as latches, buffers, drivers and transceivers.
7. **74-Low Voltage series.** A new series of logic using a nominal supply voltage of 3.3 *V* has been developed to meet the extremely low power design requirements of battery powered and hand held devices. These *ICs* are being designed into the circuits of notebook computers, mobile radios, hand-held video games, telecommunication equipment, personal digital assistant (*PDA*) and high performance work station computers.

   Table 8-2 shows some of the common Low-voltage families identified by the suffixes as indicated:

Table 8-2.

Suffixes	*Low-voltage series*
LV	Low-voltage *HCMOS*
LVC	Low-voltage *CMOS*
LVT	Low-voltage technology
ALVC	Advanced Low-voltage *CMOS*
HLL	High speed Low-power Low-voltage

The power consumption of *CMOS* logic *ICs* decreases approximately with the square of the supply voltage. On the other hand, the propagation delay increases slightly at this reduced voltage. However the speed is restored and even increased by using finer geometry and submicron (<1μm) *CMOS* technology that is tailored for low-power and low-voltage applications.

8. **74 AHC/AHCT series.** This is an enchanced version of 74 *HC/HCT*00 series. It provides superior speed and low power consumption. The 74*AHC* series devices have half the static power consumption, one-third the propagation delay, high output drive current, and can operate at V_{CC} of 3.3 to 5 *V*.

Table 8-3 shows a comparison of the important characteristics of the *CMOS* logic families. It is eveident from this table that the devices from 74*HC/AHCT* series has the lowest propagation delay and the smallest value of power dissipation.

Table 8-3.

S. No.	*Characteristic*	*CMOS logic family*			
		4000 *B*	**74 *HC/HCT***	**74 *BiCMOS***	**74 *AHC/AHCT***
1.	Propagation delay (ns)	50	8	2.9	3.7
2.	Power dissipation static (mW)	0.001	0.0025	0.0003 – 7.5	0.000009
3.	Speed-power product (pw-s)	105	15	0.00087 to 22	—

8-31. Interfacing ICs from different Logic Families

While designing digital systems using integrated circuits from different logic families, there is a strong need to interface the devices correctly. This is particularly true for connecting *TTL* devices to *CMOS* and vice versa. In such situations, we have to make sure that a *HIGH* logic level out of a *TTL* logic device looks like a *HIGH* to the input of a *CMOS* gate. The same holds true for the *LOW* logic levels. The driving logic gate must sink or source enough current to meet the input current requirements of the logic gate being driven.

8-32. Interfacing TTL to CMOS

Consider the interfacing of a standard *TTL* (*i.e.* 7400 series) logic gate to a 4000 *B* series *CMOS*. When a *TTL* gate is used to drive the *CMOS* gate, there is no problem for the *LOW* level output. It is because of the reason that *TTL* guarantees a maximum *LOW*-level output of 0.4 *V* whereas *CMOS* will accept any voltage up to 1.5 *V* as a *LOW*-level input. But for the *HIGH* level, the *TTL* may output as little as 2.4 *V* as a *HIGH*. However, the *CMOS* expects at least 3.5 *V* as a *HIGH* level input. Therefore 2.4 *V* is unacceptable because it falls within indeterminate region. This problem is solved by connecting a resistor between the *CMOS* input and V_{CC} as shown in Fig. 8-33.

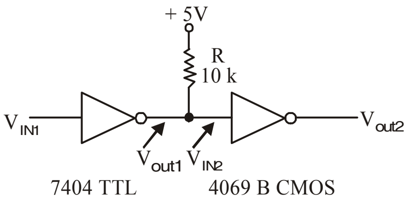


Fig. 8.33.

It may be noted when $V_{OUT\,1}$ is *LOW*, the 7404 inverter will sink current from 10 $k\Omega$ resistor and I_{IL} from the 4069B Inverter, making $V_{OUT\,2}$ logic *HIGH*. On the other hand when $V_{OUT\,1}$ is *HIGH*, the 10 $k\Omega$ resistor will pull the voltage at $V_{IN\,2}$ up to 5 *V*. As a result, $V_{OUT\,2}$ goes *LOW*.

The 10 $k\Omega$ resistor shown in Fig. 8-33 is called pull-up resistor. Its purpose is to raise the output of the *TTL* gate close to 5 *V* when it is in *HIGH* output state. It may be carefully noted that with $V_{OUT\,1}$ *HIGH*, the voltage at V_{IN2} will be nearly 5 *V* because input current drawn by 4069B is extremely *LOW* (approx. 1μ *A*), therefore the voltage drop across 10 $k\Omega$ resistor is negligible.

The other issue to look at when interfacing is the current levels of all logic devices that are involved. In this case, 7404 can sink (I_{OL}) 16 *mA* which is easy enough for the I_{IL} of the 4069*B* $(\approx 1\mu A)$ plus the current from the 10 $k\Omega$ resistor (whose value is 5 *V*/10 $k\Omega$ = 0.5 *mA*). Also notice that I_{OH} of the 7404 $(\approx -400\,\mu A)$ is also not a problem because with the pull-up resistor, 7404 will not have to source current.

8-33. Interfacing CMOS to TTL

Strictly speaking, when driving *TTL* from *CMOS*, the voltage levels are no problem. It is true because the *CMOS* will output about 4.9 *V* for *HIGH* and 0.1 *V* for a *LOW*. These are easily interpreted by the *TTL* gate. However, the real problem is with the current levels. The *CMOS* devices have severe output-current limitations. It may be carefully noted that the 74 *C* and 74 *HC* series have much better output current capabililty.

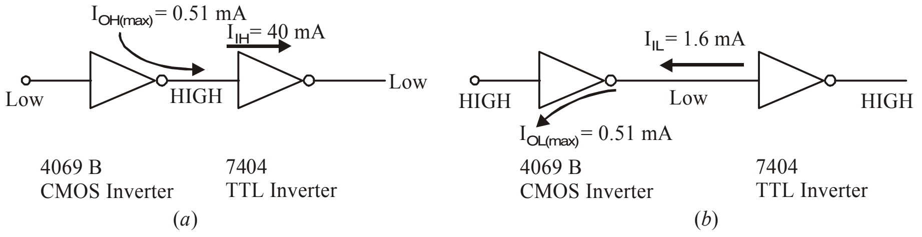


Fig. 8-34.

Fig. 8-34 shows the input/output currents that flow when interfacing *CMOS* to *TTL*. As seen from Fig. 8-34 (*a*), for the *HIGH* input condition, 4069*B CMOS* inverter can source a maximum current of 0.51 *mA* which is enough to supply the *HIGH*-level input current (I_{IH}) to one 7404 *TTL* inverter. But for the *LOW* output condition, as shown in Fig. 8-34 (*b*), the 4069 *B* can also sink only 0.51 *mA*. This is not enough for the 7404 *LOW*-level input current I_{IL} whose value is 1.6 *mA*.

Most of the 4000 *B CMOS* series has the same problem of *LOW*-output drive current capability. In order to solve this problem, special logic devices (called buffers) have been designed. The buffers can provide high output current when required. In the present situation 4050 buffer and 4049 (inverting) buffer are useful. These devices have drive capability of I_{OL} = 4.0 *mA* and I_{OH} = – 0.9 *mA*. This is enough to drive two 7400 series *TTL* gates.

It may be carefully noted that the interfacing problem discussed above was *CMOS* (4000 series) to standard *TTL* (C7400 series) gates. If you are considering the interfacing of *CMOS* (4000 series) to 74 *LS* series, you must refer to *TTL* data book to determine how many loads could be connected without exceeding the output current limit. As a matter of interest, 4050 *B* can actually drive 10 74 *LS* loads.

There is a variety of *CMOS* as well as *TTL* series devices. For interfacing, issues, we will strongly recommend you to refer to a data book to check the worst-case voltage and current parameters.

8-34. Level Shifting

There is another interesting problem that arises when we interface logic gates from the *IC* logic families that have different supply voltages. For example 4000 *B CMOS* series can use anywhere from + 3 *V* to + 15 *V* for a supply, and the *ECL* series uses – 5.2 *V* for a supply.

This type of problem is solved by using special logic devices called level-shifters. The 4049 *B* and 4050 *B* buffer *ICs* introduced earlier are also used for voltage-level shifting.

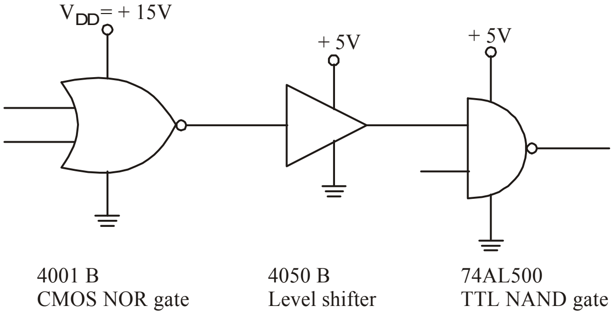


Fig. 8-35.

Fig. 8-35 shows the connections for interfacing a *CMOS* logic gate operating from a 15 *V* supply to a *TTL* gate operating at 5 *V* supply. Notice that the 4050 *B* level shifting buffer is operated from a 5 *V* supply. This device can accept 0 *V*/15 *V* logic levels at its input and output the corresponding 0 *V*/ 5 *V* logic levels at the output.

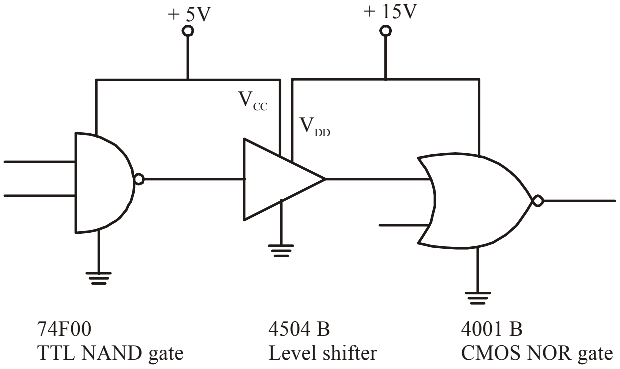


Fig. 8-36.

In order to interface 5 *V* to 15 *V CMOS*, the problem is solved by using 4504 *B CMOS* level shifter. This is shown in Fig. 8-36. As seen from this figure, 4504 *B* requires two power supply inputs. The 5 *V* supply is required to enable it to recognise the 0 *V*/5 *V* input levels and the 15 *V* supply to enable it to provide 0 *V*/15 *V* output logic levels.

8-35. Interfacing ECL to TTL/TTL to ECL

We have already mentioned in previous articles that *ECL* series devices uses – 5.2 *V* supply for their operation. Interfacing 0 *V*/5 *V* logic levels to – 5.2 *V*/0 *V ECL* circuitry requires another set of level shifter called translators. The 10125 is ECL-to-TTL translator and 10124 is a TTL-to-ECL translator.

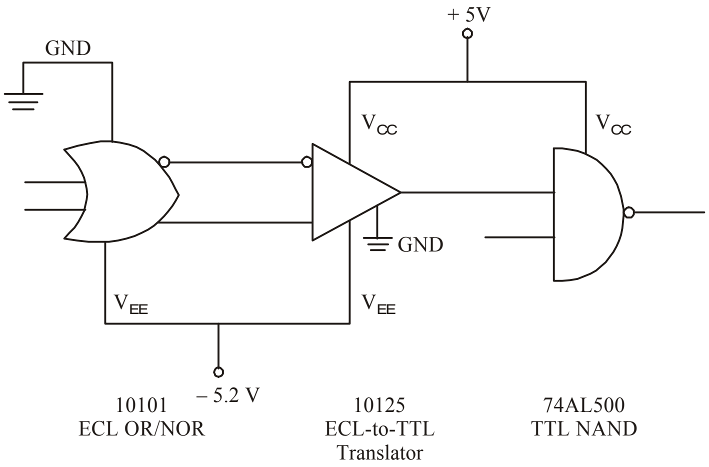


Fig. 8-37.

Fig. 8-37 shows the application of 10125 ECL-to-TTL translator for interfacing ECL 10101 logic device to 74*ALS*00 *TTL NAND* gate. The V_{EE} and V_{CC} supply connections between the *ECL* and *TTL* devices may be carefully noted.

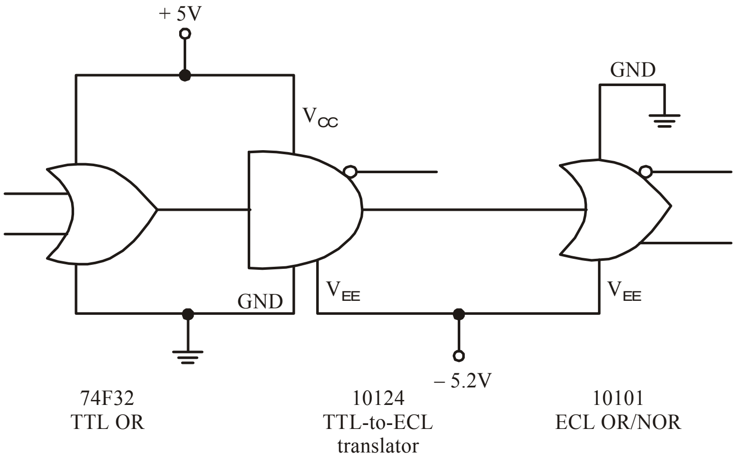


Fig. 8-38.

Fig. 8-38 shows the application of 10124 TTL-to-ECL translator for interfacing *TTL* 74*F*32 to *ECL* 10101. Again the V_{EE}, V_{CC} supply and ground connections between the three devices may be carefully noted.

SUMMARY

In this chapter, you have learned that :

1. There are several *IC* logic families but *TTL*, *CMOS*, *ECL*, I^2L and *Bi CMOS* are more popular these days.

2. In order to understand the basic operation of logic circuits in an logic family, you must understand the meaning of important operational characteristics. These include dc supply voltage, logic levels, noise immunity, noise margin, power dissipation, propagation delay, speed-power product and loading and fan-out.

3. TTL (transistor-transistor logic) is one of the most popular logic families. It has several sub-families like 74 *L*, 74 *S*, 74 *LS*, 74 *ALS*, 74 *F*.....
4. *ECL* – emitter coupled logic has the highest speed of any of the currently available logic circuits.
5. I^2L uses only the bipolar transistors. Because of its high speed and less power dissipation, it is used in large computers.
6. *CMOS* (Complementary metal-oxide semiconductor logic) has the lowest power dissipation among all the *IC* logic families and has the highest packing density.
7. Like *TTL, CMOS* also has several sub-families. These are 74 *C*, 74 *HC*, 74 *HCT*, 74 *AH*, 74 *BiCMOS*, and 74 *LV*.
8. Logic gates from different logic families cannot be connected directly. You must refer to the data book to check the worst-case voltages and current parameters of the devices being used in the circuit.

GLOSSARY

Bi CMOS. A logic family that is fabricated from a combination of bipolar transistor and complementary MOSFETs. It has low power dissipation and extremely high speed.

CMOS. Complementaly metal-oxide semiconductor (*CMOS*) uses both *NMOS* and *PMOS* devices in the same circuit. It has the lowest power dissipation among different logic families.

ECL. Emitter-coupled logic (*ECL*) has the highest speed of any of the currently available logic circuits. The circuit operation is based on a differential amplifier.

Fan-out. The number of logic gate inputs that can be driven from a single gate output of the same sub-family.

Interfacing. Connection between the logic devices from different logic families.

Noise-margin. The voltage difference between the guaranteed output voltage level and the required input voltage level of a logic gate.

Power dissipation. The electrical power consumed by a device (or a circuit) and dissipated (or given off) in the form of heat.

Propagation delay. The time required for a change in logic level to travel from the input to the output of a logic gate.

Sink current. The current entering the output or input of a logic gate.

Source current. The current leaving the output or input of a logic gate.

Totem-pole output. The output stage of the *TTL* family having two opposite-acting transistors, positioned one above the other.

DESCRIPTIVE QUESTIONS

1. Sketch the circuit of *DTL NAND* gate with three input terminals. Briefly explain the operation of the circuit. (*A.M.I.E., Summer 2002*)
2. Sketch the circuit of a typical integrated circuit *TTL* gate. Explain the function of each component. (*A.M.I.E., Summer 2001*)
3. Describe what is meant by the following terms :

   (*a*) fan-in (*b*) fan-out (*c*) sink current (*d*) source current
4. Describe how an open-collector *TTL* circuit is different from a normal *TTL* circuit.
5. What is the function of a pull-up resistor when interfacing a *TTL IC* to a *CMOS IC*. Describe with an example.

6. Draw a tristate inverter and draw its truth table.
(*Anna University, Nov./Dec. 2005*)

7. What are the advantage of the TTL family? Give numeric designation of TTL gates commonly used to perform NAND and NOR operation.
(*Guahati University, 2007*)

8. What are the properties of CMOS inverter.
(*Mahatma Gandhi University, Jan. 2007*)

9. What are the characteristics of TTL gates?
(*Mahatma Gandhi University, Jan. 2007*)

10. Explain an open collector gate. What its uses.
(*Mahatma Gandhi University, Jan. 2007*)

11. Compare the characteristics of ECL and IIL logic families.
(*Mahatma Gandhi University, Jan. 2007*)

12. Explain with figures the principle of operation of CMOS (*i*) NAND gate and (*ii*) NOR gate
(*Mahatma Gandhi University, Jan. 2007*)

13. What is meant by open collector output? What are its advantages?
(*Mahatma Gandhi University, Dec. 2007*)

14. Define and explain fan in and fan out with reference to TTL.
(*Mahatma Gandhi University, Dec. 2007*)

15. With a circuit diagram, explain the schottky clamped TTL NAND gate? What are its advantages?
(*Mahatma Gandhi University, Dec. 2007*)

16. Explain the circuit for interfacing TTL and CMOS devices.
(*Mahatma Gandhi University, Dec. 2007*)

17. Explain noise margin and figure of merit of a logic family.
(*Mahatma Gandhi University, Dec. 2007*)

18. What is noise margin? Explain with reference to TTL, giving standard values.
(*Mahatma Gandhi University, May/Jun. 2006*)

19. Define Fan-out: Explain with reference to TTL family, giving standard values.
(*Mahatma Gandhi University, Nov. 2005*)

20. Explain the principle of tri-state gates, giving their applications.
(*Mahatma Gandhi University, Nov. 2005*)

21. With a diagram explain the operation of a CMOS NAND gate. What are the advantages and disadvantages of CMOS logic?
(*Mahatma Gandhi University, Nov. 2005*)

22. With the help of a circuit diagram, explain the working of a 3-input NAND gate using TTL family. Mention the different members of TTL family and compare their typical power dissipation per gate and propagation delay.
(*Mahatma Gandhi University, Nov. 2005*)

23. Draw the circuit of a two input TTL NAND gate and verify the truth table.
(*Mahatma Gandhi University, May 2005*)

24. Explain why the power consumption of a CMOS circuit increases with the operating frequency.
(*Mahatma Gandhi University, May 2005*)

25. Explain the circuit of a CMOS inverter to verify the truth table.
(*Mahatma Gandhi University, May 2005*)

26. Describe a circuit of ECL gate.

(*Mahatma Gandhi University, May 2005*)

27. With the aid of a neat circuit diagram explain the operation of a 2-input TTL NAND gate with totem output.

(*VTU, Jan./Feb. 2004*)

28. Discuss how a resistor could be constructed using MOSFET. Give the resistor characteristics.

(*VTU, Jan./Feb. 2004*)

29. Draw the NMOS as well as PMOS circuit diagrams to realise a NAND gate. Give the relevant truth tables.

(*VTU, Jan./Feb. 2004*)

30. Explain the following properties of integrated circuits of the SSI type small scale integration type

(*i*) Propagation Delay (*ii*) Noise Margin

(*iii*) Fan-in (*iv*) Fan-out

(*VTU, Jan./Feb. 2005*)

31. Compare TTL and CMOS gates, in respect of these properties.

(*VTU, Jan./Feb. 2005*)

32. With a circuit diagram explain the operation of the CMOS NAND gate. What are its advantages over corresponding TTL gates?

(*VTU, Jan./Feb. 2005*)

33. Define fan-in and fan-out.

(*VTU, Jul./Aug. 2005*)

34. Explain a two input NAND gate TTL with totem pole output with a neat circuit diagram.

(*VTU, Jul./Aug. 2005*)

35. Explain the operation of a two input TTL NAND gate with totem-pole output with a neat circuit diagram.

(*VTU, Jan./Feb. 2006*)

36. Explain with the help of a circuit diagram the operation of a two input CMOS NOR gate.

(*VTU, Jan./Feb. 2006*)

37. Define the following terms

(*i*) Fan-in and Fan-out (*ii*) The propagation delay

(*VTU, Jul. 2006*)

38. What is the principle of operation of an schottky TTL? Explain with a circuit diagram the operation of a schottky TTL.

(*VTU, Jul. 2006*)

39. Define

(*i*) Propogation Delay (*ii*) Power-delay product

(*iii*) Fan-out (*iv*) Noise Margin

(*VTU, Dec./Jan. 2008*)

40. Write the circuit diagram of a TTL NAND gate draw and explain the transfer characteristic.

(*VTU, Dec./Jan. 2008*)

41. Explain with respect to TTL the following output stages

(*i*) Totem pole (*ii*) Open collector

(*iii*) Tri-state output

(*VTU, Dec./Jan. 2008*)

42. Describe a CMOS inverter with relevant circuit diagram.

(*VTU, Dec./Jan. 2008*)

43. Compare performance of ECL with TTL.

(*PTU, May 2007*)

44. Draw the schematic of ECL OR gate explain its operation.

(*PTU, May 2007*)

45. Give an order of magnitude which is applicable to various logic families for

(*i*) Fan out (*ii*) Power Dissipation per gate

(*iii*) Propagation delay per gate (*iv*) Clock rate

(*PTU, May 2008*)

46. Draw the circuit of TTL NAND gate and its operation. Compare the TTL and EFL logic families.

(*PTU, May 2009*)

47. What limit the speed of TTL gate? How can the speed of TTL gate be enhanced?

(*PTU, Dec. 2009*)

48. Draw a TTL gate that gives an output (AB) and explain its operation.

(*Anna University, Nov./Dec. 2005*)

49. Write down Fan-in and Fan-out of a standard TTL IC.

(*Anna University, May/Jun. 2006*)

50. Draw a CMOS NAND gate and explain its operation. What are the characteristics of CMOS?

(*Anna University, May/Jun. 2006*)

51. Draw a tristate TTL gate and explain its operation.

(*Anna University, May/Jun. 2006*)

52. What is propagation delay of a gate?

(*Anna University, Nov./Dec. 2006*)

53. Enumerate the precautionary measure to be considered while handling CMOS device.

(*Anna University, Nov./Dec. 2006*)

54. Compare the performance characteristics of TTL and CMOS.

(*Anna University, Nov./Dec.* 2006, *May /Jun. 2006*)

55. Explain the precautionary measure to be considered while handling CMOS device.

(*Anna University, May /Jun. 2006*)

56. Explain the operation of 3 I/P TTL NAND Gate with required diagram and truth table.

(*Anna University, Apr./May 2008*)

57. Explain the operation of CMOS NAND and NOR gates with the circuit and truth table.

(*Anna University, Apr./May 2008*)

58. Show the working of an RTL-NOR gate. Draw the circuit diagram

(*Guahati University, 2007, PTU, Dec. 2009*)

59. Why totem pole outputs cannot be connected together?

60. Explain

(*i*) Fan-in (*ii*) Fan-out

(*iii*) Noise Margin (*iv*) Propagation delay

(*v*) Sourcing current (*vi*) Sinking current of a standard TTL NAND gate

(*Mahatma Gandhi University, May* 2005; *Nagpur University, 2008*)

61. What is meant by open collector output of TTL gate ? What is its utility ? Draw the circuit showing open collector output and pull-up resistor. Explain its operation. What are the applications of open collector output?

(*GBTU/MTU, 2006-07*)

62. What are the advantages of BiCMOS logic circuit ? Draw the circuits of BiCMOS NAND and NOR gates. Explain their operations. Draw a circuit diagram to interface CMOS to TTL.

(*GBTU/MTU, 2006-07*)

63. Compare the various logic families qualitatively. If fast speed and FAN IN and OUT are the main design concern. Which logic family would you recommend?

(*Jamia Millia Islamia University, 2007*)

64. Give the characteristics of TTL logic family.

(*Nagpur University, 2004*)

65. Draw and explain the circuit of CMOS inverter.

(*Nagpur University, 2004*)

66. Define Fan-in and Fan-out.

(*Gujarat Technological University, Dec. 2009*)

67. Draw and explain the basic CMOS inverter circuits.

(*Gujarat Technological University, Dec. 2009*)

68. Compare the CMOS and TTL logic.

(*Gujarat Technological University, Dec. 2009*)

69. Write merits/demerits to different logic families.

(*GBTU/MTU, 2006-07*)

70. Explain the parameters used to characterize logic families.

(*GBTU/MTU, 2006-07*)

71. Explain ECL logic. Why is it faster than TTL logic ?

(*RGTU., June 2010, 2011*)

72. Explain the working of CMOS logic. Compare its speed with PMOS and NMOS. Give reasons for your answer.

(*RGTU., June 2011*)

73. Compare in detail RTL, DTL, TTL, ECL, IIL.

(*RGTU., Dec. 2010, June 2011*)

74. Discuss interfacing between TTL and MOS.

(*RGTU., June 2011*)

75. Describe PMOS, NMOS and CMOS logic.

(*RGTU., Dec. 2010*)

76. With the help of basic circuit of TTL NAND gate explain tristate output.

(*RGTU., June 2009*)

77. Explain the principle of IIL logic family.

(*RGTU., June 2009*)

78. Explain the working of NAND gate using N-channel MOS logic circuit.

(*RGTU., June 2009*)

79. Draw and explain the circuit diagram of a 3-input I^2L NOR gate.

(*RGTU., Dec. 2009*)

80. Write a note on interfacing (MOS with TTL).

(*RGTU., Dec. 2009*)

81. Draw the basic Bi-CMOS inverter and explain its operation.

(*RGTU., Dec. 2009*)

82. Write notes on the following :

(*i*) noise immunity
(*ii*) fan-out
(*iii*) fan-in
(*iv*) transition time
(*v*) propagation delay

(*RGTU., Dec. 2009*)

83. Discuss the major difference between a bipolar integrated circuit and a MOS integrated circuit.

(*RGTU., June 2010*)

84. What are BiMOS logic circuits ? What are their advantages ?

(*RGTU., June 2010*)

TUTORIAL PROBLEMS

1. The total supply current for *IC* 7402 *NOR* gate is given as I_{CCL} = 14 *mA* and I_{CCH} = 8 *mA*. Determine the power dissipation for the *IC*. (**Ans.** 55 mW)

MULTIPLE CHOISE QUESTIONS

1. In a saturated bipolar logic circuit, transistor operate.

(*a*) in deep cut-off (*b*) over active region
(*c*) in saturation (*d*) just short of saturation

2. Saturated logic circuits have inherently.

(*a*) short saturation delay time (*b*) low switching speed
(*c*) higher power dissipation(*d*) lower noise immunity

3. Noise margin is expressed in

(*a*) decibel (*b*) watt (*c*) volt (*d*) none of these

4. DTL family employs

(*a*) resistors and transistors (*b*) diode and resistor
(*c*) diode and transistor (*d*) diode, resistors and transistors

5. The circuit shown in Fig. 8-39, is a

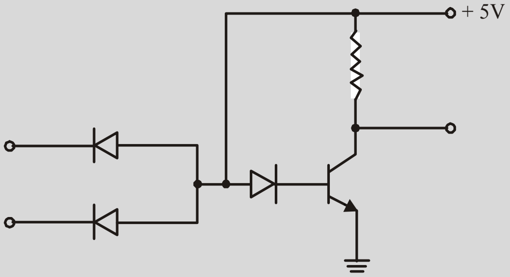


Fig. 8-39.

(*a*) DTL NOR gate (*b*) TTL NOR gate
(*c*) RTL NAND gate (*d*) DTL NAND gate

(*Grad. I.E.T.E., Dec.* 1996)

6. The main advantage of Schottly TTL logic family is its least

(*a*) power dissipation (*b*) propagation delay
(*c*) fan-in (*d*) noise-immunity

7. Which of the following logic family has the highest speed

(*a*) DTL (*b*) TTL (*c*) ECL (*d*) RTL

(*Dip. I.E.T.E., Dec.* 2000)

8. The logic family which produces highest speed is

(*a*) TTL (*b*) ECL (*c*) CMOS (*d*) DTL

(*A.M.I.E., Summer* 1999)

9. The fastest logic family is,

(*a*) TTL (*b*) ECL (*c*) IIL (or I^2L) (*d*) RTL

(*A.M.I.E., Summer* 2000)

10. The main advantage claimed for ECL family is its

(*a*) very large fan-in (*b*) use of negative power supply voltage
(*c*) extremely low propagation time (*d*) least power dissipation

11. Special feature of I^2L circuit is that it

(*a*) uses only high value resistors (*b*) dissipates negligible power
(*c*) is a bipolar saturated logic (*d*) is easy to fabricate
(*e*) uses no biasing and loading resistors

12. A unique feature of CMOS logic family is its,

(*a*) use of NMOS circuits (*b*) power dissipation in nand watt range
(*c*) speed (*d*) dependence on frequency for power dissipation

13. CMOS logic circuits are extensively used for on-chip computers mainly because of their extremely

(*a*) low-power dissipation (*b*) large packing density
(*c*) high-noise immunity (*d*) low cost

14. The main advantage of CMOS logic family over the TTL family is its

(*a*) much reduced power (*b*) increased speed of operation
(*c*) extremely low cost (*d*) series base resistor

15. CMOS logic family uses only

(*a*) MOSFET and resistors (*b*) MOSFETs
(*c*) bipolar transistors (*d*) NMOS devices

16. Power is drawn by a CMOS circuit only when

(*a*) its output is HIGH (*b*) its output is LOW
(*c*) it switches logic level (*d*) in static state

17. The most obvious identifying feature of a TTL gate is its

(*a*) large fan-out (*b*) high-power dissipation
(*c*) interconnected transistors (*d*) multiemitter input transistor

18. In digital circuits, schottky transistors are preferred over normal transistor because of their

(*a*) lower propagation delay (*b*) higher propagation delay

(*c*) lower power dissipation (*d*) higher power dissipation.

19. A unique operating feature of ECL circuit is its

(*a*) very high speed (*b*) high power dissipation

(*c*) series base resistor (*d*) compatibility with other logic families.

20. The fan-in of a logic gate refers to the number of

(*a*) input devices that can be connected (*b*) input terminals

(*c*) output terminals (*d*) circuits that can be connected at the output

21. Which of the following statements regarding ICs is not correct

(*a*) ECL has the least propagation delay (*b*) TTL has the largest fan-out

(*c*) CMOS has the biggest noise-margin (*d*) TTL has the largest fan-out

22. In a CMOS logic family, the power consumption will be lowest under the operating condition:

(*a*) $V_{DD} = 5\text{ V}, f_{max} = 1\text{ MHz}$ (*b*) $V_{DD} = 5\text{ V}, f_{max} = 10\text{ k Hz}$

(*c*) $V_{DD} = 10\text{ V}, f_{max} = 10\text{ kHz}$ (*d*) $V_{DD} = 10\text{ V}, f_{max} = 1\text{ MHz}$

(*A.M.I.E., Summer* 1999)

23. The propagation delay time per gate is the least for the following:

(*a*) 74 S (*b*) 74 LS (*c*) 74 L (*d*) 74 H

(*Dip. I.E.T.E. Dec.* 1996)

ANSWERS

1. (*c*)	**2.** (*b*)	**3.** (*c*)	**4.** (*d*)	**5.** (*d*)	**6.** (*b*)
7. (*c*)	**8.** (*b*)	**9.** (*b*)	**10.** (*c*)	**11.** (*c*)	**12.** (*d*)
13. (*b*)	**14.** (*a*)	**15.** (*b*)	**16.** (*c*)	**17.** (*a*)	**18.** (*a*)
19. (*a*)	**20.** (*a*)	**21.** (*d*)	**22.** (*b*)	**23.** (*a*)	

9

MEDIUM-SCALE INTEGRATED LOGIC CIRCUITS

OUTLINE

Objectives

Upon completion of this chapter, you will be able to :

- List the different types of MSI logic circuits
- Know the block diagram and operation of
  — decoder
  — encoder
  — multiplexer
  — demultiplexer and
  — magnitude comparator
- Application of MSI logic circuits to specific areas in the field of digital electronics.

9-1. Introduction

In the field of electronics, the digital systems have to handle/process data that comes in several different formats. The mechanism for conversion, transfer and selection of data is handled by medium scale integrated (MSI) combinational logic circuits.

The MSI logic circuits are those which have 12 to 99 logic gates fabricated on a single silicon chip. These include decoders, encoders, multiplexers, demultiplexers, and magnitude comparators. All these devices are discussed one by one in the following pages.

9-2. Combinational Circuit

A combinational circuit consists of input variables, logic gates and output variables. The logic to gates accepts signals from inputs and outputs signals are generated according to the logic circuits employed in it. The given data information is transformed to desired output data. Both the input and output are the binary signals, i.e. there are only two possible states logic 1 and logic 0.

Fig. 9-1, shows the block diagram of combinational logic circuit. These are *n* number of input variable and *m* number of output variables.

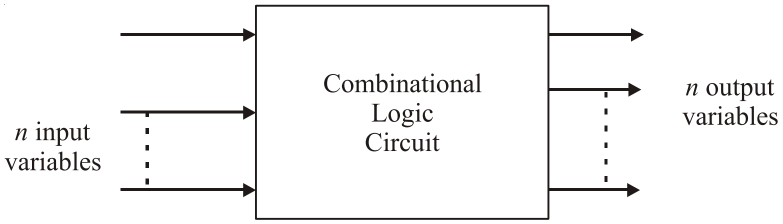


Fig. 9-1.

There are 2^n possible combinations of binary input states are possible for *n* number of input variables. For each possible combination there is one and only one possible output combination. Some of the important combinational circuit are given below;

1. Adder
2. Subtractors
3. Comparators
4. Decoders
5. Encoders
6. Multiplexers
7. Demultiplexers

Adders, subtractors and comparator combinational circuits are already discuss in chapter 6. Other remaing combinational circuits are discuss one by one.

9-3. Design Procedure of Combinational Circuit

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

1. Determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table.
3. Find the relation between inputs and outputs with the help of truth table.
4. Obtain the simplified Boolean functions for each output as a function of the input variables.
5. Draw the logic diagram.
6. Verify the correctness of the design.

The truth table of a combinational circuit consists of input and output columns. The input columns are obtained from the 2^n binary numbers for the n input variables. The binary values of the outputs are determined from the stated specifications. The output function of the truth table give the exact definition of the combinational circuit. The output function of the truth table are simplified by any variable method, like, K-map, algebraic manipulation etc.

A practical design method would have to consider

1. Minimum number of gates
2. Minimum number of inputs to a gate
3. Minimum propagation time of the signal through the circuit
4. Minimum number of interconnections
5. Limitations of the driving capabilities of each gate.

Since all these criteria cannot be satisfied simultaneously and since the importance of each constraint is dictated by a particular application.

The logic circuit for digital systems may be combinational circuit. The combinational circuit consists of logic gates whose outputs at any time are determined by combining the values of applied inputs using logic operations. A combinational circuit performs an operation that can be specified logically by a set of Boolean expression.

9-4. Analysis Procedure of Combinational Circuit

The analysis procedure for a combinational circuit consists of determining the function that the circuit implements. We analysis the given logic circuit diagram, set of Boolean functions or a truth table, together with a possible explanation of the operation of the circuit.

If the logic diagram to be analyzed is accompanied by a function name or an explanation of what it is assumed to accomplish, then the analysis problem reduces to a verification of the stated function. The analysis can be also performed manually by finding the Boolean function or truth table.

In analysis procedure we have to sure that the given circuit is combinational and not sequential circuit. The combinational circuit has logic gates with no feedback paths or memory elements. A feedback path is a connection from the output of one gate to the input of a second gate that forms part of the input to the first gate. The digital circuit with a feedback path is a sequential circuit and must be analyzed according to the other procedures.

After verified that the given circuit is the combinational circuit, we can proceed to obtain the output Boolean functions or the truth table. To obtain the output Boolean functions from a logic diagram, we proceed as follows:

1. First label all the gate outputs that are a function of input variables with arbitrary symbols. Find the Boolean function for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean function for these gates.
3. Process 2 is repeated until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtained the output Boolean functions in terms of input variables.

Late take an example to analysis the combinational circuits. We note that the circuit has three binary input A, B, and C and a binary output F_1. The circuit is shown in Fig. 9-2.

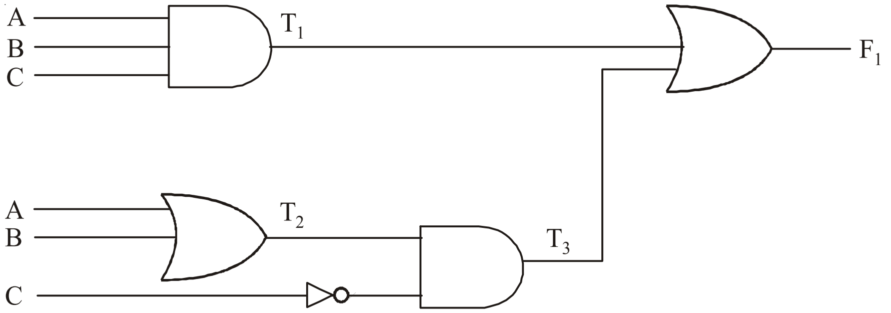


Fig. 9-2.

The output of various gates are labeled with intermediate symbols. The output of the gates that are a function only of input variables are T_1 and T_2. T_3 is the function of T and input variables C. The Boolean functions for T_1, T_2 and T_3 are

$$T_1 = ABC$$
$$T_2 = A + B$$
$$T_3 = (A + B)\,\overline{C} = A\overline{C} + B\overline{C}$$

The output function F_1 is,

$$F_1 = T_1 + T_3$$
$$= ABC + A\overline{C} + B\overline{C}$$

Converting in func of product

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

To obtain the truth table directly from the logic diagram without going through the derivations of the Boolean function, we can follow the following steps:

1. Number of variables are determine from the circuit. For n inputs, from the 2^n possible input combinations and list the binary numbers from 0 to 2^n–1 in a table.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates which are a function of the input variables only.
4. Proceed the obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

The truth table for the circuit is shown in Fig. 9-3. The truth table for T_1 are the AND function of variable A, B and C and for T_2 it is OR function of variable A and B. The values for T_3 is derived from T_2 and variable C. Finally the function F_1 is the OR function of T_1 and T_3.

A	B	C	T_1	T_2	T_3	F_1
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	0	1	1	1
1	1	1	1	1	0	1

Fig. 9-3.

Finding the sum of product from the truth table we get,

$$F = \overline{A}B\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$$

Above is the function for implement the combinational circuit. Another way of analyzing a combinational circuits is by means of logic simulation.

9-5. Decoders

Strictly speaking, a *decoder* is a logic circuit that accepts a set of inputs which represents a binary number and activates only the output that corresponds to the input number. In other words, a decoder circuit looks at its inputs, determines which binary number is present there, and activates the one output that corresponds to that number— all other outputs remain inactive.

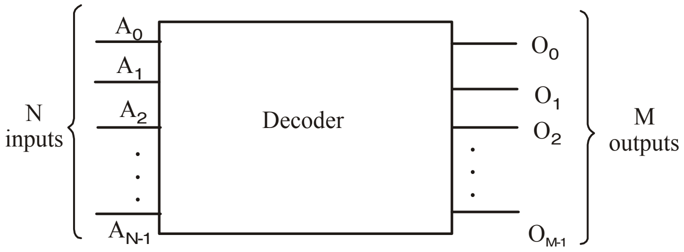
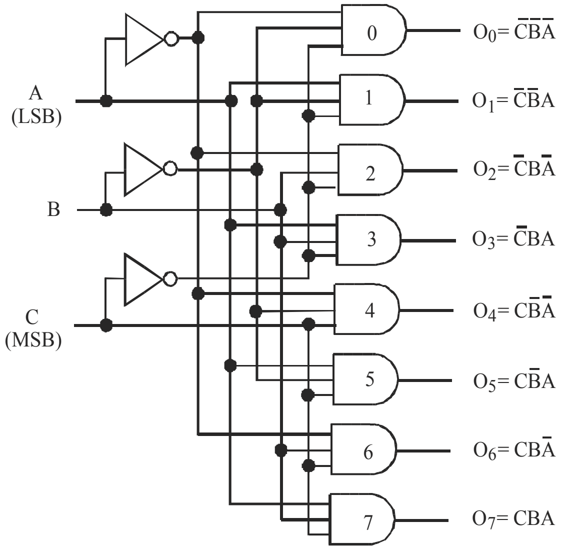


Fig. 9-4. A general decoder diagram

Fig. 9-4 shows a general decoder diagram. As seen from the diagram, it has N inputs and M outputs. Since each of the N inputs can be 0 or 1, there are 2^N possible input combinations or codes. For each of these input combinations only one of the M outputs will be active (HIGH) and all the other outputs are inactive (LOW). Many decoders are designed to produce *active-LOW* outputs where only the selected output is LOW while all others are HIGH. This would be indicated by the presence of small circles on the output lines in the decoder diagram.

Fig. 9-5 shows the logic diagram for a decoder with 3 inputs and 8 (= 2^3) outputs. It uses all AND gates and so the outputs are active-HIGH. It may be noted that for a given input code, the only output that is active (HIGH) is the one corresponding to the decimal equivalent of the binary input code. For example, output O_0 goes HIGH only when CBA = 000_2 = 0_{10}. Similarly O_1 goes HIGH only when CBA = 001_2 = $1_{10}$and O_5 goes HIGH when CBA = 101_2 = 5_{10} and so on. The complete operation of the decoder is given in the truth table shown in Fig. 9-5 (*b*).


A (LSB)
B
C (MSB)
0
1
2
3
4
5
6
7
$O_0 = \overline{C}\overline{B}\overline{A}$
$O_1 = \overline{C}\overline{B}A$
$O_2 = \overline{C}B\overline{A}$
$O_3 = \overline{C}BA$
$O_4 = C\overline{B}\overline{A}$
$O_5 = C\overline{B}A$
$O_6 = CB\overline{A}$
$O_7 = CBA$

(*a*) *Logic diagram*

Inputs			*Outputs*							
C	*B*	*A*	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

(*b*) Truth table

Fig. 9-5. Logic circuitry of a 3-line-to-8-line (or 1-of-8) decoder

This decoder can be referred to in several ways. It can be called a 3-line-to-8-line decoder because it has 3 input lines and 8 output lines. It could also be called a *binary-to-octal decoder* or *converter* because it takes a 3-bit binary input code and activates one of the eight (octal) outputs corresponding to that code. It is also referred to as a *1-of-8 decoder* because only 1 of the 8 outputs is activated at one time.

Table 9-1 lists some of the popular TTL/HCMOS decoder ICs. As seen fromthis table, 74138 is a 3-line-to-8 line (or 1-of-8-line) decoder, 7442 is a BCD-to-Decimal (or 4-line-to-10-line) decoder, 7447 is a BCD-to-Seven-Seg,emt Decpder/Driver, and 74154 is a 4-line-to-16-line (or 1-of-16) decoder.

Table. 9-1. Popular Decoder ICs

Device Number	*Function*
74138	3-line-to-8-line (or 1-of-8) decoder
7442	BCD-to-decimal (or 4-line-to-10-line) decoder
7447	BCD-to-Seven Segment Decoder/Driver
74154	4-line-to-16-line (or 4-of-10) decoder

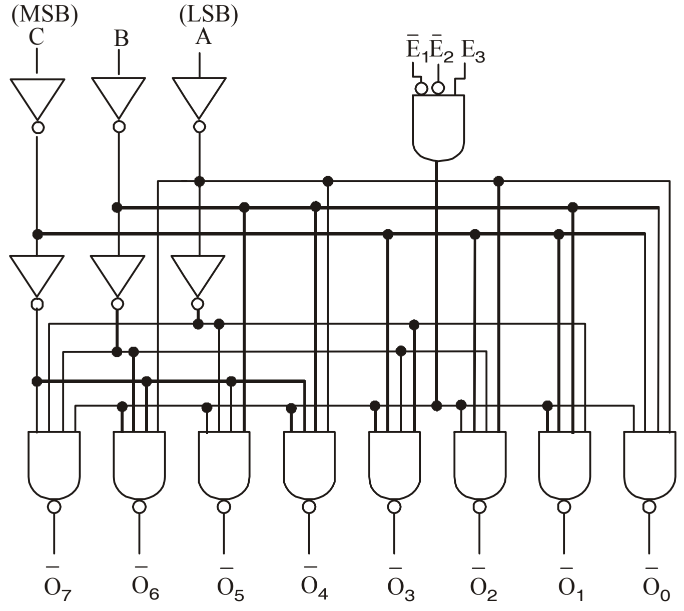
9-6. Decoder with ENABLE Inputs - IC 74138

We have already discussed in the last article that decoder is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to that number. Some decoders have one or more ENABLE inputs that are used to control the operation of the decoder. For example, refer to the decoder in Fig. 9-5 and visualize having a common ENABLE line connected to a fourth-input of each gate. With this ENABLE line held HIGH, the decoder will function normally – *i.e.*, with A, B and C input code will determine which output is HIGH. However, if ENABLE is held LOW, all of the outputs will be forced to the LOW state regardless of the levels at A, B and C inputs. Thus the decoder is enabled only if ENABLE is HIGH.

Fig. 9-6 (*a*) shows the logic diagram of a typical 74138 decoder. If we examine this diagrams carefully, we can determine exactly how this decoder functions. First of all, notice that 74138 decoder has NAND gate outputs. This means the outputs of this decoder are active –LOW. Secondly, the outputs are labelled as $\overline{O}_1, \overline{O}_2, \overline{O}_{3, ...,} \overline{O}_{6,}$ and $\overline{O}_7$. The over bar indicates active –LOW outputs.

The input code is applied at C, B and A inputs of the decoder. Note carefully that C is the most-significant bit (MSB), while A is the least-significant bit (LSB). With 3 inputs and 8 outputs, this is a 3-to-8 decoder or equivalently, a 1-of-8 decoder.

It may also be noted that 74138 decoder has three enable inputs labelled as E_1, E_2 and E_3. All the three enable inputs are combined in the AND gate as shown in Fig. 9-6(*a*).

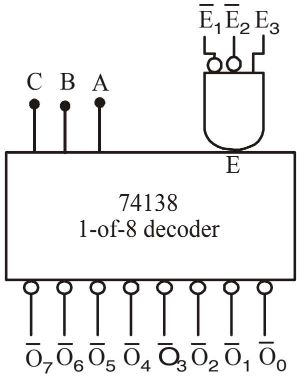


(*a*) *Logic diagram*

$\overline{E}_1$	$\overline{E}_2$	E_3	***Outputs***
0	0	1	Respond to input code CBA
1	X	X	Disabled – all HIGH
X	1	X	Disabled – all HIGH
X	X	0	Disabled – all HIGH

1 = HIGH voltage level
0 = LOW voltage level
X = 0 or 1

(*b*) Truth table


$\overline{E}_1 \overline{E}_2 E_3$
C B A
E
74138
1-of-8 decoder
$\overline{O}_7 \overline{O}_6 \overline{O}_5 \overline{O}_4 \overline{O}_3 \overline{O}_2 \overline{O}_1 \overline{O}_0$

(*c*) Logic symbol

Fig. 9-6. TTL IC 74138 decoder

In order to enable the output NAND gates to respond to the input code at C, B and A the AND gate output must be HIGH. This will occur only when $\overline{E}1 = \overline{E}2 = 0$ and $E_3 = 1$. In other words, $\overline{E}_1$ and $\overline{E}_2$ are active-Low and E_3 is active-HIGH. All these three enable inputs must be in their active states to activate the decoder outputs. It may be noted that if one or more of the enable inputs is in its inactive state, the AND gate output will be low. This will force all NAND gate outputs to their inactive HIGH state regardless of the input code. This operation is summarized in the truth table shown in Fig. 9-6 (*b*). Here X represents don't-care condition (i.e. the value can be logic 0 or 1). The complete operation is given in the form of a truth table shown in Fig. 9-7.

Enable			***Inputs***			***Active Output***
E_3	$\overline{E}_2$	$\overline{E}_1$	*C*	*B*	*A*	
1	0	0	0	0	0	$\overline{O}_0$
1	0	0	0	0	1	$\overline{O}_1$
1	0	0	0	1	0	$\overline{O}_2$
1	0	0	0	1	1	$\overline{O}_3$
1	0	0	1	0	0	$\overline{O}_4$
1	0	0	1	0	1	$\overline{O}_5$
1	0	0	1	1	0	$\overline{O}_6$
1	0	0	1	1	1	$\overline{O}_7$
X	X	1	X	X	X	None
X	1	X	X	X	X	None
0	X	X	X	X	X	None

Fig. 9-7. A complete Truth table for 74138 decoder

Fig. 9.7 (*c*) shows the logic symbol of 74138 decoder. Notice how the active-LOW outputs are represented and how the enable inputs are represented. Even though the enable AND gate is shown external to the decoder block, but it is a part of the IC's internal circuitry.

Example 9-1. *Implement the following function using 3 to 8 decoder.*

$$f(A, B, C) = \Sigma m\ (0, 1, 4, 5, 7) \qquad (\textit{PTU., May 2009})$$

Solution. Given the function

$$f(A, B, C) = \Sigma m\ (0, 1, 4, 5, 7)$$

There are three inputs and total five minterms, the implementation is shown in Fig. 9-8. The decoder generates the five minterms for *A*, *B*, *C*.

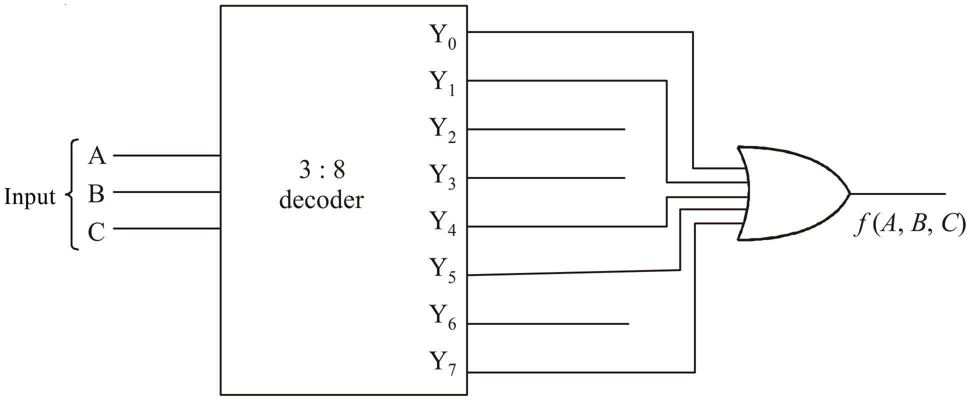


Fig. 9-8.

Example 9-2. *A combinational circuit is defined by following Boolean functions. Design a circuit with decoder and external gates.*

$$F_1(x, y, z) = \overline{x}\,\overline{y}\,\overline{z} + xz$$
$$F_2(x, y, z) = x\,\overline{y}\,z + \overline{x}\,z$$

(*GBTU, 2007*)

Solution. Given : The function

$$F_1(x, y, z) = \overline{x}\,\overline{y}\,\overline{z} + x\,z$$

$$F_2(x,y,z) = x\,\overline{y}\,z + \overline{x}\,z$$

The given functions are not in standard from, so we convert these function in standard form, taking functions one by one. Lets take function F_1

$$F_1\,(x,y,z) = \overline{x}\,\overline{y}\,\overline{z} + x\,z$$

The second term has missing variable y or $\overline{y}$ so we multiple the term by $(y + \overline{y})$

$$xz = xz\,(y + \overline{y})$$

$$= xzy + xz\overline{y} = xyz + x\overline{y}z$$

Thus the function F_1

$$F_1\,(x,y,z) = \overline{x}\,\overline{y}\,\overline{z} + xyz + x\overline{y}z.$$

Lets take functions F_2,

$$F_2\,(x,y,z) = x\overline{y}z + \overline{x}z$$

The second term has missing variable y or $\overline{y}$. So we multiply the term by $(y + \overline{y})$

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

These the function F_2,

$$F_2\,(x,y,z) = x\,\overline{y}\,z + \overline{x}\,y\,z + \overline{x}\,\overline{y}\,z$$

The truth table for the function F_1 and F_2 is shown in Table 9-2.

Table. 9-2.

Inputs			***Output Function***	
x	y	z	F_1	F_2
0	0	0	1	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	0

From the table 9.2 we see that the output function F_1 is obtained by ORed the output O_0, O_5 and O_7 of the decodes output. The output function F_2 is obtained by ORed the output O_1, O_3 and O_5 of the decoder output as shown in Fig. 9-9.

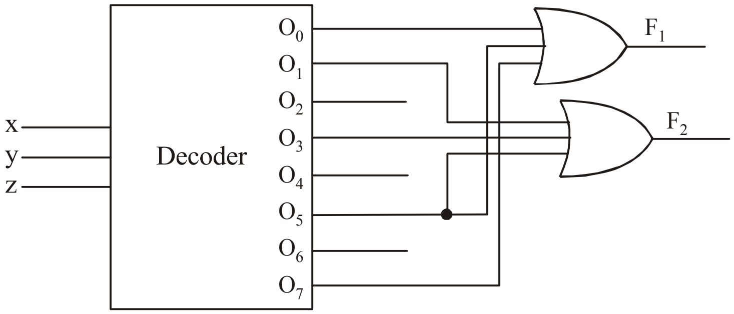


Fig. 9-9.

Example 9-3. *Implement using 3:8 decoders*

(*i*) $f(A, B, C) = \Sigma m\,(0, 1, 5, 6)$

(*ii*) $f(A, B, C) = \prod M\,(1, 2, 5, 6)$

(*Nagpur University, 2008*)

Solution. (*i*) *Given* : The function

$$f(A, B, C) = \Sigma m\,(0, 1, 5, 6)$$

Lets consider for active low output, when decoder output is active low, the output complemented form, i.e., it generates maxterms (sum terms) for input variables. The SOP is achieved by connecting NAND gate. The given function is the form of SOP, the output of decoder 0, 1, 5 and 6 is connected to NAND gate as shown in Fig. 9-10.

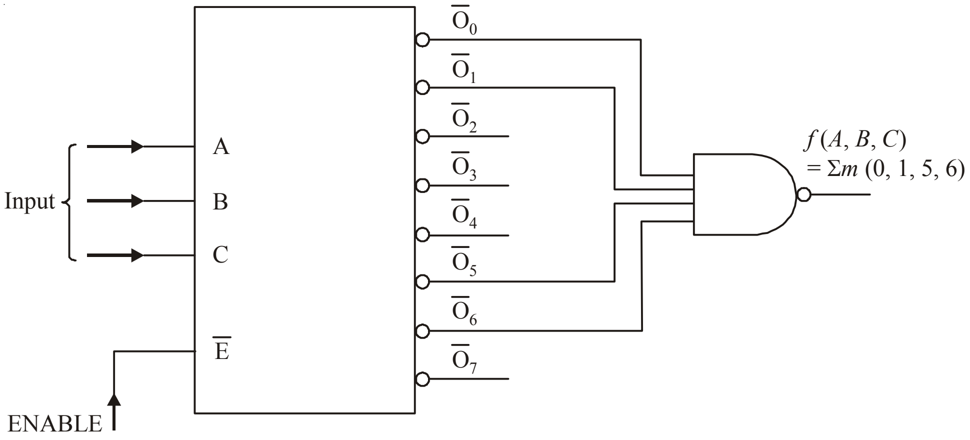


Fig. 9-10.

(*ii*) *Given* : The function

$$f(A, B, C) = \prod M\,(1, 2, 5, 6)$$

Lets consider for active low output, when decoder output is active low, the output complemented form, i.e., it generates maxterms (sum terms) for input variables. The POS is achieved by connecting AND gate. The given function is the form of POS, the output of decoder 1, 2, 5 and 6 is connected to AND gate as shown in Fig. 9-11.

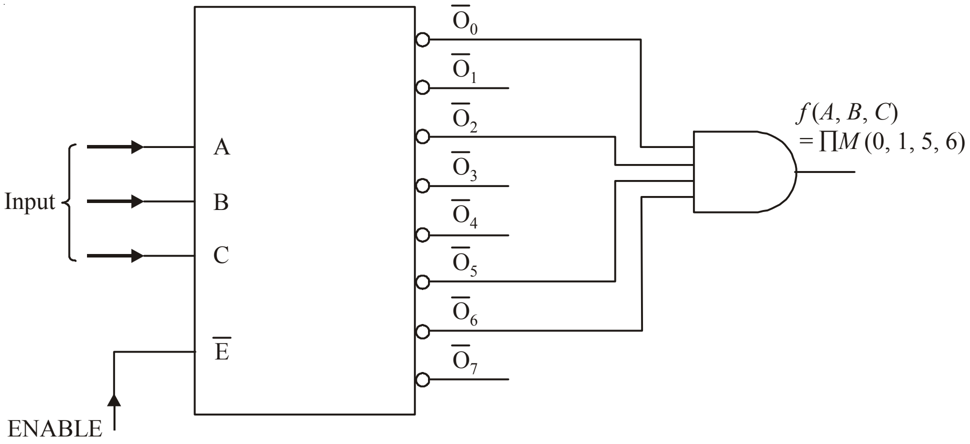


Fig. 9-11.

Example 9-4. *Implement the function using 3 : 8 decoder*

$$F_1\,(A, B, C) = \Sigma\,(0, 1, 3, 5, 6)$$
$$F_2\,(A, B, C) = \Sigma\,(0, 2, 4, 7)$$

(*GBTU/MTU, 2009-10*)

Solution. *Given* : The given functions are

$$F_1\,(A, B, C) = \Sigma\,(0,1,3,5,6)$$
$$F_2\,(A, B, C) = \Sigma\,(0,2,4,7)$$

From the function we know that there are three variables A, B and C and total number of term are 8. So we need 3-of-8 decoder. The table for the output function is shown in Fig. 9-12.

Inputs			***Output Function***	
A	B	C	F_1	F_2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Fig. 9-12.

The output function F_1 is obtained by ORed the output O_0, O_1, O_3, O_5 and O_6 of the decoder output. The output function F_2 is obtained by ORed the output O_0, O_2, O_4 and O_7 of the decoder output as shown in Fig. 9-13.

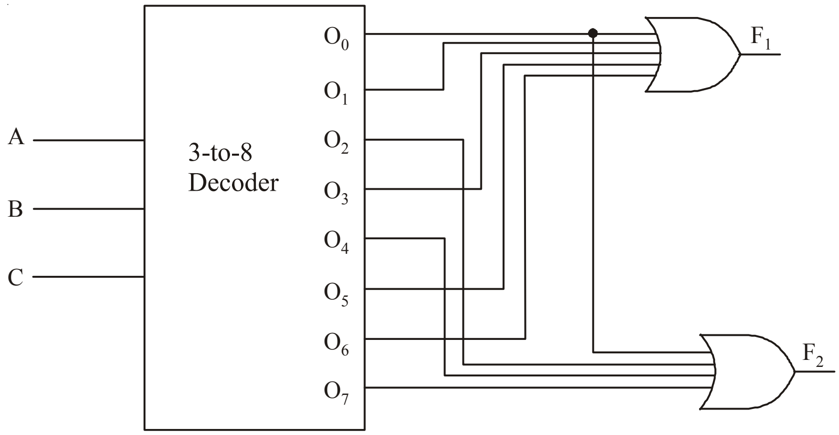


Fig. 9-13.

Example 9-5. *Show how to use* IC 74138 *(*1*-of-*8 *decoder) to form a* 1*-of-*16 *decoder.*

Solution. Given the IC 74138 (1-of-8 decoder).

Recall that 74138 has 3 lines (C, B and A) for an input code and three enable inputs labelled as $\overline{E}_1, \overline{E}_2$ and E_3. We can combine two 74138's together to form 1-of-16 decoder as shown in Fig. 9-14.

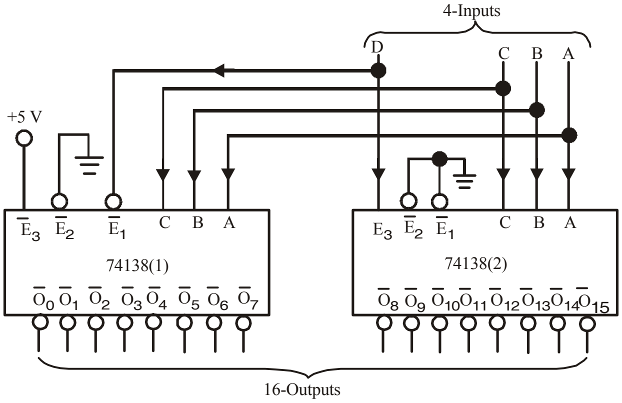

4-Inputs
D
C
B
A
+5 V
$\overline{E}_3$ $\overline{E}_2$ $\overline{E}_1$ C B A
74138(1)
$\overline{O}_0$ $\overline{O}_1$ $\overline{O}_2$ $\overline{O}_3$ $\overline{O}_4$ $\overline{O}_5$ $\overline{O}_6$ $\overline{O}_7$
E_3 $\overline{E}_2$ $\overline{E}_1$ C B A
74138(2)
$\overline{O}_8$ $\overline{O}_9$ $\overline{O}_{10}$ $\overline{O}_{11}$ $\overline{O}_{12}$ $\overline{O}_{13}$ $\overline{O}_{14}$ $\overline{O}_{15}$
16-Outputs

Fig. 9-14.

As seen from this diagram, three inputs CBA are applied to C, B and A lines of the decoders 74138 (1) and 74138 (2). The D input is connected to E_1 of 74138 (1) and $E_3$of 74138 (2). The enable inputs E_3 and E_2 of 74138(1) are permanently connected HIGH and LOW respectively. On the other hand both $\overline{E}_2$ and $\overline{E}_1$ of 74138(2) are permanently connected LOW.

The operation of the circuit shown in Fig. 9-14 may be explained as follows: when D = 0, the 74138 (1) decoder is enabled, so the input code on C, B and A lines will select one of its eight output lines (*i.e.*, $\overline{O}_0$ to $\overline{O}_7$). Similarly, when D = 1, the 74138(2) decoder is enabled, so the input code on C, B and A lines will select one of its eight output lines (*i.e.*, $\overline{O}_8$ to $\overline{O}_{15}$). The complete operation is summarized in the truth table shown below (refer to Table 9-3.)

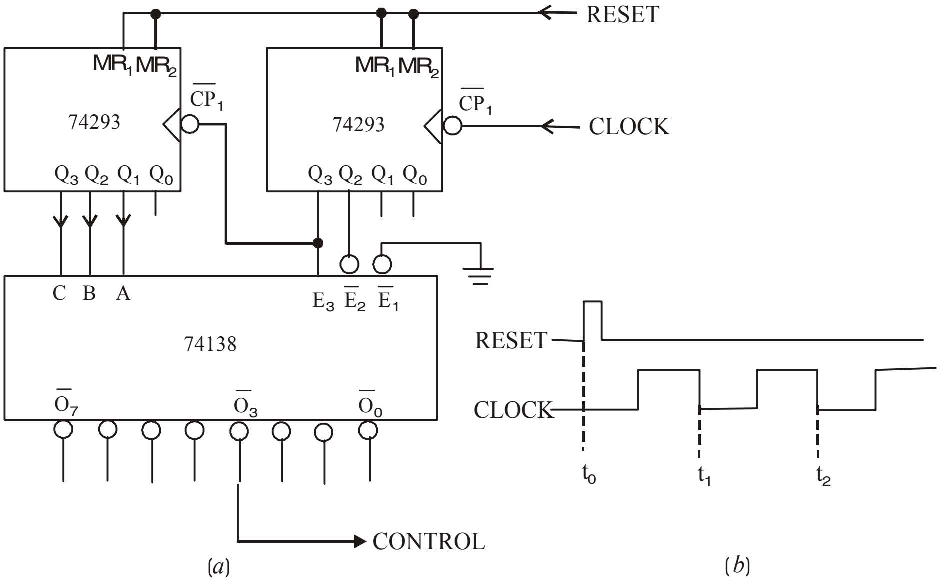
Table 9-3.

Inputs				***Active Output***
D	***C***	***B***	***A***	
0	0	0	0	$\overline{O}_0$
0	0	0	1	$\overline{O}_1$
0	0	1	0	$\overline{O}_2$
0	0	1	1	$\overline{O}_3$
0	1	0	0	$\overline{O}_4$
0	1	0	1	$\overline{O}_5$
0	1	1	0	$\overline{O}_6$
0	1	1	1	$\overline{O}_7$
1	0	0	0	$\overline{O}_8$
1	0	0	1	$\overline{O}_9$
1	0	1	0	$\overline{O}_{10}$
1	0	1	1	$\overline{O}_{11}$
1	1	0	0	$\overline{O}_{12}$
1	1	0	1	$\overline{O}_{13}$
1	1	1	0	$\overline{O}_{14}$
1	1	1	1	$\overline{O}_{15}$

It may be noted that for any input code on DCBA lines, only one of the output is active (LOW), all other outputs are HIGH.

Note. A 4-to-16-line decoder is available as IC 74154. The device is similar to 74 138 decorder. It has 4 input lines and 16 output lines. It has two ENABLE inputs indicated as $\overline{CS_1}$ and $\overline{CS_2}$.

Example 9-6. *Fig.* 9-15 *shows the use of a decoder for the generation of control signals. Assuming that a RESET pulse has occurred at time* $t_{0.}$*determine the CONTROL waveform for* 32 *clock pulses.*



(*a*) (*b*)

Fig. 9-15.

Solution. We know that 74293 is a four-bit binary counter (Refer to Art 7-10 in Chapter 7). The circuit shown in Fig. 9-15 employs two 74293s. Both the counters are connected as MOD-8 counter. Notice Q_0 output of both the counters is not used. Since the CONTROL waveform is to be generated at $\overline{O}_3$, therefore this is possible only when $\overline{O}_3$ is active. The output $\overline{O}_3$ will be active only when the left counter outputs $Q_3\,Q_2\,Q_1 = 011$ and the right counter outputs $Q_3\,Q_2\,Q_1 = 100$ or 101. The right counter outputs $Q_3\,Q_2 = 10$, will enable the decoder while Q_1 is unused. This condition corresponds to

$$\underbrace{011}_{\text{Left counter outputs}}\ \underbrace{100}_{\text{Right counter outputs}}{}_2 = 28_{10} \quad \text{or} \quad \underbrace{011}_{\text{Left counter outputs}}\ \underbrace{101}_{\text{Right counter outputs}}{}_2 = 29_{10}$$

Thus the CONTROL waveform at $\overline{O}_3$ will appear as shown in Fig. 9-7 below *i.e.*, it will go LOW during t_{28} and t_{29} falling edge of the clock pulse and will go HIGH at t_{30} and remains HIGH thereafter.

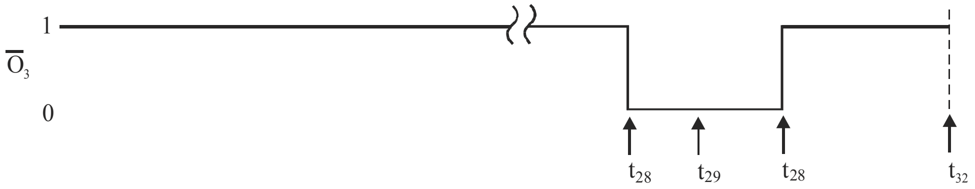


Fig. 9-16.

Example 9-7. *Without using any addition logic, modify the circuit shown in Fig.* 9-6 *(Example* 9-2*) to generate a CONTROL waveform that goes LOW from t_{20} to t_{24}.*

Solution. There are several possible solutions to this problem. One possible solution is as shown in Fig. 9-17. As seen from the figure the $\overline{E}_2$ input of the decoder has been disconnected from Q_2 of right 74293 and is permanently connected LOW.

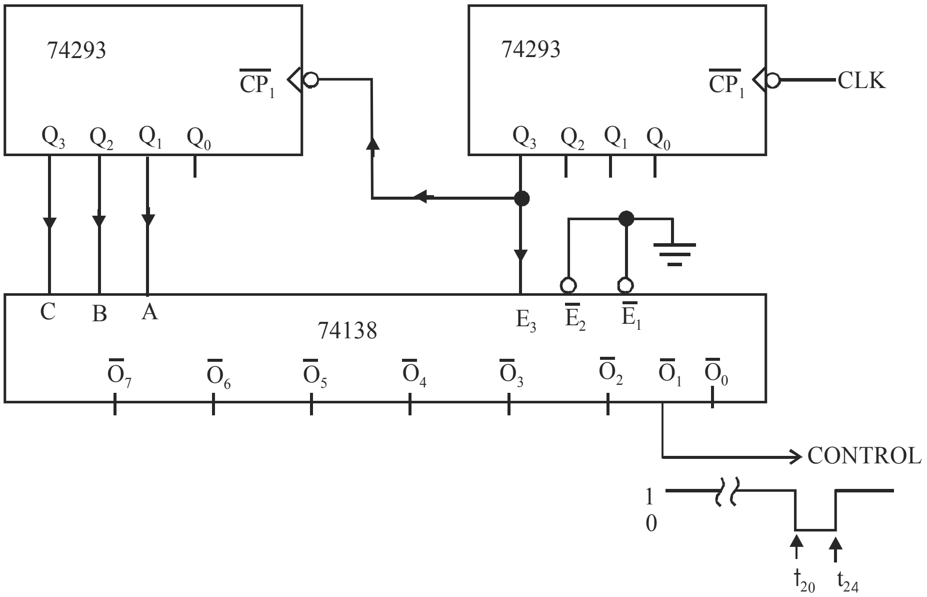

74293
$\overline{CP}_1$
Q_3 Q_2 Q_1 Q_0
74293
$\overline{CP}_1$
CLK
Q_3 Q_2 Q_1 Q_0
C B A
74138
E_3 $\overline{E}_2$ $\overline{E}_1$
$\overline{O}_7$ $\overline{O}_6$ $\overline{O}_5$ $\overline{O}_4$ $\overline{O}_3$ $\overline{O}_2$ $\overline{O}_1$ $\overline{O}_0$
CONTROL
1
0
t_{20} t_{24}

Fig. 9-17.

Note that 20 corresponds to 010100 and 24 corresponds to 011000.

$$\begin{array}{rcccccc} & C & B & A & \overline{E}_3 & - & - \\ 20_{10} = & 0 & 1 & 0 & 1 & 0 & 0 \end{array} \qquad \begin{array}{rcccccc} & C & B & A & \overline{E}_3 & & \\ 24_{10} = & 0 & 1 & 1 & 0 & 0 & 0 \end{array}$$

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

decoder is enabled if $E_3 = 1$ decoder is disabled if $E_3 = 0$

Thus the modified circuit will generate a CONTROL waveform which is LOW from t_{20} to t_{24} at $\overline{O}_2$ as shown in Fig. 9-17.

Example 9-8. Fig. 9-9 *shows the waveforms applied at the A, B and C and enable input E_3 of a 74138 decoder respectively.*

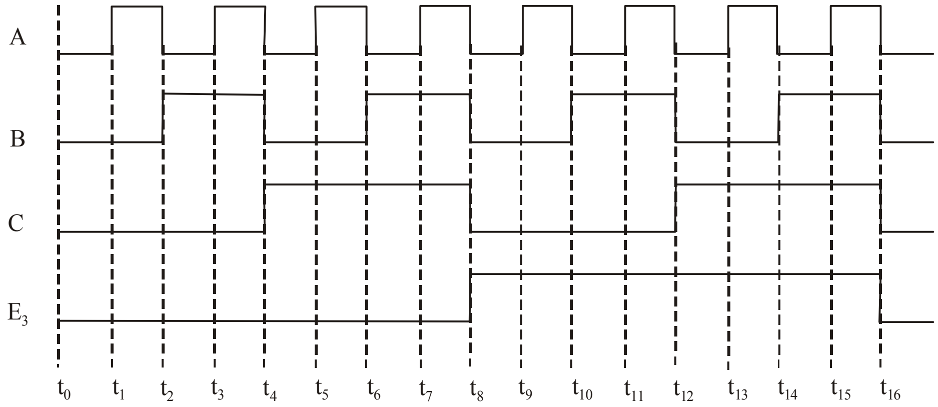


Fig. 9-18.

Assuming $\overline{E}_1$ and $\overline{E}_2$ are tied LOW, sketch the waveforms at the outputs $\overline{O}_0, \overline{O}_2, \overline{O}_4$ and $\overline{O}_6$.

Solution. We know that the output of the 74138 decoder is given by the truth table shown

Enable			*Inputs*			*Active Output*
E_3	$\overline{E}_2$	$\overline{E}_1$	*C*	*B*	*A*	
1	0	0	0	0	0	$\overline{O}_0$
1	0	0	0	0	1	$\overline{O}_1$
1	0	0	0	1	0	$\overline{O}_2$
1	0	0	0	1	1	$\overline{O}_3$
1	0	0	1	0	0	$\overline{O}_4$
1	0	0	1	0	1	$\overline{O}_5$
1	0	0	1	1	0	$\overline{O}_6$
1	0	0	1	1	1	$\overline{O}_7$

Fig. 9-19. Truth table of a 74138 decoder

in Fig. 9-19. As seen from this truth table, we find that $\overline{O}_0$ will be active (LOW) only when CBA = 000. From the input waveforms applied at the CBA inputs of the decoder, we find that during the time

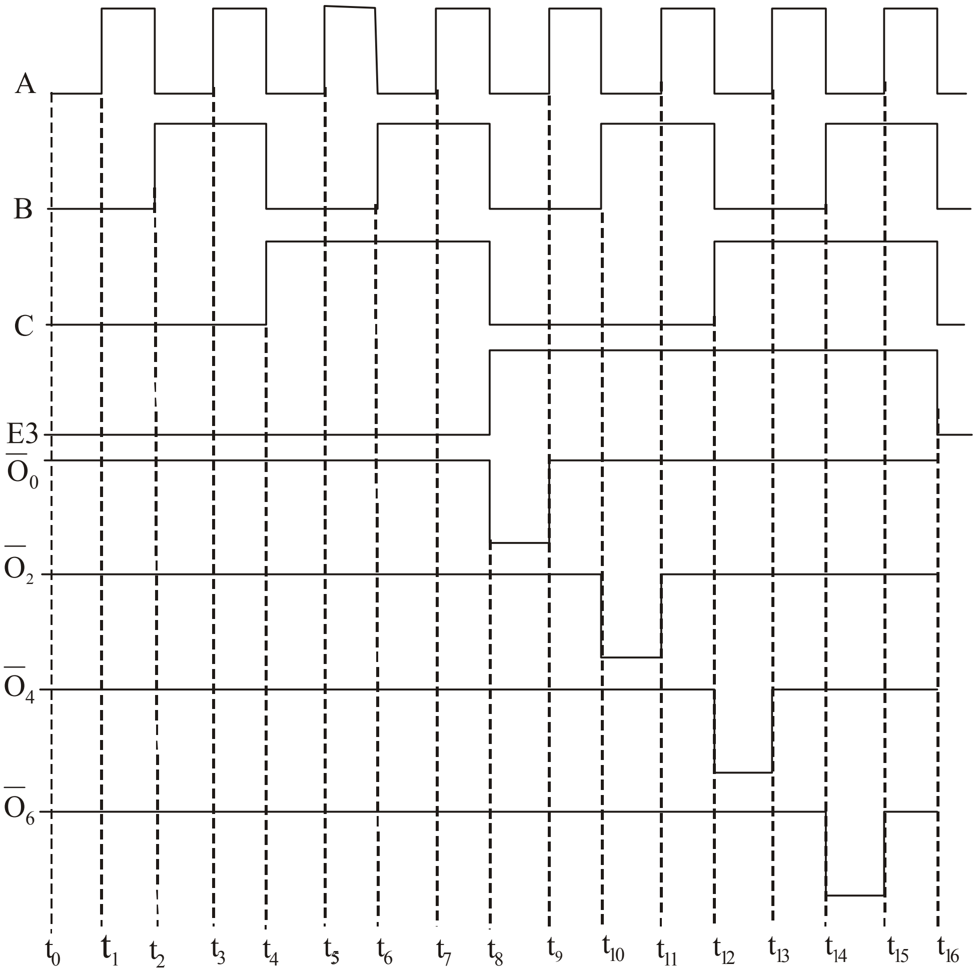


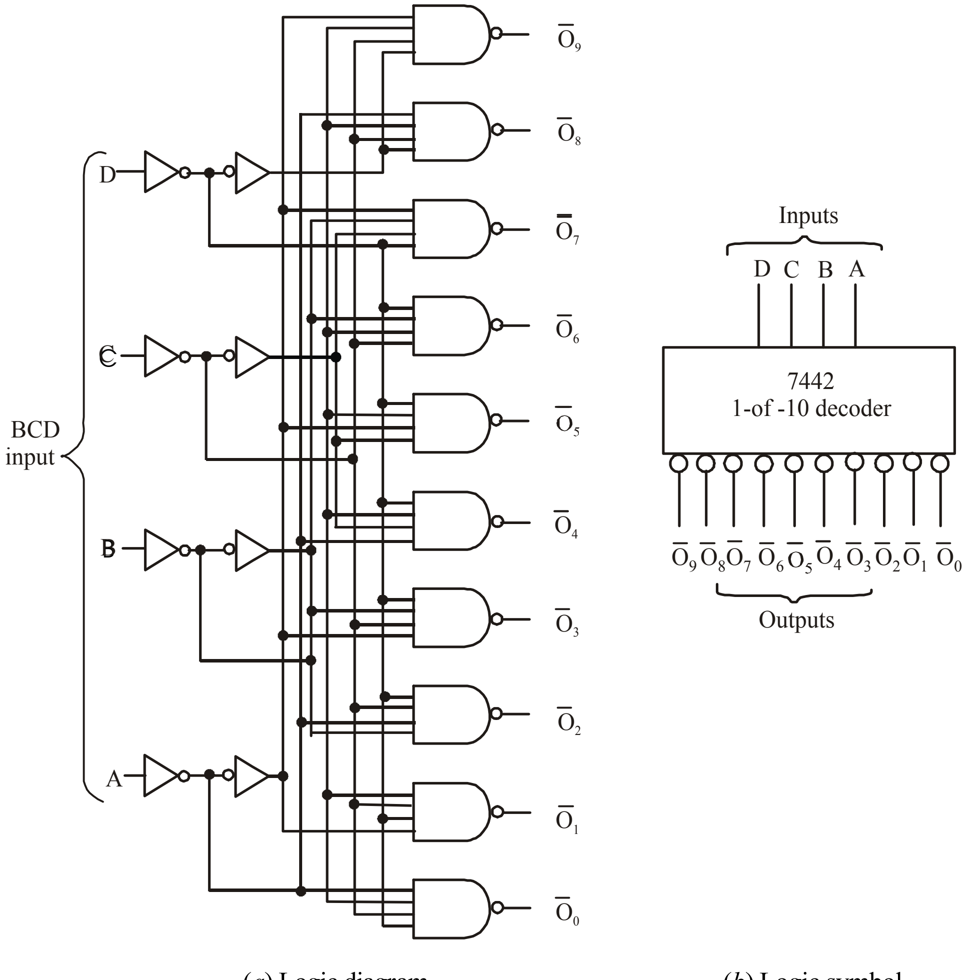
Fig. 9-20.

period between $t_8 – t_9$, C = 0, B = 0 and A = 0. Therefore $\overline{O}_0$ will be active (LOW) during this time period and the output waveform is sketched for $\overline{O}_0$ as shown in Fig. 9-20. Similarly, $\overline{O}_2$ will be active (LOW) only when CBA = 010. Again from the input waveforms we find that during the time period between t_{10} - t_{11}, C = 0, B =1 and A= 0. Therefore $\overline{O}_2$ will be active (LOW) during this time period and the output waveform for $\overline{O}_2$ is sketched as shown in Fig. 9-11.

In a similar manner, we find that $\overline{O}_4$ is active (LOW) during the time period t_{12} - t_{13}, and $\overline{O}_6$ is active (LOW) during t_{14} - t_{15}. So the output waveforms for these two outputs are sketched as shown in Fig. 9-20.

9-7. BCD-to-Decimal Decoders

Fig. 9-21 (*a*) shows the logic diagram for a TTL IC 7442, *BCD-to-decimal decoder.* Each output of the decoder goes LOW only when its corresponding BCD input is applied. For example, $\overline{O}_6$ will go LOW only when inputs DCBA = 0110; $\overline{O}_7$ will go LOW only when inputs DCBA = 0110. It may be noted that BCD input combinations ranging from DCBA = 0000 through 1001 are valid whereas those ranging from 1010 through 1111 are invalid. For invalid BCD input combinations, none of the outputs will be activated.



(*a*) Logic diagram (*b*) Logic symbol

Fig. 9-21. TTL IC 7442 – a BCD to decimal decoder

Table 9-4. Truth Table for IC 7442-a BCD-to-decimal decoder

Inputs				*Active output*
D	***C***	***B***	***A***	
0	0	0	0	$\overline{O}_0$
0	0	0	1	$\overline{O}_1$
0	0	1	0	$\overline{O}_2$
0	0	1	1	$\overline{O}_3$
0	1	0	0	$\overline{O}_4$
0	1	0	1	$\overline{O}_5$
0	1	1	0	$\overline{O}_6$
0	1	1	1	$\overline{O}_7$
1	0	0	0	$\overline{O}_8$
1	0	0	1	$\overline{O}_9$
1	0	1	0	None
1	0	1	1	None
1	1	0	0	None
1	1	0	1	None
1	1	1	0	None
1	1	1	1	None

1 = HIGH Voltage Level   0 = LOW Voltage Level

The TTL IC 7442 can also be referred to as a *4-to-10 decoder* or a *1-of-10 decoder*. The logic symbol for this decoder is shown in Fig 9-21 (*b*) and truth table is shown in Table 9-4.

Example 9-9. *The BCD-to-decimal decoder* (7442) *shown in Fig* 9-21 (*a*) *does not have an ENABLE input. However, we can operate it as a* 1-*of*-8 *decoder by not using outputs* $\overline{O}_8$ *and* $\overline{O}_9$ *and by using the D input as an ENABLE. This is illustrated in Fig* 9-22.

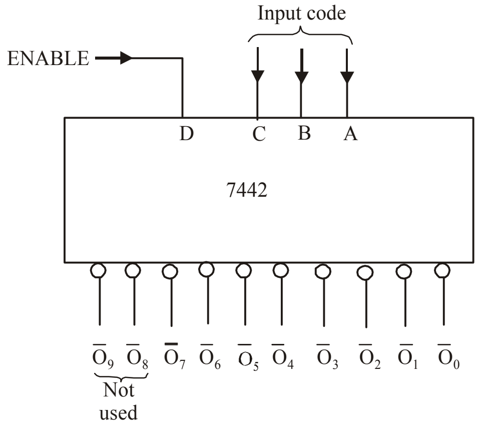


Fig 9-22.

Describe how this arrangement works as an enabled 1-*of*-8 *decoder and state how the level on D either enables or disables the outputs.*

Solution. Given : the BCD-to-decimal decoder 7442 shown in Fig. 9-22. We know that when D = 0, the C, B and A inputs determine which one of outputs $\overline{O}_7$ to $\overline{O}_0$ will be activated (*i.e.*, goes LOW). However, if D = 1, all the outputs will be inactive (*i.e.*, HIGH) because the input code will be greater than 0111_2 (7_{10}).

9-8. BCD-to-Decimal Decoder/Driver

The TTL IC 7445 is a BCD-to-decimal decoder/*driver*. The term driver is added to its description because this IC has open–collector outputs that can operate at high current and voltage limits than a normal TTL output. The IC 7445's outputs can sink up to 80 mA in the LOW state and they can be pulled up to 30 V in the HIGH state. This makes them suitable for directly driving loads such as indicator LEDs or lamps, relays, dc motors etc.

9-9. Applications of Decoder

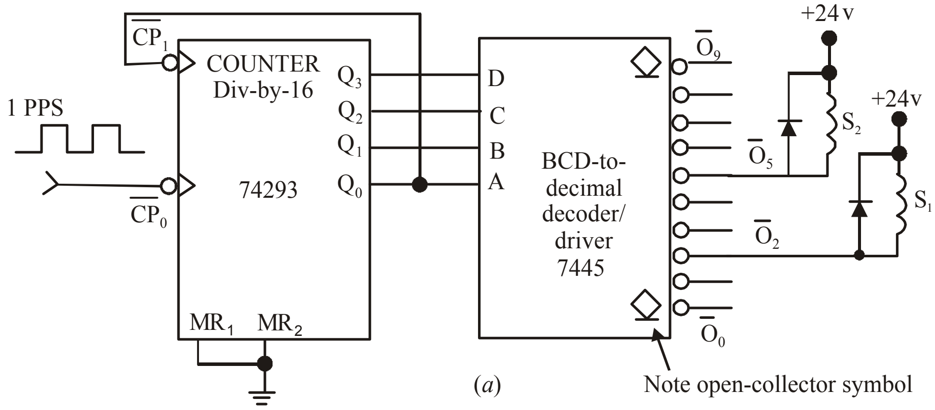
Although there are several applications of decoders in digital systems, yet the followings are important from the subject point of views:

1. Can be used for generating timing or sequencing signals to turn devices on or off at specific times.
2. Used in memory system of a computer where they respond to the address code generated by the microprocessor to activate a particular memory location.
3. Used in computers for selection of external devices that include printers, modems, scanners, internal disk drives, keyboard, video monitor etc.

All these applications are discussed one by one in the following pages.

9-10. Decoder in Timing and Sequencing Operation Circuit

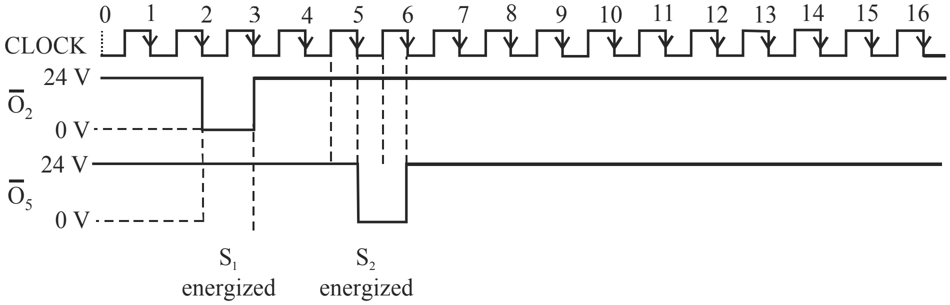
Fig. 9-23 shows the use of a decoder to provide timing and sequencing operations. The circuit shows the combination of two TTL ICS namely 74293 (binary counter) and 7445 (BCD-to-decimal decoder/driver). As seen from the diagram, the counter is being pulsed by a 1-pps (pulse per second) signal so that it will sequence through the binary counts at the rate of 1 count per second. The counter flip-flop outputs are connected as inputs to the decoder. The 7445 open-collector outputs $\overline{O}_2$ and $\overline{O}_5$ are used to switch relays S_1 and S_2 on and off.



(*a*) Note open-collector symbol

Fig. 9-23.

In order to understand the operation of the circuit, let us suppose $\overline{O}_2$ is in its inactive-HIGH state. This means its output transistor will be off (*i.e.* non-conducting) so that no current can flow through relay S_1 and it will be de-energised. When $\overline{O}_2$ is in active – LOW state, its output transistor is on and acts as a current sink for current through S_1 so that S_1 is energised. It may be noted that relays operate from +24 V supply. Also note the presence of the diodes across the relay coils. These diodes protect the decoder's output transistors from the large "inductive kick" voltage that could be produced when coil current is abruptly stopped.



(b)

Fig. 9-24.

Fig. 9-24 shows the timing diagram for the sequence of events. Assuming the counter is in 0000 state at time 0, the timing diagram shows that both outputs $\overline{O}_2$ and $\overline{O}_5$ are initially in the inactive-HIGH states. This means their output transistors are OFF and both relays are de-energised. As the clock pulses are applied, the counter will be incremented once per second. On the falling edge of the second pulse (time 2), the counter will go to the 0010 (2) state. This will activate decoder output $\overline{O}_2$ and thereby energise relay S_1. On the falling edge of the third pulse, the counter goes to 0011(3) state. This will deactivate $\overline{O}_2$ and de–energise relay S_1.

Similarly at time 5, the counter will go to the 0101(5) state. This will activate decoder output $\overline{O}_5$ and energise S_2. At time 6 the counter goes to 0110 (6) and deactivate $\overline{O}_5$ to deenergise S_2. The counter will continue counting as pulses are applied. After 16 pulses, the sequence described above will start over again.

9-11. Decoder in Memory System of a Computer

As a matter of fact, decoders are widely used in the memory system of a computer. Here they respond to the address code generated by the central processor to activate a particular memory location. Each memory IC contains many registers that can store binary numbers (called data). Each register needs to have its own unique address to distinguish it from all the other registers. A decoder is built into the memory IC's circuitry. This allows a particular storage register to be activated when a unique combination of inputs (*i.e.*, its address) is applied. In a typical digital system, there are usually several memory ICs combined to make up the entire storage capacity. A decoder is used to select a memory chip in response to a range of addresses by decoding the most significant bits of the system address and enabling (or selecting) a particular chip. This application will be examined in more detail in the chapter on memories.

In more complicated memory systems, the memory chips are arranged in *multiple banks.* These banks must be selected individually or simultaneously depending whether the microprocessor wants one or more bytes at a time. This means that under certain circumstances, more than one output of the decoder must be activated. For systems such as this, a *programmable logic device* (PLD) is often used to implement the decoder. Since a simple 1-of-8 decoder alone is not sufficient, programmable logic devices can easily be used for custom decoding applications.

9-12. Decoder for Selecting Input/Output Devices

Computers must communicate with a variety of external devices called *peripherals* by sending / receiving data through input /output (I/O) ports. These external devices include printers, modems scanners, external disk drives, keyboard, video monitor, CD-ROM drives, floppy disk drives and other computers.

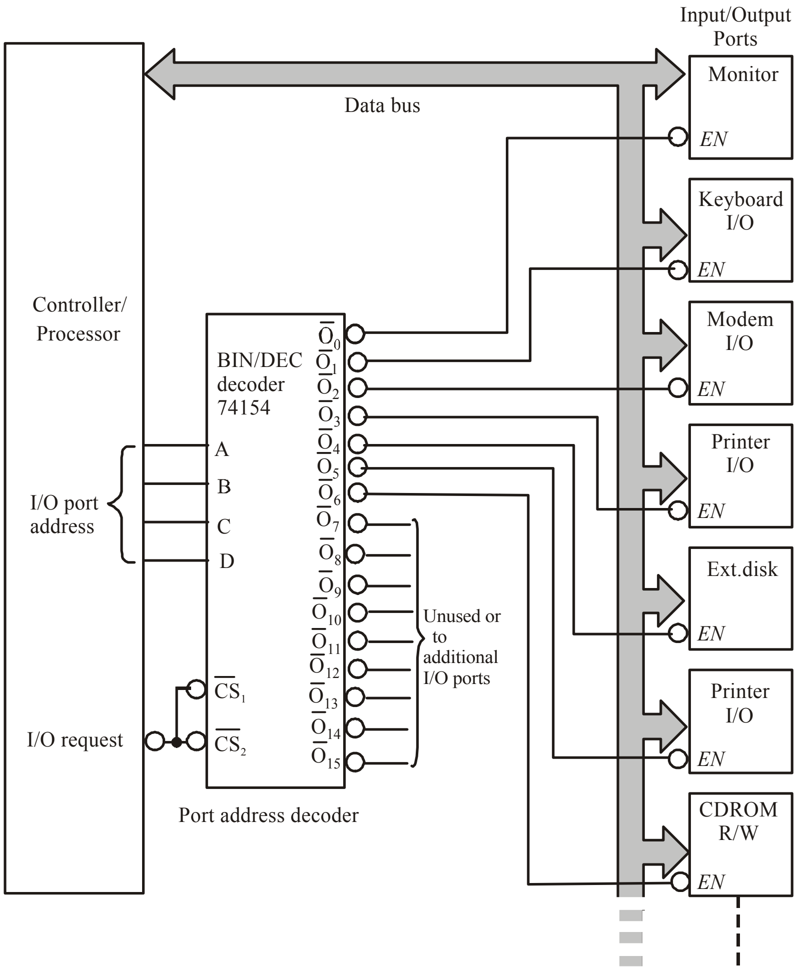


Fig. 9-25.

As seen from the diagram shown in Fig. 9-25 a 4-line-to-16-line decoder is used to select the I/O port as determined by the computer so that data can be sent or received from a specific external device. It may be noted that each I/O port has a number called as an address, which uniquely identifies it. When the computer wants to communicate with a particular device, it issues the appropriate code for the I/O part to which that particular device is connected. This binary port address is decoded and the appropriate decoder output is activated to enable the I/O port. It may be also be noted that binary data is transferred within the computer on a data bus, which is a group of parallel lines. For example, an 8-bit data bus consists of eight parallel lines that can carry one byte of data at a time. The data bus goes to all the I/O ports. However, any data coming in or going out will only pass through the port that is enabled by the port address decoder.

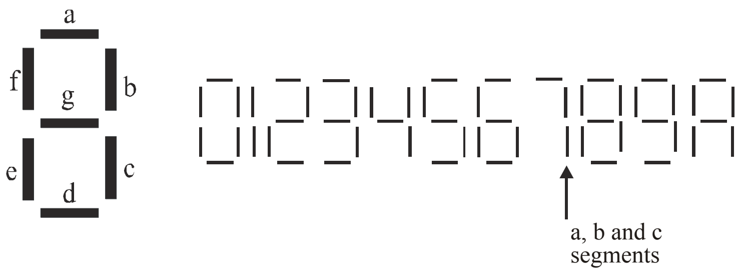
9-13. BCD-to-Seven-Segment Decoder/Drivers

We have already discussed in chapter 7 (counter and Registers) the counter circuits that are used to display decimal (i.e 0 through 9) numbers. In such circuits, the counters must output BCD (binary coded decimal) numbers.

As described in chapter 2, BCD is a 4-bit binary number which is used to represent the 10 decimal digits. To be useful, the BCD must be decoded by a decoder into a format that can be used to drive a decimal numeric display. The most popular method to display is the seven-segment LED display.

A seven-segment display is actually made up of seven separate light-emitting diodes (LEDs) in a single package. The LEDs are oriented in such a way that they form a figure of 8. Most seven-segment LEDs have an eighth LED used for a decimal point. Refer to Fig. 9-18 (*a*).

The function of a decoder is to convert the 4-bit BCD code into a seven-segment code that will turn on the appropriate LED segements to display the correct decimal digit. Fig. 9-18 (*b*) shows the numerical disignations for the 10 allowable digits.



(a) (b)

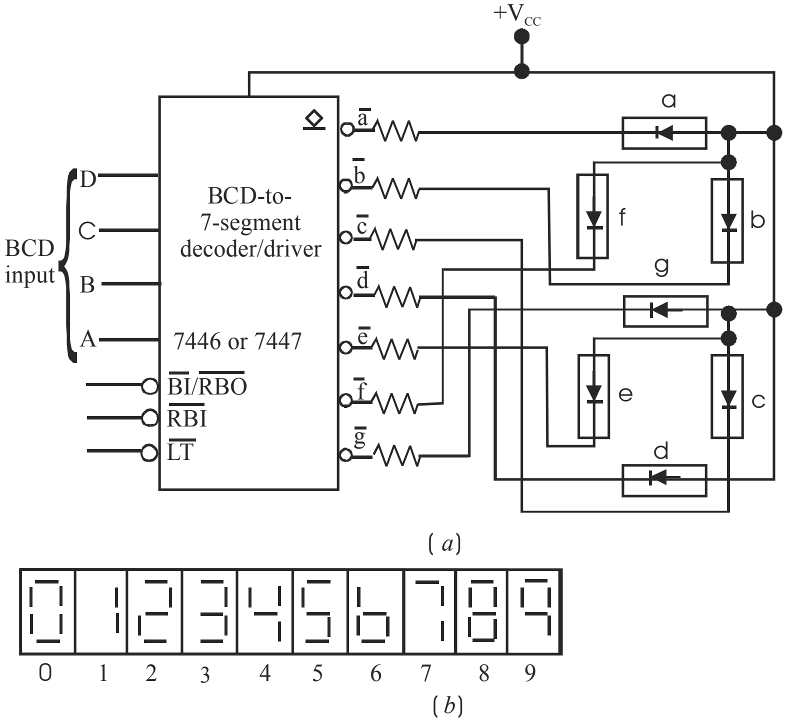
Fig. 9-26.

A BCD-to-seven-segment decoder/driver is used to take a four-bit BCD input and provide the outputs that will pass through the appropriate segments to display the decimal digit. The logic diagram for this decoder is more complicated than the logic diagram of 3-to-8-line decoder or 4-to-16 line decoders. It is due to the fact that each output is activated for more than one combination of inputs. For example, the c segment has to be activated for any of the digits 0, 3, 4, 5, 6, 8 and 9.

Fig. 9-27 (*a*) shows a seven-segment decoder/driver (TTL IC 7446/7447) being used to drive a seven-segment LED readout. It may be carefully noted that each segment of the readout consists of one or two LEDs. The anodes of the LEDs are all connected to V_{CC} (*i.e.*, +5 V) supply. The cathodes of the LEDs are connected through current-limiting resistors to the appropriate outputs of the decoder/driver. Notice that the decoder/driver has active-LOW outputs. Which are open-collector driver transistors that can sink a fairly large value of current. This is because of the fact that the LED readouts may require 10 mA to 40 mA per segment depending on their type and size.

In order to illustrate the operation of the circuit, consider a BCD input, DCBA = 0110 which is a BCD number for 6. With these inputs, the decoder/driver outputs $\bar{a}, \bar{c}, \bar{d}, \bar{e}, \bar{f}$ and $\bar{g}$ will be driven LOW. This will allow the current to pass through the a, c, d, e, f and g segments thereby displaying the number 6. The $\bar{b}$ output will be HIGH so that the LED segment b cannot conduct.

As a matter of fact, seven-segment decoders/drivers such as the 7446/7447 are exceptions to the rule that decoder circuits activate only one output for each combination of inputs. Rather, they activate a unique pattern of outputs for each combination of inputs.



(*a*)

(*b*)

Fig 9-27.

Notes.

1. The LED display used in Fig. 9-27 is a common-anode type. In such a display, the anodes of all the segments are connected together to V_{CC} supply. There is another type of seven-segment display called common-cathode type. In this type of display, the cathodes of all the segments are connected together to ground. The common-cathode type seven-segment display must be driven by a BCD-to-seven segment decoder/driver with active-HIGH outputs. An active-HIGH output will apply a HIGH voltage to the anodes of those segments that are to be activated.

2. Since each segment requires 10 mA to 20 mA of current to light it, therefore, TTL and CMOS devices are normally not used to drive the common-cathode type display directly. Recall from Chapter 8 that TTL and CMOS outputs are not able to source large amounts of current. In order to solve this problem, a transistor interface circuit is often used between decoder chips and the common-cathode type seven-segment display.

3. In addition to decoding a BCD input and producing the appropriate seven-segment outputs, 7446/7447 has lamp test $\left(\overline{\text{LT}}\right)$ and blanking controls $\left(\overline{\text{BI}}/\overline{\text{RBO}}\text{ and }\overline{\text{RBI}}\right)$. These inputs are used for testing all the seven segments and zero suppression. The zero suppression is a feature used for multidigit display to blank out unnecessary zeros.

Example 9-10. *Show how to connect BCD-to-*7 *segment decoder/drivers and LED* 7*-segment displays to the clock circuit of Fig.* 7-27 *to display minutes and hours. Assume that each segment is to operate at approximately* 10 *mA at* 2.5 *V.*

Solution. Fig. 9-28 shows the circuit arrangement to display minutes. As seen the circuit makes use of two BCD counters which constitute MOD-60 counter.

Minutes section

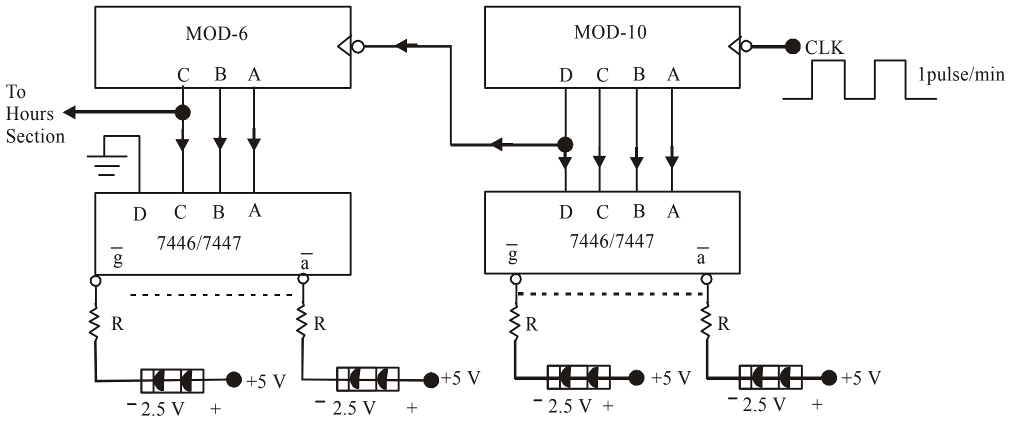


Fig. 9-28.

In order to operate each segment at approximately 10 mA at 2.5 V, the value of resistor

$$R = \frac{5\,\text{V} - 2.5\,\text{V}}{10\,\text{mA}} = 250\,\Omega$$

The same idea is used for Hours Section.

9-14. Encoders

The function of an encoder is opposite to that of a decoder. Encoder is used to generate a coded output (such as binary or BCD) from the active input line.

An encoder has a number of input lines, only one of which is activated at a given time and produces an N-bit output code, depending on which input is activated. Fig. 9-29 shows a general diagram for an encoder with M inputs and N outputs. Here the inputs are active-HIGH, which means they are normally low.

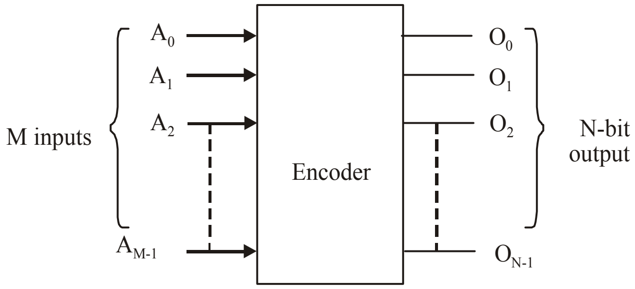
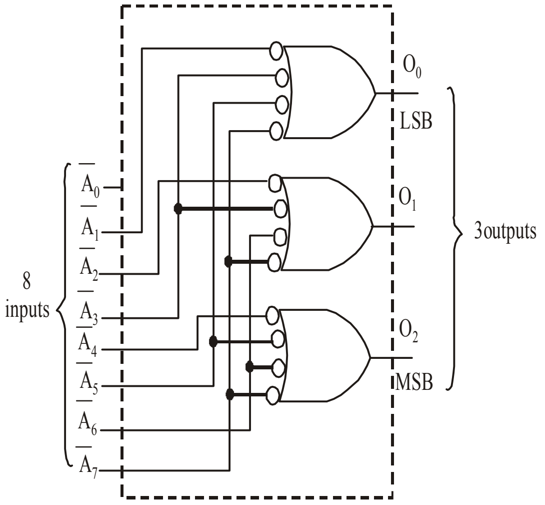


Fig. 9-29. General encoder diagram

Fig. 9-30 (*a*) shows the logic diagram of a octal-to-binary (8-line-to-3-line) encoder with active low inputs. It accepts 8 input lines and produces a 3-bit output code corresponding to activated input. Fig. 9-30 (*b*) shows the truth table for this encoder.



(*a*) Logic diagram

Inputs								***Outputs***		
$\overline{A}_0$	$\overline{A}_1$	$\overline{A}_2$	$\overline{A}_3$	$\overline{A}_4$	$\overline{A}_5$	$\overline{A}_6$	$\overline{A}_7$	O_2	O_1	O_0
X	1	1	1	1	1	1	1	0	0	0
X	0	1	1	1	1	1	1	0	0	1
X	1	0	1	1	1	1	1	0	1	0
X	1	1	0	1	1	1	1	0	1	1
X	1	1	1	0	1	1	1	1	0	0
X	1	1	1	1	0	1	1	1	0	1
X	1	1	1	1	1	0	1	1	1	0
X	1	1	1	1	1	1	0	1	1	1

(*b*) Truth table

Fig. 9-30. Octal-to-binary (8-line-to-3-line) encoder

It may be noted from the encoder's logic diagram that $\overline{A}_0$ is not connected to the logic gates. It is because of the fact that encoder outputs will normally be at 000 when none of the inputs $\overline{A}_1$ to $\overline{A}_9$ is LOW. By following through the logic, we can verify that a low at any single input will produce the output binary code corresponding to that input. For instance, a LOW at $\overline{A}_1$ (while all other inputs are HIGH) will produce $O_2 = 0$, $O_1 = 0$ and $O_0 = 1$. This is the binary code for 1. Similarly a LOW at $\overline{A}_2$ (while all other inputs are HIGH) will produce $O_2 = 0$, $O_1 = 1$ and $O_0 = 0$. This is the binary code for 2.

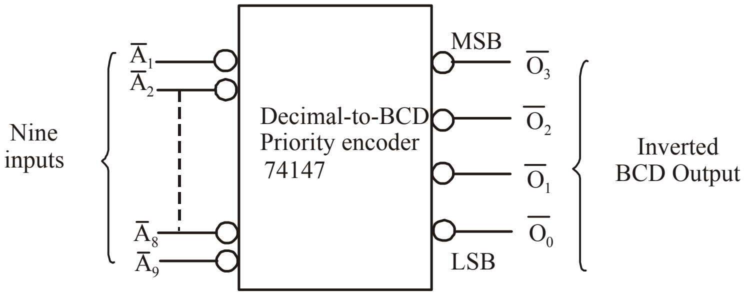
9-15. Priority Encoders

The encoder discussed in the last article requires only one input to be LOW at any time to produce a valid output code. However, if at any instant, more than one inputs are LOW, the output code will be an invalid code. Consider for example, a situation when $\overline{A}2$ and $\overline{A}4$ are simultaneously LOW. Following through the logic gates, we find that LOW at these two inputs will produce HIGHs at each output. In other words, the binary code 110 is produced at the outputs. Clearly, this is not the code for either activated input. This is a drawback of the simple encoder circuit shown in Fig. 9-30 (*a*).

A modified version of the simple encoder circuit is called a *priority encoder.* It includes the necessary logic to ensure that when two or more inputs are activated, the output code will correspond to the *highest* numbered input. For example, if $\overline{A}_2$ and $\overline{A}_4$ are LOW, the output code will be 100 (4). Similarly when, $\overline{A}_5$, $\overline{A}_3$ and $\overline{A}_0$ are all LOW, the output code is 101 (5). The TTL ICs 74148, 74LS148 and 74HC 148 are all octal–to–binary priority encoders.

9-16. Decimal–to-BCD Priority Encoder

Fig. 9-31 (*a*) shows the logic symbol for an IC 74147. It functions as a decimal–to–BCD priority encoder. Its truth table is shown in Fig. 9-31 (*b*). As seen from the logic diagram, the encoder has nine active-LOW inputs representing the decimal digits 1 through 9. It produces the *inverted* BCD code corresponding to the highest-numbered activated input.



(*a*) Logic diagram

Inputs									***Outputs***			
$\overline{A}_1$	$\overline{A}_2$	$\overline{A}_3$	$\overline{A}_4$	$\overline{A}_5$	$\overline{A}_6$	$\overline{A}_7$	$\overline{A}_8$	$\overline{A}_9$	$\overline{O}_3$	$\overline{O}_2$	$\overline{O}_1$	$\overline{O}_0$
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

X = either 0 or 1

(*b*) Truth table

Fig. 9-31. Decimal-to-BCD priority encoder - 74147

In order to understand the operation of 74147, let us examine its truth table. The first line in the truth table shows all inputs in their inactive HIGH state. For this condition, the outputs are 1111, which is inverse (or complement) of 0000, the BCD code for 0. The second line in the truth table indicates that a LOW at, $\overline{A}_9$ regardless of the states of the other inputs, will produce an output code of 0110, which is inverse of 1001, the BCD code for 9. The third line in the truth table indicates a LOW at $\overline{A}_8$, provided that $\overline{A}_9$ is high will produce an output code of 0111, the inverse of 1000, the BCD code for 8. In a similar manner, the remaining lines in the truth table show that a LOW at any input, provided that all higher-numbered inputs are HIGH, will produce the inverse of the BCD code for that input.

It may be noted that the 74147 outputs will normally be HIGH when none of the inputs are activated. This corresponds to the deimal 0 input condition. Notice that there is *no* $\overline{A}_0$ input indicated in logic diagram or truth table. It is because of the fact that the encoder assumes the decimal 0 input state when all other inputs are HIGH.

The 74147 *inverted BCD outputs* can be converted to *normal BCD* by putting each one through an INVERTER.

Example 9-11. *Determine the output levels for the decimal-to-BCD priority encoder* (74HC147) *when* $\overline{A}_7 = \overline{A}_3 = 0$ *and all other inputs are HIGH.*

Solution. The 74147 is a priority encoder, *i.e.*, when two or more inputs are activated (LOW) at the same time, the output code will correspond to the highest-numbered input. In the present case, the 74147 has $\overline{A}_7 = \overline{A}_3 = 0$, So the device will produce an output code that corresponds to $\overline{A}_7$.

Now since the 74147 has inverted outputs, refer to the truth table of 74147 shown in Fig. 9-22 (*b*) (page 394), therefore, the output code will be complement of the BCD code for 7, *i.e.*, $\overline{0111} = 1000$

Example 9-12. *Consider the waveforms shown in Fig.* 9-32. *Apply these waveforms to the* 74147 *encoder as follows*:

$$A \rightarrow \overline{A}_7, B \rightarrow \overline{A}_4, C \rightarrow \overline{A}_2, D \rightarrow \overline{A}_1$$

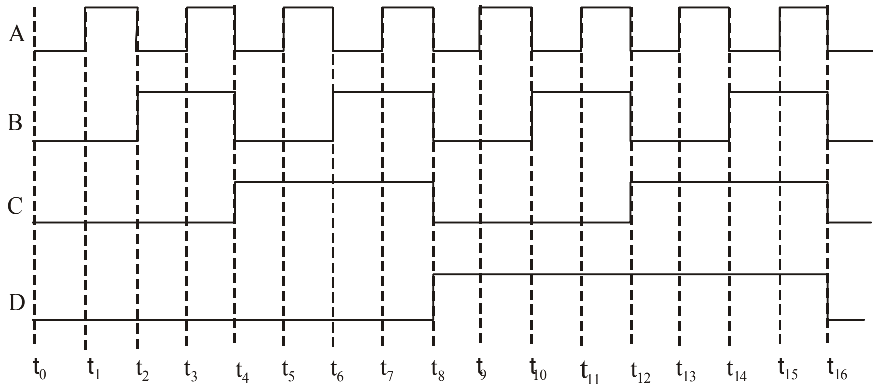


Fig. 9-32.

Sketch the waveforms for the encoder outputs.

Solution. Given : the waveforms shown in Fig. 9-32 applied to 74147 – a priority encoder.

Recall that in 74147, when two or more inputs are activated (LOW) at the same time, the output code correspond to the highest-numbered input. Further, since the 74147 has inverted outputs, therefore, output code will be complement of the BCD code corresponding to the highest-numbered active input.

Let us now examine the 74147 inputs, Given that,

$$A \rightarrow \overline{A}_7, B \rightarrow \overline{A}_4, C \rightarrow \overline{A}_2 \text{ , } D \rightarrow \overline{A}_1$$

In order to determine the encoder output code, consider the period during t_0 - t_1 time instances, and see that all the four inputs of the encoder $\overline{A}_7\ \overline{A}_4\ \overline{A}_2\ \overline{A}_1$ = $\overline{0}\overline{0}\overline{0}\overline{0}$. Therefore the highest-num bered active input is $\overline{A}_7$, and output code corresponding to the 7 is $\overline{0}\overline{1}\overline{1}\overline{1}$=1000. Hence the output waveforms for $Q_3\,Q_2\,Q_1\,Q_0$ are sketched as 1000 levels during $t_0 - t_1$ time instances. Next consider the period during t_1- t_2 time instances and see that the four inputs of the encoder are $\overline{A}_7\,\overline{A}_4\,\overline{A}_2\,\overline{A}_1$ = 1000. This time the highest-numbered active (LOW) input is $\overline{A}_4$. Therefore, the output code for 4 is $\overline{0}\overline{1}\overline{0}\overline{0}$=1011. Similarly during the period t_2 - t_3 time instances, the encoder inputs are $\overline{A}_7\,\overline{A}_4\,\overline{A}_2\,\overline{A}_1$ = 0010. The encoder again corresponds to the higher-numbered active (LOW) input i.e. $\overline{A}_7$. Therefore it produces an output code corresponding to BCD code for 7, *i.e* $\overline{0111}$=100. Similarly we can determine the highest-numbered active inputs during the time instances $t_3 - t_4$, $t_5 - t_6$, $t_7 - t_8$, $t_8 - t_9$, $t_{10} - t_{11}$, $t_{11} - t_{12}$, $t_{12} - t_{13}$, $t_{14} - t_{15}$, $t_{15} - t_{16}$ respectively and then can find the corresponding output code accordingly. The complete output waveforms for the encoder outputs $\overline{Q}_3\,\overline{Q}_2\,\overline{Q}_1\,\overline{Q}_0$ are as shown in Fig 9-33.

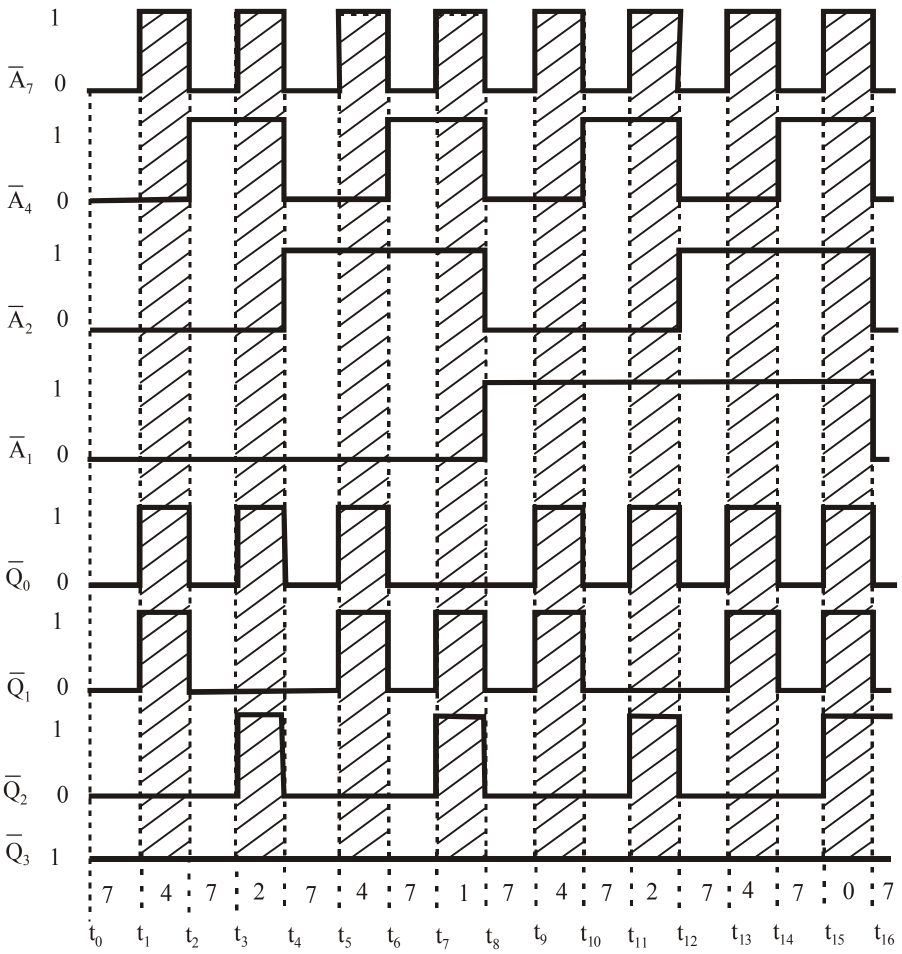


Fig. 9-33.

9-17. Switch Encoder Using 74147

Fig. 9-34 shows the use of 74147 as a switch encoder. The 10 switches (indicated as SW0, SW1 … SW9) might be the keyboard switches on a calculator representing digits 0 through 9. The switches are of the normally open type. Thus with all the switches open, the encoder inputs are all normally HIGH and the BCD output is 0000 (note the INVERTERS at the output of 74147). When a digit key is depressed, the circuit will produce the BCD code for that digit. Since the 74147 is a priority encoder, simultaneous key depressions will produce the BCD code for the higher-numbered key.

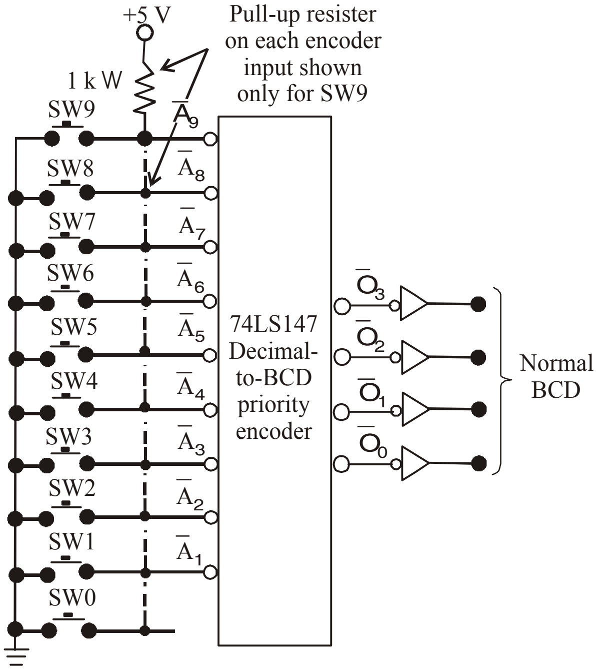


Fig. 9-34. 74LS147 used as a switch encoder

The switch encoder of Fig. 9-34 can be used whenever BCD data must be manually entered into a digital system such as an electronic calculator. Here the operator depresses several keyboard switches in succession to enter a decimal number. In a simple, basic calculator, the BCD code for each decimal digit is entered into a four-bit storage register. In other words, when the first key is depressed, the BCD code for that digit is sent to a four-bit FF register. When the second switch is depressed, the BCD code for that digit is sent to *another* four-bit FF register and so on. Thus a calculator that can handle eight digits will have eight four-bit registers to store the BCD codes for these digits. Each four-digit register drives a decoder/driver and a numerical display so that the eight-digit number can be displayed.

9-18. Multiplexers

These are also called *data selectors*. A digital multiplexer (in short MUX) is a logic circuit that accepts several digital data inputs and selects one of them at any given time to pass on to the output. The routing of the desired data input to the output is controlled by *SELECT inputs. A digital multiplexers is also called data selector.

* The SELECT inputs are usually refered to as ADDRESS inputs.

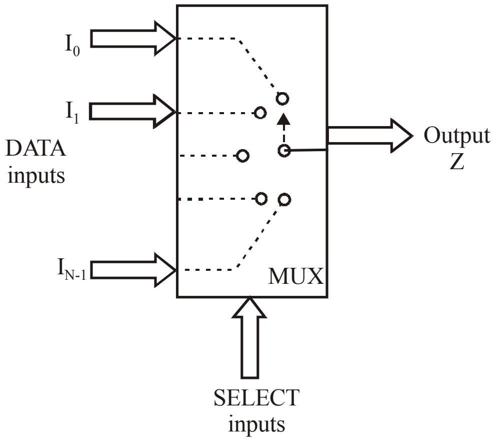


Fig. 9-35. Functional diagram of a digital multiplexer

Fig. 9-35 shows the functional diagram of a general digital multiplexer (MUX). As seen from the diagram, the inputs and outputs are drawn as wide arrows rather than lines. This indicates that they may actually be more than one signal line.

As a matter of fact, the multiplexer acts like a digitally controlled multiposition switch where a digital code applied to SELECT inputs controls which data inputs will be switched to the output. For example, output Z will equal data input I_0 for some particular SELECT input code, Z will equal data input I_1 for another particular SELECT input code and so on.

It is evident from the above discussion that a multiplexer selects 1 out of N input data sources and transmits the selected data to a single output channel. This is called multiplexing.

9-19. Two-input Multiplexer

Fig. 9-36 shows the logic circuitry for a two-input multiplexer (MUX) with data inputs I_0 and I_1 and SELECT input, S. The logic level applied to S input determines which AND gate is enabled so that its data input passes through the OR gate to output Z. Looking at it another way, the Boolean expression for the output is,

$$Z = I_0\ \overline{S} + I_1 S$$

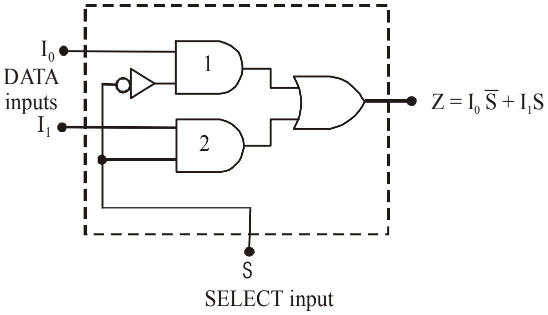
With S = 0, the Boolean expression becomes,

$$\begin{aligned} Z &= I_0.\overline{0} + I_1.\overline{0} \\ &= I_0.1 + I_1.0 \\ &= I_0 \end{aligned}$$

This indicates that Z will be identical to input signal I_0, which inturn can be a fixed logic level or a time-varying input signal. This condition is listed as the first lime in the truth table shown in Fig. 11(*b*). with S = 1, the Boolean expression becomes,

$$\begin{aligned} Z &= I_0.\overline{1} + I_1.1 \\ &= I_0.\ 0 + I_1.1 \\ &= I_1 \end{aligned}$$

This indicates that output Z will be identical to input signal I_1. This condition is listed as the second line in the truth table.



(*a*) Logic circuitry

S	*Output*
0	$Z = I_0$
1	$Z = I_1$

(*b*) Truth table

Fig. 9-36. A basic two-input multiplexer

Example 9-13. *The timing diagram shown in Fig.* 9-37 (*a*) *is applied to a two-input multiplexer shown in Fig.* 9-27 (*b*),

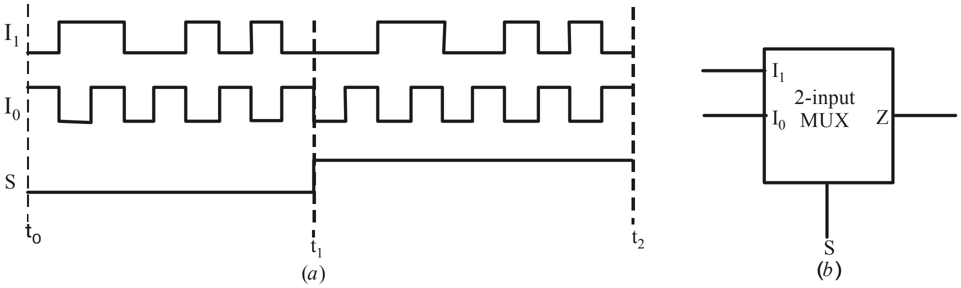


Fig. 9-37.

Sketch the output waveform Z.

Solution. Given : The timing diagram shown in Fig. 9-37 (*a*) applied to the inputs fo a 2-input multiplexer shown in Fig. 9-37 (*b*).

Recall the operation of a two-input multiplexer (refer to the truth table shown in Fig. 9-36 (*b*). If the select line input, S = 0, the output, $Z = I_0$, and if S = 1, $Z = I_1$. These conditions are applied to sketch the output waveform for the two-input multiplexer with the given set of waveforms in Fig 9.37. Notice that within the time period between $t_0 - t_1$, S = 0, therefore $Z = I_0$, and within the time period between $t_1 - t_2$, S = 1, therefore, $Z = I_1$. Therefore we can sketch the output waveform as shown in Fig. 9.38. Notice that within the time period, $t_0 - t_1$, the Z waveform is the same as I_0 waveform. Similarly, within the time period $t_1 - t_2$, the Z waveform is the same as I_1.

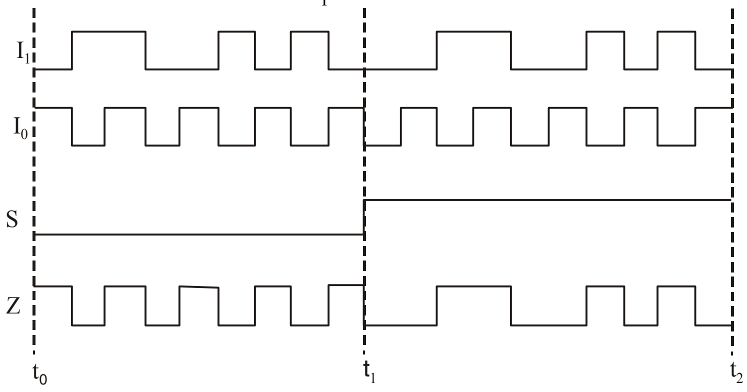


Fig. 9-38.

Example 9-14. *The circuit shown in Fig* 9-39 *uses three two-input multiplexers labelled as MUX* (1), *MUX* (2) *and MUX* (3). *The MUX* (1) *and* (2) *have a common select line, S_1. The MUX* (3) *has a separate select line, S_0.*

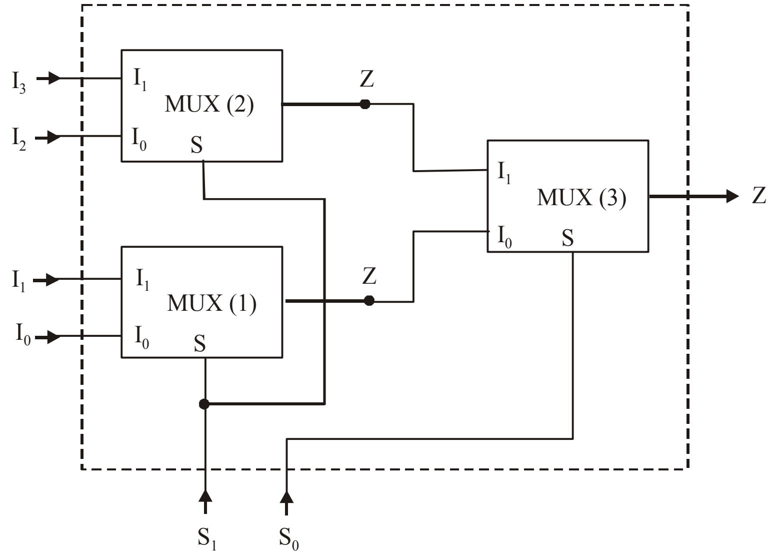


Fig. 9-39.

Determine the function performed by the circuit.

Solution. Recall the truth table of a 2-input multiplexer. (refer to Fig. 9-40 (*a*)). As seen, the output Z = I_0, if select input, S = 0, the output, Z = I, if the select input, S = 1.

S	*Z*
0	I_0
1	I_1

(*a*)

S_1	S_0	*MUX 1* *Z*	*MUX 2* *Z*	*MUX 3* *Z*
0	0	I_0	I_2	I_0
0	1	I_0	I_2	I_2
1	0	I_1	I_3	I_1
1	1	I_1	I_3	I_3

(*b*)

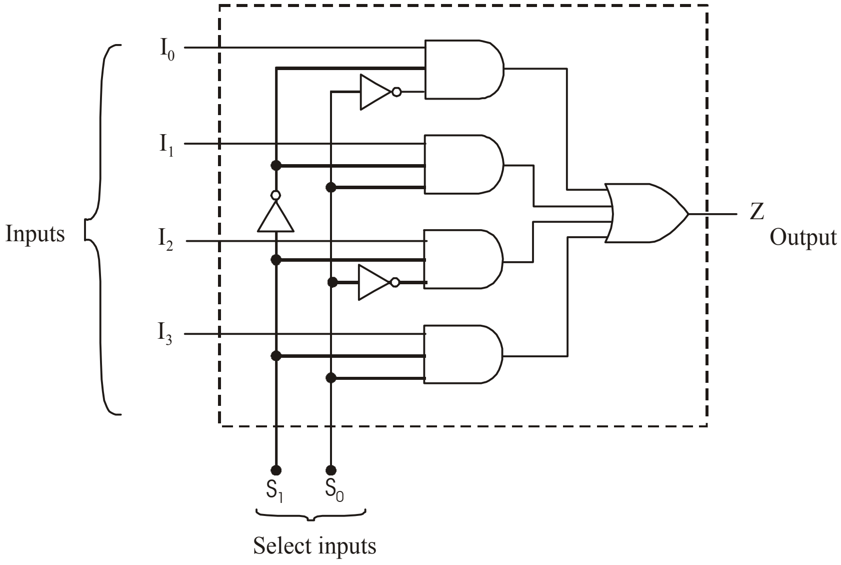
Fig. 9-40.

Using this truth table, we can find the output of all the three multiplexers discussed below. Note that when S_1 S_0 = 00, the MUX 1 output is I_0 and that of MUX 2 is I_2, and that of MUX 3 is I_0. Similarly, when S_1S_0 = 01, the MUX 3 output changes to I_2. When S_1S_0 = 10, the MUX 1 output is I_1, of MUX 2 is I_3 and that of MUX 3 is I_1. Finally when S_1S_0 = 11, the MUX 1output is still I_1 of MUX 2, is I_2 and of MUX 3 is I_3. The complete operation is indicated in truth table shown in Fig 9-40 (*b*).

It is evident from the truth table of Fig. 9-40 (*b*) that the circuit of Fig. 9-39 functions as a 4-input multiplexer.

9-20. Four-input Multiplexer

We have already discussed a two-input multiplexer in the last article. The same idea is used to form the four-input multiplexer. Fig. 9-41 (*a*) shows the logic circuitry and (*b*) the truth table of a four-input multiplexer.



(*a*) Logic circuitry

S_1	S_0	*Output*
0	0	$Z = 1_0$
0	1	$Z = 1_1$
1	0	$Z = 1_2$
1	1	$Z = 1_3$

(*b*) Truth table

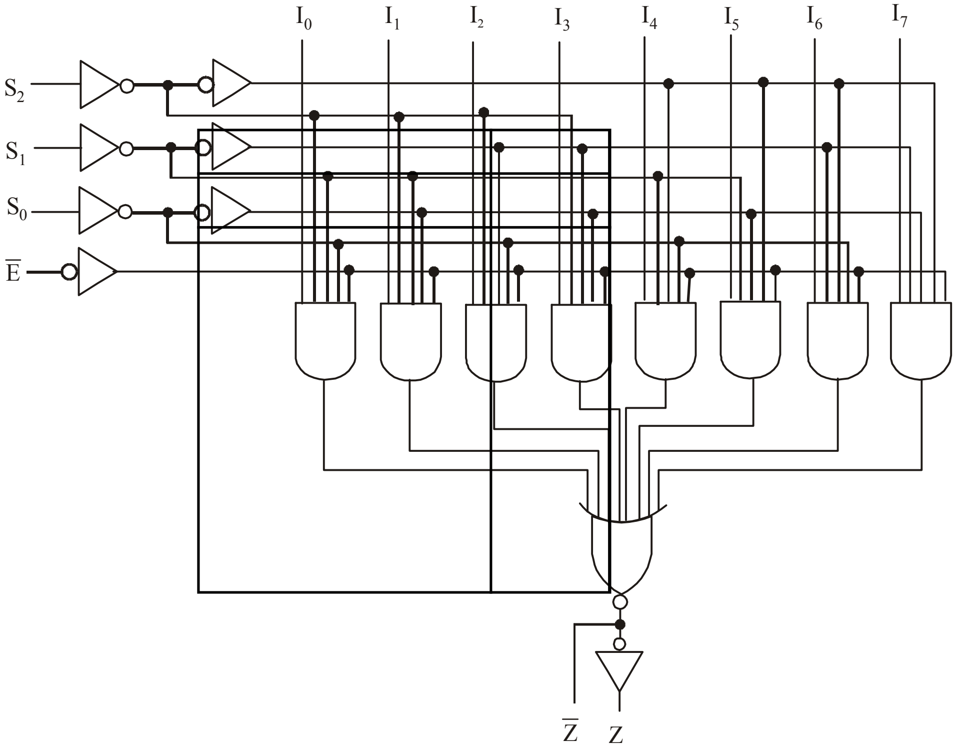
Fig. 9-41.

As seen from the logic circuitry, there are four inputs labelled as I_0, I_1, I_2 and I_3. These inputs are selectively transmitted to the output according to the four possible combinations of the S_1S_0 select inputs. Each data input is gated with different combinations of select input levels. For instance, I_0 is gated with $\bar{S}_1 \bar{S}_0$ so that I_0 will pass through its AND gate to output Z only when $S_1 = 0$ and $S_0 = 0$ (refer to the first line in the truth table). Similarly, I_1 is gated with $\bar{S}_1 S_0$ so that I_1 will pass through its AND gate to output Z only when $S_1 = 0$ and $S_0 = 1$ (refer to second line in the truth table). I_2 is gated with $\bar{S}_1 \bar{S}_0$ so that I_2 will pass through its AND gate to output Z only when $S_1 = 1$ and $S_0 = 0$ (refer to the third line in the truth table). Finally, I_3 is gated with $S_1 S_0$ so that I_3 will pass through its AND gate to output Z only when $S_1 = 1$ and $S_0 = 1$ (refer to the last line in the truth table).

9-21. Eight-input Multiplexer

Fig. 9-42 shows the logic circuitry for the eight–input multiplexer which is available as IC 74151. As seen from the diagram, the multiplexer has eight inputs labelled as I_0, I_1, I_2 … I_7 and an enable input

$\overline{E}$. It has three select lines labelled as S_2, S_1 and S_0 respectively. Further it provides both the normal and inverted outputs (refer to Z and $\overline{Z}$ in the figure).

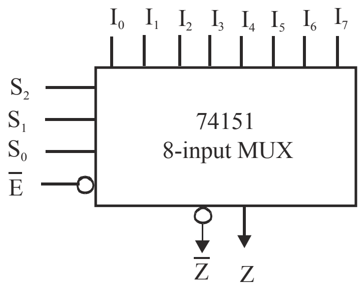


(*a*) Logic circuitry

Inputs				*Outputs*	
$\overline{E}$	S_2	S_1	S_0	$\overline{Z}$	Z
1	X	X	X	1	0
0	0	0	0	$\overline{I_0}$	I_0
0	0	0	1	$\overline{I_1}$	I_1
0	0	1	0	$\overline{I_2}$	I_2
0	0	1	1	$\overline{I_3}$	I_3
0	1	0	0	$\overline{I_4}$	I_4
0	1	0	1	$\overline{I_5}$	I_5
0	1	1	0	$\overline{I_6}$	I_6
0	1	1	1	$\overline{I_7}$	I_7

X = either 0 or 1

(*b*) Truth table



(*c*) Logic symbol

Fig. 9-42. Eight-input multiplexer (74151)

The operation of the multiplexer may be explained as follows. When $\overline{E}=0$, the select inputs S_2 S_1 S_0 will select one data input (from I_0, I_1, I_2....I_7) for passage to output Z. When $\overline{E}=1$, the multiplexer is disabled, so that Z = 0 regardless of the select input code. The whole operation is summarized in the truth table shown in Fig. 9-42 (*b*). The logic symbol for 74151 is shown in Fig. 9-42 (*c*).

Note. Two-, Four, eight- and sixteen-input multiplexers are readily available in the TTL and CMOS IC logic families. These basic ICS can be combined to form a multiplexer with larger number of inputs.

Example 9-15. F*ig.* 9-43 *shows a circuit that selects various clocking frequencies. Show how to replace the rotary switch with a* 74151 *multiplexer and give the control input conditions necessary to select the* 1 *kHz clock.*

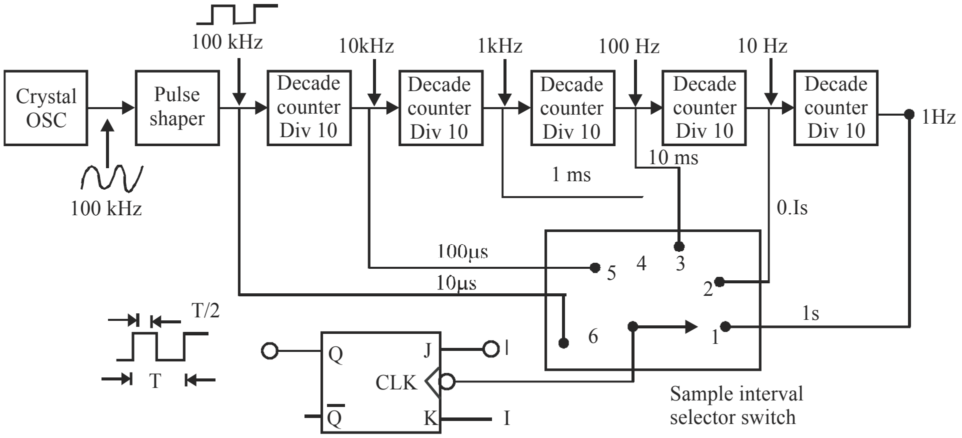


Fig. 9-43.

Solution. Notice the rotary switch has 6 selections. The 1 kHz signal is connected to point '4' on the rotary switch. Let us connect the signals as shown at the various inputs of the multiplexer. For Z = I_3, the SELECT inputs S_2 S_1 S_0 = 011 as shown in Fig. 9-44.

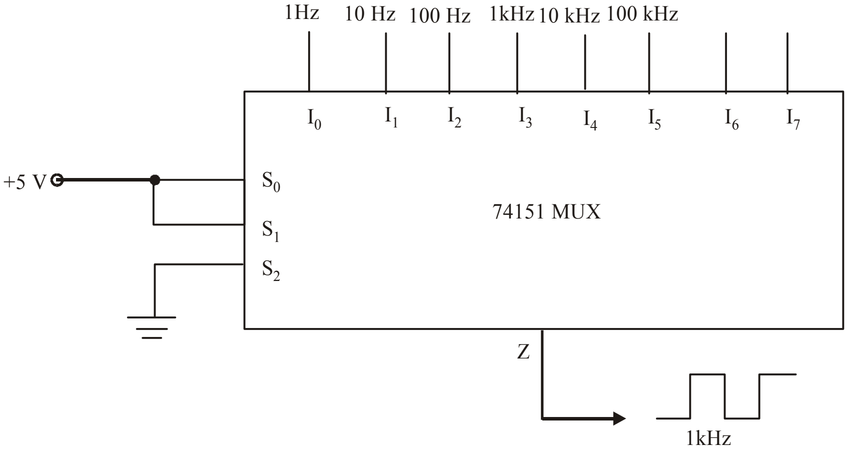


Fig. 9-44.

9-22. Sixteen-input Multiplexer

We have already mentioned in the last article that the basic ICs (such as two, four, eight-input multiplexers) can be combined to form multiplexers with larger number of inputs. We will illustrate this by combining two eight-input multiplexers (74151s) to form a sixteen-input multiplexer as shown in Fig 9-45.

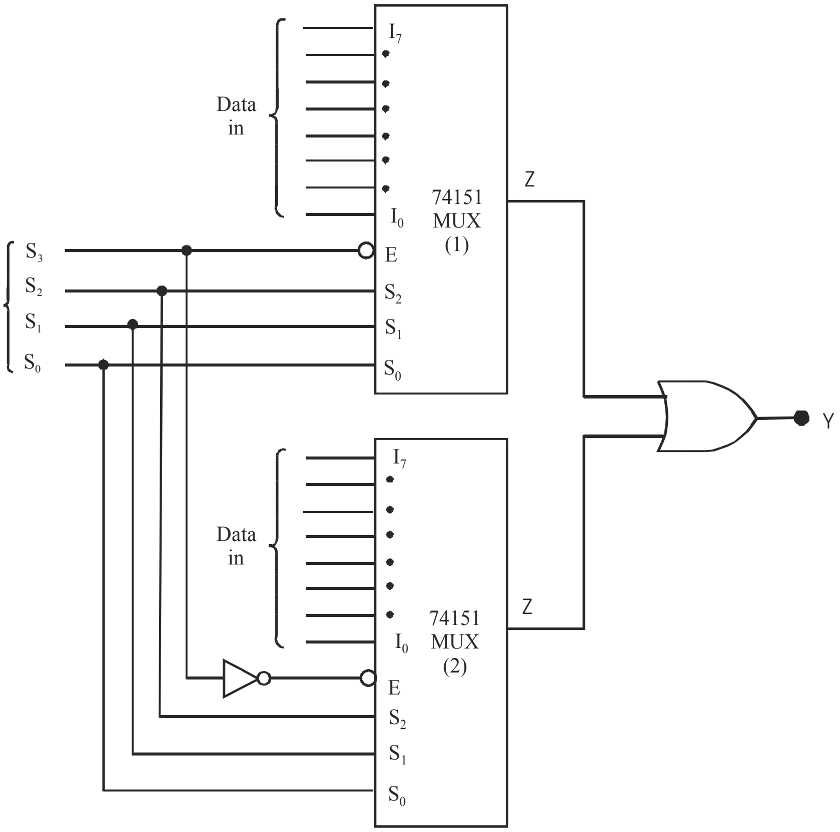


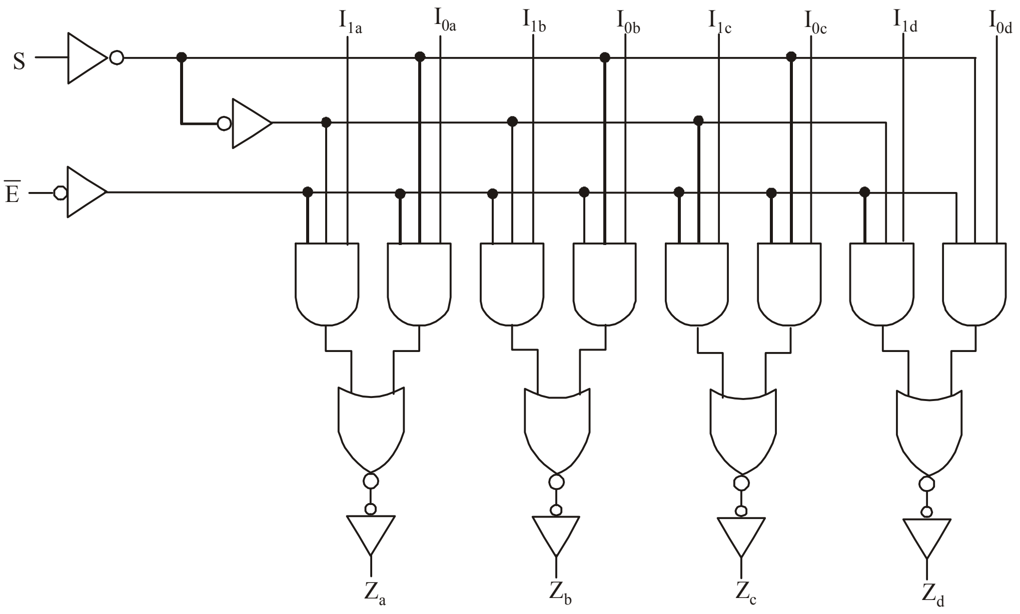
Fig. 9-45. Sixteen-input multiplexer by using two 74151s

As seen from the diagram, the circuit has a total of 16 data inputs, eight applied to each multiplexer. The two multiplexer outputs are combined in the OR gate to produce a single output Y. The circuit has four select inputs S_3 S_2 S_1 S_0 will select one of the sixteen inputs to pass through to Y.

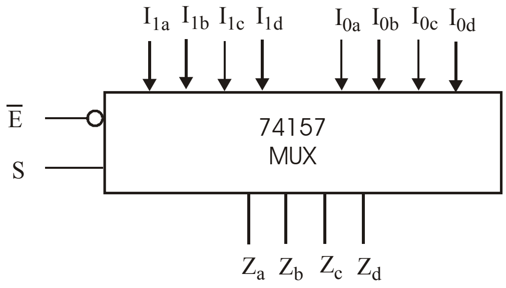
It may be carefully noted that S_3 input determines which multiplexer is enabled. When $S_3 = 0$, the multiplexers (1) is enabled the three select inputs S_2 S_1 S_0 inputs determine which of its data inputs (I_0, I_1, I_2...I_7) will appear at its output and pass through the OR gate to Y. When S_3=1, the multiplexer (2) is enabled, and the S_2 S_1 S_0 inputs select one of its data inputs (I_8, I_9, I_{10} … I_{15}) for passage to output Y.

9-23. Quad Two-input Multiplexer

Fig. 9-46 shows the logic diagram of 74157. This IC contains four two-input multiplexer like the one shown in Fig 9-31. The data inputs and output for the first two-input multiplexer are labelled as I_{1a} I_{oa} and Z_a. Similarly the data inputs and output for the second two-input multiplexer are labelled as I_{1b}, I_{0b} and Z_b. For the third two-input multiplexer, the data inputs and output are labelled as I_{1c}, I_{oc} and Z_c. And for the fourth two-input multiplexer, the data in inputs and output are labelled as I_{1d}, I_{od} and Z_d.



(*a*) Logic diagram


I_{1a} I_{1b} I_{1c} I_{1d}
I_{0a} I_{0b} I_{0c} I_{0d}
$\overline{E}$
S
74157
MUX
Z_a Z_b Z_c Z_d

(*b*) Logic symbol

Inputs		***Outputs***			
$\overline{E}$	S	Z_a	Z_b	Z_c	Z_d
H	X	L	L	L	L
L	L	I_{0a}	I_{0b}	I_{0c}	I_{0d}
L	H	I_{1a}	I_{1b}	I_{1c}	I_{1d}

(*c*) Truth table

Fig. 9-46. A quad two-input multiplexer (74157)

Fig. 9-46 (b) shows the logic symbol for the quad two-input multiplexer. The operation of the multiplexer may be explained as follows. When $\overline{E} = 0$ and S = 1, the Z outputs will follow the set of I_1, inputs, *i.e.*, $Z_a = I_{1a}$, $Z_b = I_{1b}$, $Z_c = I_{1c}$ and $Z_d = I_{1d}$. However, when $\overline{E} = 0$ and S = 0, the Z outputs will follow the set of I_0 inputs, *i.e.*, $Z_a = I_{0a}$, $Z_b = I_{0b}$, $Z_c = I_{0c}$ and $Z_d = I_{0d}$. Note that all of the outputs will be disabled (LOW) when $\overline{E} = 1$. The operation is summarized in the truth table shown in Fig. 9-46(*c*).

You can visualize the quad two-input multiplexer as being a simple two-input multiplexer, but one in which each input is four lines and the output is four lines. The four output lines switch back and forth between the two sets of four input lines under the control of select input (S). Because of this reason, the logic symbol of 74157 (refer to Fig. 9-46 (*b*)) indicates the two sets of four line inputs clearly.

9-24. Applications of Multiplexer

Although there are numerous applications of a multiplexer in digital systems but these may be broadly classified under the following six heads:

1. Data selection
2. Data routing
3. Operation sequencing
4. Parallel-to-serial conversion
5. Waveform generation
6. Logic function generation

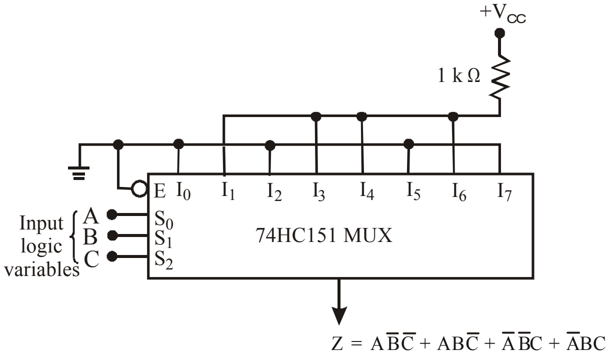
Now we shall the application of logic-function generation in detail.

9-25. Logic Function Generation

As a matter of fact, multiplexers can be used to implement logic functions directly from a truth table without the need for simplification. When a multiplexer is used for this purpose, the select inputs are used as the logic variables, and each data input is connected permanently HIGH or LOW as necessary to satisfy the truth table.

Inputs			*Output*	
C	***B***	***A***	***Z***	
0	0	0	0	
0	0	1	1	$\leftarrow A\overline{B}\overline{C}$
0	1	0	0	
0	1	1	1	$\leftarrow AB\overline{C}$
1	0	0	1	$\leftarrow \overline{A}\overline{B}C$
1	0	1	0	
1	1	0	1	$\leftarrow \overline{A}BC$
1	1	1	0	

(*a*) Truth table



(*b*) Circuit diagram

Fig. 9-47. Illustrating logic circuit generation using multiplexer

Let us suppose we have to implement a logic function from the truth table shown in Fig. 9-47 (*a*). As seen from this table, the output Z is 1 for $A\overline{B}\overline{C}$, $AB\overline{C}$, $\overline{A}\,\overline{B}C$ and $\overline{A}BC$. The output is 0 for all the other input combinations. Fig. 9-47 (*b*) shows how an eight-input multiplexer can be used to implement the desired logic function.

As seen from this figure, the input variables A, B and C are connected to S_0, S_1 and S_2 (*i.e.*, SELECT input lines) respectively. The logic levels (*i.e.*, HIGH or LOW) on S_0, S_1 and S_2 will determine which input appears at output Z. Recall that if $S_2\ S_1\ S_0 = 000$, the output $Z = I_0$. Similarly if $S_2\ S_1\ S_0 = 001$, the output $Z = I_1$, and so on.

Now according to the truth table, Z is supposed to be LOW when CBA = 000. Thus multiplexer input I_0 should be connected LOW. Like wise Z is supposed to be LOW for CBA = 010, 101 and 111. As a result, multiplexer inputs I_2, I_5 and I_7 should also be connected LOW. The other combinations of CBA must produce Z = 1. So the multiplexer inputs I_1, I_3 I_4 and I_6 are connected permanently HIGH.

It is evident from the above discussion that any three-variable truth table can be implemented with this eight-input multiplexer. Sometimes this method of implementation is often more efficient than using separate logic gates.

Example 9-16. *Use the* 74151 8-*to*-1 *multiplexer shown in Fig.* 9-48 *and any NOT gates, if necessary, to implement the following Boolean expression:*

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

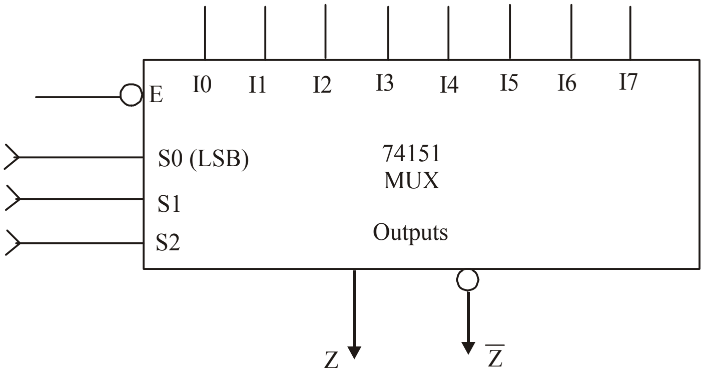


Fig. 9-48.

Solution. (*a*) Given, the logic function $Z = A\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + \bar{A}B\bar{C} + ABC$

As seen from the given function, the output Z is 1 for $A\bar{B}\bar{C}$, $\bar{A}BC$, $A\bar{B}C$, $\bar{A}B\bar{C}$ and ABC. The output Z is zero for all the other input combinations. Fig. 9-49 shows how an eight-input multiplexer can be used to implement the desired function.

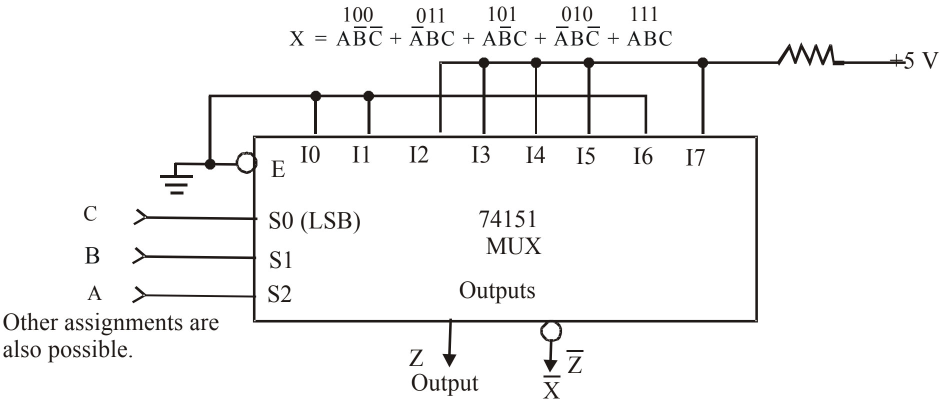


Fig. 9-49.

As seen from this figure, the input variable A, B and C are connected to S_0, S_1 and S_2 (i.e., select input lines) respectively. Other assignments are also possible. The logic levels (i.e., HIGH or LOW) on S_0, S_1 and S_2 will determine which input appears at Z. Recall that if $S_2\ S_1\ S_0 = 000$, the output $Z = I_0$. Similarly if $S_2\ S_1\ S_0 = 001$, the output $Z = I_1$ and so on.

Now according to the truth table for a 8-input, MUX, When ABC = 000, the output $Z = I_0 = 0$. This MUX input I_0 should be connected LOW. Likewise Z is supposed to be LOW for ABC = 001 and 110. As a result, MUX inputs I_1 and I_6 are also connected LOW. The other combinations of A, B and C must produce Z = 1. So the MUX inputs I_2, I_3, I_4, I_5 and I_7 are connected HIGH.

Example 9-17. *Show how a* 74151 *can be used to generate the logic function Z = AB + BC + AC.*

Solution. First of all, rewrite the given logic function so that we can include all the missing variables in each sum term, *i.e.*,

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

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

Inputs			***Output***	
C	***B***	***A***	***Z***	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	← $\bar{C}BA$
1	0	0	0	
1	0	1	0	
1	1	0	1	← $CB\bar{A}$
1	1	1	1	← CBA

Fig. 9-50.

Now construct a truth table and write output Z = 1, corresponding to the above Boolean terms. The resulting truth table is as shown in Fig 9-50. In order to implement this logic function using 74151 multiplexer, connect I_3, I_6 and I_7 to HIGH and all other inputs to LOW as shown in Fig. 9-51.

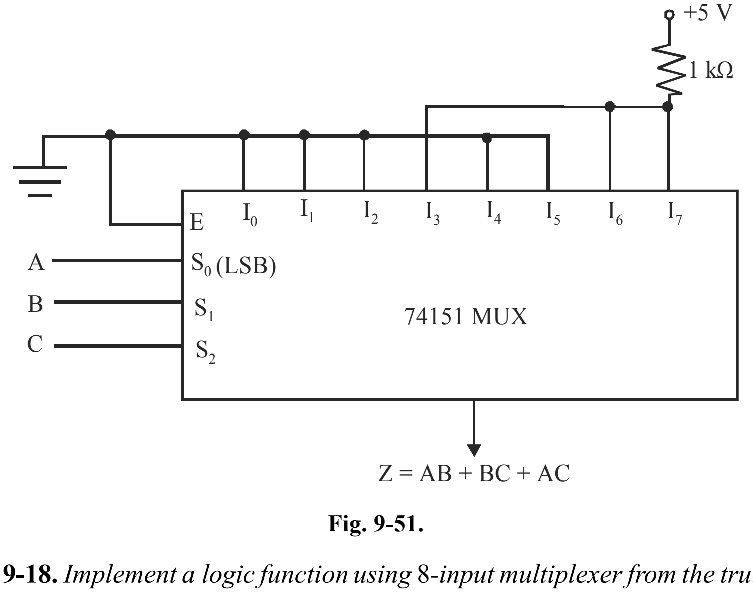


Fig. 9-51.

Example 9-18. *Implement a logic function using* 8-*input multiplexer from the truth table shown in Fig* 9.52.

Inputs			*Output*
C	*B*	*A*	*Z*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Fig. 9-52.

Verify that the implementation of output function using 74 151 *multiplexer is more efficient than using the basic logic gates* (*i.e., ANR-OR-INVERT gates*) *or NAND gates.*

Solution. According to the given truth table, Z is supposed to be LOW when CBA = 000. Thus multiplexer input I_0 should be connected LOW. Likewise, Z is supposed to be LOW for CBA = 011, 100, 101 and 110. This means 74HC151 multiplexer inputs I_3, I_4, I_5 and I_6 should also be connected LOW. The other sets of CBA conditions produce Z = 1 as indicated in the given truth table. So the multiplexer inputs I_1, I_2 and I_7 are connected permanently HIGH. Fig. 9-53 shows the implementation of the output logic function (Z) using 74 151 multiplexer.

Notice that the sum-of-products expression from the truth table shown in Fig. 9-52, is,

$$Z = A\overline{B}\overline{C} + \overline{A}B\overline{C} + ABC$$

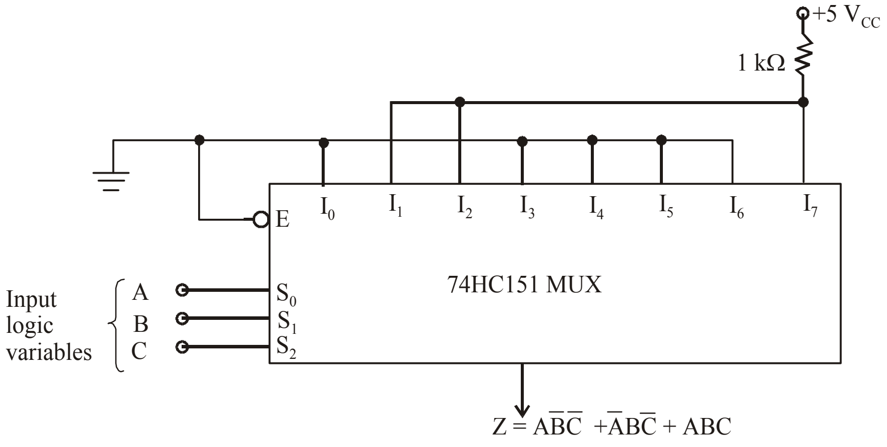
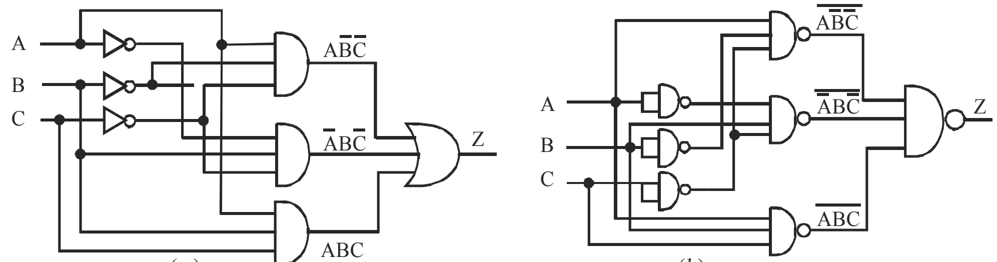


Fig. 9-53.

This function *cannot* be simplified either algebraically or by K-mapping. If we implement this function using AND-OR-INVERT gates (refer to Fig. 9-54 (*a*)) it will require 3 INVERTERS, 3 AND gates and 1 OR gate. This means a physical realization of this logic function will need a total of three ICs (*i.e.* 74 04, 74 08, 74 32). Even if we decide to realize this logic function using only NAND gates, still it will need seven 74 HC00 gates as shown in Fig. 9-54 (*b*), (*i.e.*, two ICs because one IC has only 4 NAND gates in it).



(*a*) (*b*)

Fig. 9-54.

It is evident from the above discussion that the method of implementation of a logic function using multiplexer (*i.e.*, just one IC) is more efficient than using separate logic gates.

9-26. A More Efficient Method for Generating a Logic Function

We have already discussed in the last article that an eight-input multiplexer (74151) can be used to implement a three-variable logic function. Now we will study as how an eight-input multiplexer can be used to generate a four variable logic function.

Consider for example, a logic expression,

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

This expression can also be rewritten as

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

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

Now construct a truth table with four variables, A, B, C and D as shown in Fig. 9-55. Enter Z = 1, corresponding to the terms.

Inputs				***Output***		
D	***C***	***B***	***A***	***Z***		
0	0	0	0	0		
0	0	0	1	0		
0	0	1	0	1	←	$\bar{D}\bar{C}B\bar{A}$
0	0	1	1	0		
0	1	0	0	1	←	$\bar{D}C\bar{B}\bar{A}$
0	1	0	1	0		
0	1	1	0	0		
0	1	1	1	0		
1	0	0	0	0		
1	0	0	1	1	←	$D\bar{C}\bar{B}A$
1	0	1	0	1	←	$D\bar{C}B\bar{A}$
1	0	1	1	0		
1	1	0	0	0		
1	1	0	1	0		
1	1	1	0	0		
1	1	1	1	0		

Fig. 9-55.

$D\bar{C}B\bar{A}$ (*i.e.* 1010), $\bar{D}\bar{C}B\bar{A}$ (*i.e.* 0010), $D\bar{C}\bar{B}A$ (*i.e.* 1001) and $\bar{D}C\bar{B}\bar{A}$ (*i.e.* 0100).

For other input combinations, Z = 0. Thus the truth table (or even the Boolean expression given in equation (i)) type of a logic function can be implemented by an eight-input multiplexer connected as shown in Fig 9-56.

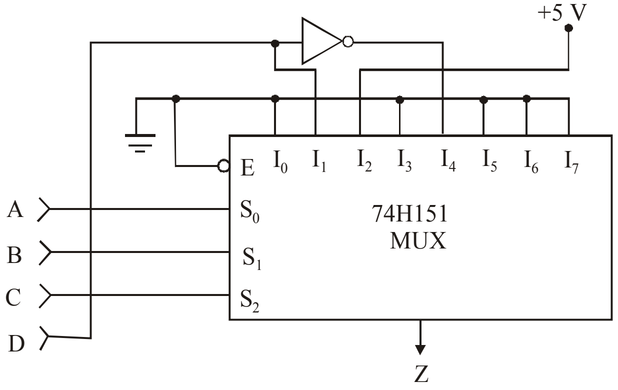


Fig. 9-56.

As seen from this circuit, three of the logic variables A, B and C are connected to the SELECT inputs (S_0, S_1, and S_2). The fourth variable D and its inverse $\bar{D}$ are connected to selected data inputs of the multiplexer as required by the desired logic function. The other multiplexer data inputs are connected to LOW or a HIGH as required by the function. For instance, when CBA = 000, Z = I_0 = 0 (refer to the first line of the truth table). Next when CBA = 001, Z = I_1. We connected I_1 to the variable D because Z = 0 when DCBA = 0001 and Z = 1 when DCBA = 1001. So if D = 0, Z = 0 and if D = 1, Z = 1.

Example 9-19. *Use the* 74151 8-*to*-1 *multiplexer and any NOT gates, if necesssary, to implement the Boolean expression* :

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

Solution. *Given* : A 74151 8-to-1 multiplexer.

Following the same approach as discussed in the last Article, you can make a truth table as shown in table 9-5

Table 9-5

A	*B*	*C*	*D*	*Y*	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	$\longrightarrow \bar{A}B\bar{C}D$
0	1	1	0	0	
0	1	1	1	1	$\longrightarrow \bar{A}BCD$
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	$\longrightarrow A\bar{B}C\bar{D}$
1	0	1	1	0	
1	1	0	0	1	$\longrightarrow AB\bar{C}\bar{D}$
1	1	1	0	0	
1	1	1	1	1	$\longrightarrow ABCD$

Notice that we have entered Y = 1 for $A\bar{B}C\bar{D}$ (i.e. 1010), $AB\bar{C}\bar{D}$ (i.e. 1100), $\bar{A}BCD$ and ABCD. The learn for entering 1 for the last two terms is that BCD can be rewritten as $\left(A+\bar{A}\right)BCD = ABCD + \bar{A}BCD$. For all other input combinations Z = 0.

Thus the truth table show in Fig. 9-56 can be implemented using 74151 as shown in Fig. 9-57.

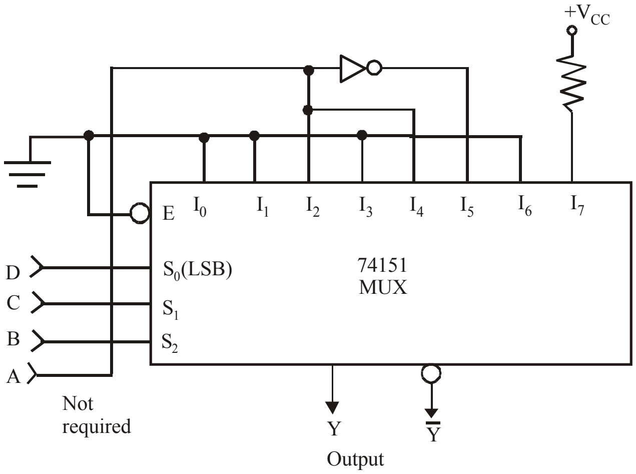


Fig. 9-57.

As seen from this circuit, three of the logic variables, B, C and D are connected to SELECT inputs (S_0, S_1 and S_2). Other combinations are possible. The fourth variable A and its inverse $\overline{A}$ are connected to selected data inputs of the multiplexer as required by the desired logic function. The other multiplexer data inputs are connected to LOW or a HIGH as required by the function. For instance, when BCD = 000, Y = I_0 = 0. Next when BCD = 001, Y = I_1 = 0. When BCD = 010 Y = I_2. We connected I_2 to the variable A because, Y = 0 when ABCD = 0010 and Y = 1 when ABCD = 1010. Next when BCD = 011, Y = I_3 = 0. Next when BCD = 100, Z = I_4. We have connected I_4 to the variable A because Y = 0 when ABCD = 0100 and Y = 1 when ABCD = 1100. Next when BCD = 101, Z = I_5. We have connected I_4 to the variable A through an INVERTER (or NOT gate); because Y = 1 when ABCD = 0101 and Y = 0 when ABCD = 1101. Next when BCD = 110, Z = I_6 = 0. Finally BCD = 111, Z = I_7 =1. We have connected I_7 to HIGH because Z is 1 when ABCD = 0111 as well as ABCD = 1111.

Example 9-20. *Implement the following function using 8 : 1 MUX :*

$$f(A, B, C, D) = \Sigma m\ (0, 2, 4, 6, 8, 10, 12, 14)$$

(*Nagpur University, 2004*)

Solution. *Given* : The function

$$f(A, B, C, D) = \Sigma m\ (0, 2, 4, 6, 8, 10, 12, 14)$$

The truth table of the function is shown in Fig. 9-58.

List the input of the multiplexer and under then list all the minterms in two rows. The first row lists all those minterms where *A* is complement, and second row all the minterms with *A* uncomplemented, as shown in Fig. 9.59(a). Circle all the minterms of the function and inpect each column separately as follows:

1. If the two minterms in a column are not circled apply 0 to the corresponding multiplexer input.
2. If the two minterms are circled, apply 1 to the corresponding multiplexer input.
3. If the bottom minterms is circle and top is not circled, apply *A* to the corresponding multiplexer input.
4. If the top minterm is circled and the bottom is not circled, apply *A'* to the corresponding multiplexer input.

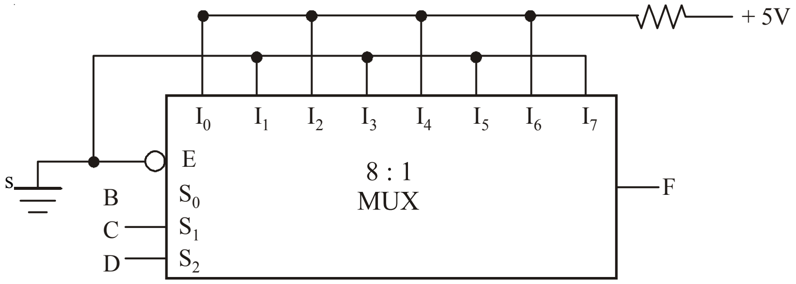
Implement the function is shown in Fig. 9-59 (b)

Inputs				*Outputs*
A	***B***	***C***	***D***	***f***
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Fig. 9-58.

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

(a) Implement Table



(b) Circuit Diagram

Fig. 9-59.

Example 9-21. *How would you realise the function* $ABCD + \overline{A}BC + BC\overline{D}$ *using an 8-to-1 multilpexer?* (*VTU, Jan. /Feb 2005*)

Solution. *Given* : The function

$$ABCD + \overline{A}BC + BC\overline{D}$$

The given function is not in the form of standard SOP. So we convert the function in standard form taking term one by one. The first term contain all variable. The second term has missing variable D or $\overline{D}$. So we multiply this term by $D + \overline{D}$. The third term has a missing variable A and $\overline{A}$. So we multiply this term by $A + \overline{A}$. Thus.

$$\begin{aligned} ABCD + \overline{A}BC + BC\overline{D} &= ABCD + \overline{A}BC\,(D + \overline{D}) + BC\overline{D}\,(A + \overline{A}) \\ &= ABCD + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D} + \overline{A}BC\overline{D} \\ &= ABCD + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D} \\ &= m_0 + m_7 + m_6 + m_{14} \end{aligned}$$

We implement the function in the form of table according to the step given in Example 9-20. The implement table of the function is shown in Fig. 9-59.

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

Fig. 9-60.

From the implement table we find that 0 is applied to the I_1, I_2, I_3, I_4 and I_5 of the multiplexer. +5V is applied to the I_6 and $\overline{A}$ to I_0 and I_7 of the multiplexer as shown in Fig. 9-61.

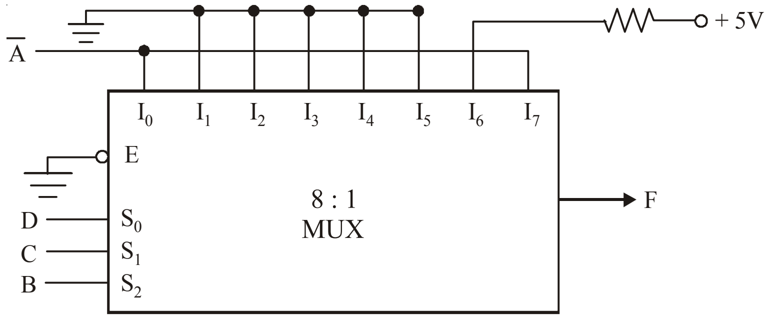


Fig. 9-61.

Example 9-22. *Give a 4-to-1 MUX implementation of the three variable function*

$f = \Sigma m\,(1, 4, 5, 7)$ (*VTU, Jan. /Feb. 2004*)

Solution. Given : The function

$$f = \Sigma m\,(1, 4, 5, 7)$$

The truth table of the function is shown in Fig. 9-62 from the function there are three variable *A*, *B* and *C*.

Inputs			*Outputs*
A	*B*	*C*	*f*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Fig. 9-62.

We implement the function in the form of table according to the step given in Example 9-20. Now we find the implementation table of the function. The table is shown in Fig. 9-63.

	I_0	I_1	I_2	I_3
$\overline{A}$	0	①	2	3
A	④	⑤	6	⑦
	A	1	0	A

Fig. 9-63.

From the implement table, we implement the function in 4 : 1 multiplexer. The input is 1 corresponding to I_1 and A corresponding to input I_0 and I_3. The input is 0 corresponding to input I_2. The multiplexer is shown in Fig. 9-64.

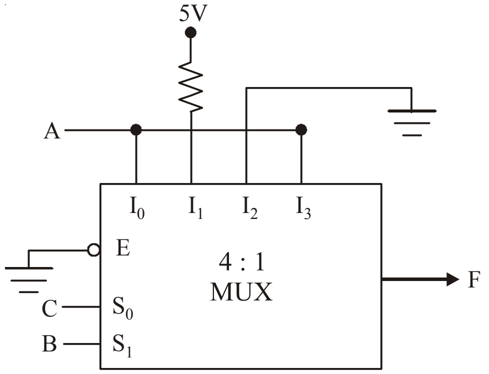


Fig. 9-64.

Example 9-23. *Implement the following function using a multiplexer*

$$F(A, B, C, D) = \Sigma(0, 2, 3, 7, 9)$$

(*RGTU., Dec. 2009*)

Solution. Given : The function,

$$F(A, B, C, D) = \Sigma(0, 2, 3, 7, 9)$$

We construct the truth tables shown in Fig. 9-65.

Decimal	*Inputs*				*Outputs*	
Number	*A*	*B*	*C*	*D*	*F*	
0	0	0	0	0	1	→ $\bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	0	
2	0	0	1	0	1	→ $\bar{A}\bar{B}C\bar{D}$
3	0	0	1	1	1	→ $\bar{A}\bar{B}CD$
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	0	
7	0	1	1	1	1	→ $\bar{A}BCD$
8	1	0	0	0	0	
9	1	0	0	1	1	→ $A\bar{B}\bar{C}D$
10	1	0	1	0	0	
11	1	0	1	1	0	
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	0	

Fig. 9-65.

Notice that we have entered Y = 1 for $\bar{A}\bar{B}\bar{C}\bar{D}$, $\bar{A}\bar{B}C\bar{D}$, $\bar{A}\bar{B}CD$, $\bar{A}BCD$, $A\bar{B}\bar{C}D$. For all other input combinations Z = 0.

As seen from this circuit, three of the logic variables, *B*, *C* and *D* are connected to SELECT inputs (S_0, S_1 and S_2). Other combinations are possible. The fourth variable *A* and its inverse $\bar{A}$ are connected to selected data inputs of the multiplexer as required by the desired logic function. The other multiplexer data inputs are connected to LOW or a HIGH as required by the function. For instance, when *BCD* = 000, Y = Z = I_0. We have connected I_0 to the variable *A* through an INVERTER (or NOT gate); because Y = 1 when *ABCD* = 0000 and Y = 0 when *ABCD* = 1000. Next when *BCD* = 001, Y = 1, Y = I_1. We have connected I_1 to the variable *A* because Y = 0 when *ABCD* = 0001 and Y = 1 when *ABCD* = 1001, when *BCD* = 010, Y = Z = I_2. We have connected I_2 to the variable *A* through an INVERTER (or NOT gate); because Y = 1 when *ABCD* = 0010 and Y = 0 when *ABCD* = 1010. Next when *BCD* = 011, I_3. We have connected I_3 to the variable *A* through an INVERTER (or NOT gate); because Y = 1 when *ABCD* = 0011 and Y = 0 when *ABCD* = 1011. Next when *BCD* = 100, Y = I_4 = 0. Next when *BCD* = 101, Y = I_5 = 0. Next when *BCD* = 110, Y = I_6 = 0. Next when *BCD* = 111, Y = Z = I_7. We have connected I_2 to the variable *A* through an INVERTER (or NOT gate); because Y = 1 when *ABCD* = 0010 and Y = 0 when *ABCD* = 1010.

The truth table can be implement using 7451 multiplexer as shown in Fig. 9-66.

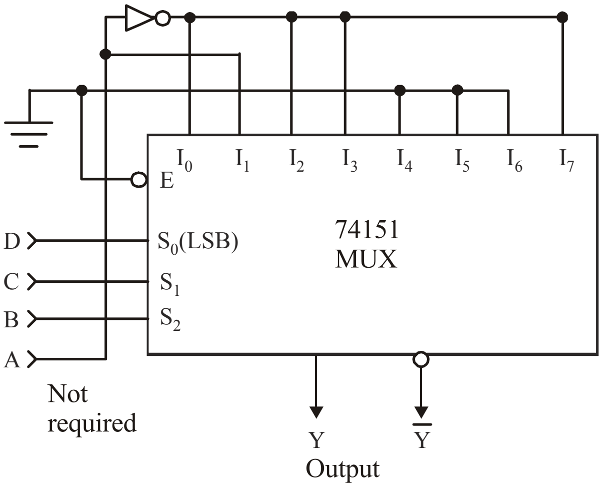


Fig. 9-66.

Example 9-24. *Implement the following function using a multiplexer.*

$$F(A, B, C, D) = \Sigma(1, 5, 6, 7, 8)$$

(RGTU., June 2010)

Solution. Given : The function,

$$F(A, B, C, D) = \Sigma(1, 5, 6, 7, 8)$$

We construct the truth tables shown in Fig. 9-67.

Decimal Number	*Inputs* *A*	*B*	*C*	*D*	*Outputs* *F*	
0	0	0	0	0	0	
1	0	0	0	1	1	→ $\overline{A}\overline{B}\overline{C}D$
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	1	→ $\overline{A}B\overline{C}D$
6	0	1	1	0	1	→ $\overline{A}BC\overline{D}$
7	0	1	1	1	1	→ $\overline{A}BCD$
8	1	0	0	0	1	→ $A\overline{B}\overline{C}\overline{D}$
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	0	
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	0	

Fig. 9-67.

Notice that we have entered Y = 1 for $\overline{A}\overline{B}\overline{C}\overline{D}$, $\overline{A}B\overline{C}D$, $\overline{A}BC\overline{D}$, $\overline{A}BCD$ and $A\overline{B}\overline{C}\overline{D}$. For all other input combinations Z = 0.

As seen from this circuit, three of the logic variables, B, C and D are connected to SELECT inputs (S_0, S_1 and S_2). Other combinations are possible. The fourth variable A and its inverse $\overline{A}$ are connected to selected data inputs of the multiplexer as required by the desired logic function. The other multiplexer data inputs are connected to LOW or a HIGH as required by the function. For instance, when BCD = 000, Y = Z = I_0. We have connected I_1 to the variable A because Y = 0 when $ABCD$ = 0001 and Y = 1 when $ABCD$ = 1001. Next when BCD = 001, Y = I_1. We have connected I_2 to the variable A through an INVERTER (or NOT gate); because Y = 1 when $ABCD$ = 0010 and Y = 0 when $ABCD$ = 1010. Next when BCD = 010, Y = Z = I_2 = 0. Next when BCD = 011, Z = I_3 = 0. Next when BCD = 100, Y = I_4 = 0. Next when BCD = 101, Y = I_5 = 0. We have connected I_2 to the variable A through an INVERTER (or NOT gate); because Y = 1 when $ABCD$ = 0010 and Y = 0 when $ABCD$ = 1010. Next when BCD = 110, Y = I_6 = 0. We have connected I_2 to the variable A through an INVERTER (or NOT gate); because Y = 1 when $ABCD$ = 0010 and Y = 0 when $ABCD$ = 1010. Next whe BCD = 111, Y = Z = I_7. We have connected I_2 to the variable A through an INVERTER (or NOT gate); because Y = 1 when $ABCD$ = 0010 and Y = 0 when $ABCD$ = 1010.

The truth table can be implement using 7451 multiplexer as shown in Fig. 9-68.

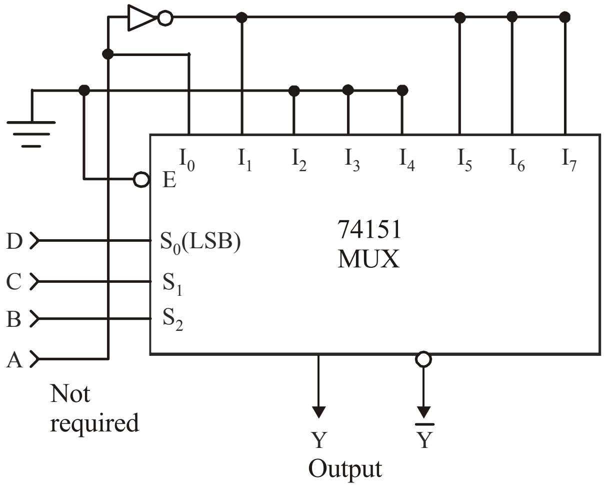


Fig. 9-68.

Example 9-25. *Implement the function using multiplexer :*

$$F\,(A,\,B,\,C) \;=\; A + AB$$

(WBUT., 2011-12)

Solution. First of all rewrite the logic function so that we can include all the missing variables in each terms, i.e.,

$$F = (B + \overline{B})\,(C + \overline{C}) + AB\,(C + \overline{C})$$

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

Inputs			*Outputs*	
A	***B***	***C***	***f***	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	→ $A\overline{B}\,\overline{C}$
1	0	1	1	→ $A\overline{B}C$
1	1	0	1	→ $AB\overline{C}$
1	1	1	1	→ ABC

Fig. 9-69.

Now construct the truth table and write output Z = 1, corresponding to the above Boolean terms. The resulting truth table is as shown in Fig. 9-69. In order to implement the logic function using 74151 multiplexer, connect I_7, I_6, I_5 and I_4 to HIGH and all other inputs to LOW as shown in Fig. 9-70.

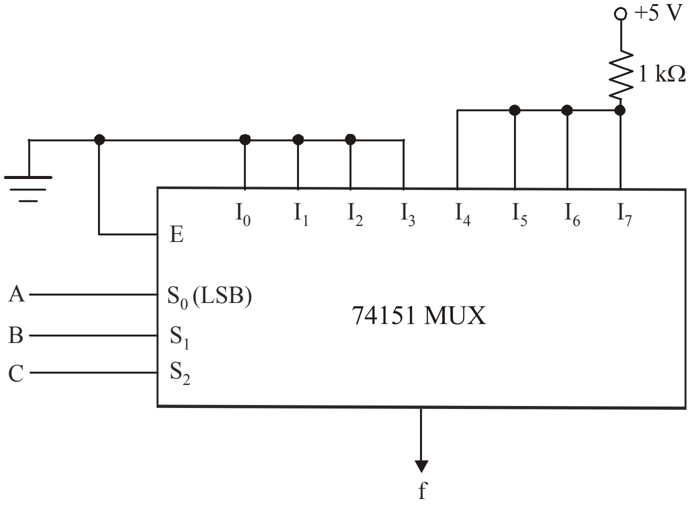


Fig. 9-70.

Example 9-26. *Implement the function F(A, B, C, D) = Σ (0, 1, 3, 4, 8, 9, 15) using multiplexers.*
(RGTU., June 2009, 2011, GTU., May 2011)

Solution. Given : The function,

$$F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$$

We construct the truth tables shown in Fig. 9.71.

Decimal		*Inputs*			*Outputs*	
Number	*A*	*B*	*C*	*D*	*F*	
0	0	0	0	0	1	→ $\bar{A}\bar{B}\bar{C}\bar{D}$
1	0	0	0	1	1	→ $\bar{A}\bar{B}\bar{C}D$
2	0	0	1	0	0	
3	0	0	1	1	1	→ $\bar{A}\bar{B}CD$
4	0	1	0	0	1	→ $\bar{A}B\bar{C}\bar{D}$
5	0	1	0	1	0	
6	0	1	1	0	0	
7	0	1	1	1	0	
8	1	0	0	0	1	→ $A\bar{B}\bar{C}\bar{D}$
9	1	0	0	1	1	→ $A\bar{B}\bar{C}D$
10	1	0	1	0	0	
11	1	0	1	1	0	
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	1	→ $ABCD$

Fig. 9-71.

Notice that we have entered Y = 1 for $\overline{A}\overline{B}\overline{C}\overline{D}$, $\overline{A}\overline{B}\overline{C}D$, $A\overline{B}\overline{C}\overline{D}$, $A\overline{B}\overline{C}D$ and $ABCD$. For all other input combinations Z = 0.

As seen from this circuit, three of the logic variables, *B*, *C* and *D* are connected to SELECT inputs (S_0, S_1 and S_2). Other combinations are possible. The fourth variable *A* and its inverse $\overline{A}$ are connected to selected data inputs of the multiplexer as required by the desired logic function. The other multiplexer data inputs are connected to LOW or a HIGH as required by the function. For instance, when BCD = 000, Y = Z = I_0 = 1. We have connected I_0 to HIGH because Z is 1 when $ABCD$ = 0000 as well as $ABCD$ = 1000. Next when BCD = 001, Y = I_1 = 1. We have connected I_0 to HIGH because Z is 1 when $ABCD$ = 0001 as well as $ABCD$ = 1001. When BCD = 010, Y = I_2 = 0. Next when BCD = 011, Z = I_3. We have connected I_3 to the variable *A* through an INVERTER (or NOT gate); because Y = 1 when $ABCD$ = 0011 and Y = 0 when $ABCD$ = 1011. Next when BCD = 100, Y = I_4. We have connected I_4 to the variable *A* through an INVERTER (or NOT gate); because Y = 1 when $ABCD$ = 0100 and Y = when $ABCD$ = 1100. Next when BCD = 101, Y = I_5 = 0. Next when BCD = 110, Y = I_6 = 0. Next when BCD = 111, Z = I_7. We have connected I_7 to the variable *A* because Y = 0 when $ABCD$ = 0111 and Y = 1 when $ABCD$ = 1111.

The truth table can be implement using 7451 multiplexer as shown in Fig. 9-72.

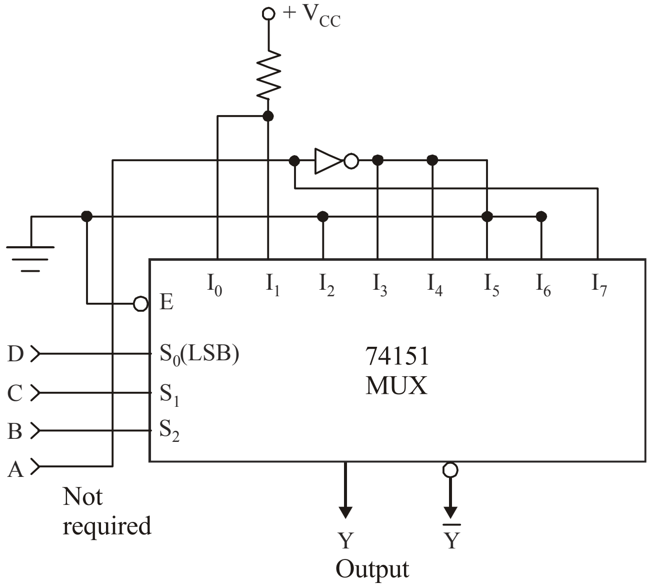


Fig. 9-72.

Example 9-27. *Implement the following Boolean function using 8 : 1 multiplexer*

$f(w, x, y, z) = \Sigma m$ (0, 1, 5, 6, 8, 10, 12, 15) (*VTU, July /Aug. 2004*)

Solution. Given : The Boolean function

$$f(w, x, y, z) = \Sigma m \; (0, 1, 5, 6, 8, 10, 12, 15)$$

The truth table of the functions is shown in Fig. 9-73.

Inputs				*Output*
w	*x*	*y*	*z*	*f*
0	0	0	0	1
0	0	1	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Fig. 9-73.

We implement the function in the form of table according to the step given in Example 9-20. The implement table of the function is shown in Fig. 9-74.

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

Fig. 9-74.

From the implement table we find that the multiplexer input I_0 is 1. $\overline{W}$ is corresponding to the input I_1, I_5 and I_6. The input is 0 corresponding to I_3 input of multiplexer. Winput is applied to the I_4 and I_7 input of multiplexer. 8 : 1 MUX is shown in Fig. 9-75.

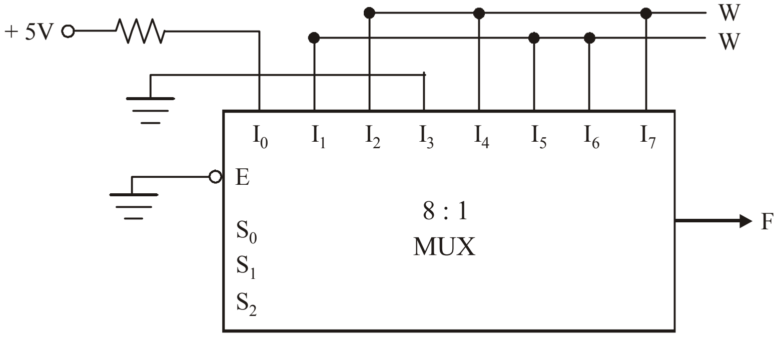


Fig. 9-75.

We can implement the function without implement method also.

Notice that we have entered $f = 1$ for $\overline{w}\,\overline{x}\,\overline{y}\,z$ (i.e. 0000), $\overline{w}\,\overline{x}\,\overline{y}\,\overline{z}$ (i.e. 0001) $\overline{w}\,x\,\overline{y}\,z$, $\overline{w}\,x\,y\,\overline{z}$, $w\,\overline{x}\,\overline{y}\,\overline{z}$, $w\,\overline{x}\,y\,\overline{z}$, $w\,x\,\overline{y}\,\overline{z}$ and $wxyz$, can be implemented using 74151 as shown in Fig. 9-76.

As seen from the circuit, three of the logic variables x, y, and z are connected to SELECT inputs (S_0, $S_1$and S_2). Other combinations are possible. The fourth variable $w$and it inverse $\overline{w}$ are connected to selected data inputs of the multiplexer as required by the desired logic function. The other multiplexer data inputs are connected to LOW and a HIGH as required by the function. For instance, when $xyz = 000, f = I_0 = 1$. We have connected I_0 to HIGH because f is 1 when $wxyz = 0000$ as well as $wxyz = 1000$. Next when $xyz = 001, f = I_1$. We have connected I_1 to the variable $w$through INVERTER (or NOT gate) because $f = 1$ when $wxyz = 0001$ and $f = 0$ when $wxyz = 1001$. Next when $xyz = 010, f = I_2$. We have connected I_2 to the variable w because $f = 0$ when $wxyz = 0010$ and $f = 1$ when $wxyz = 1010$. Next when $xyz = 011, f = I_3 = 0$. Next when $xyz = 100, f = I_4$. We have to connect I_4 to the variable w because $f = 0$ when $wxyz = 0100$ and $f = 1$ when $wxyz = 1100$. Next when $xyz = 101, f = I_5$. We have to connected I_5 to the variable $w$through INVERTER because $f = 1$ when $wxyz = 0101$ and $f = 0$ when $wxyz = 1101$. Similarly when $xyz = 110, f = I_6$ we have to connected I_6 to the variable w through INVERTER. Finally $xyz = 111, f = I_7$. We have to cannot I_7 to the variable w because $f = 0$ when $wxyz = 0111$ and $f = 1$ when $wxyz = 1111$.

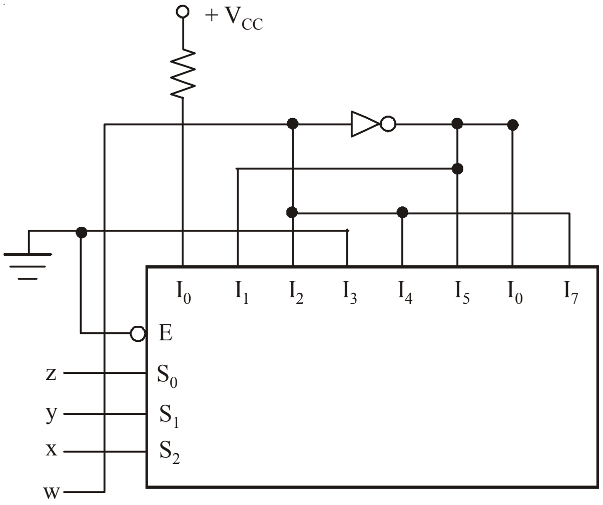


Fig. 9-76.

Example 9-28. *A* ***decoder*** *for a MOD-*10*, BCD up-counter (i.e., a counter that counts in the BCD sequence) is required* (*see Fig.* 9-77)*. The decoder has two outputs P and Q, which respond in the following manner*:

- P = 1 when the BCD counter output is equal and greater than 5_{10}.
- Q = 1 when the BCD counter output is an odd number, *i.e.*, 1_{10}, 3_{10}, etc.

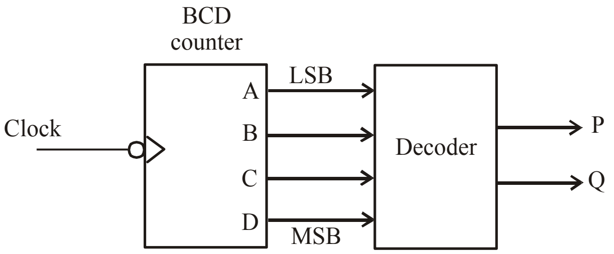


Fig. 9-77.

Your task in this question is to design this decoder circuit using the multiplexer IC.

(*a*) *Determine the truthtable for this decoder, showing all the possible combinations and expected responses at the outputs **P** and **Q**. You are reminded that A is the LSB and D is the MSB and all don't care conditions should be indicated as 'X's.*

(*b*) *Using one 74151 multiplexer IC, show how you would implement the decoder logic circuit for output **P** from the truth-table derived in part (a). The 74151 symbol given in Fig. 9-78 is to be used and multiplexer input S_0 should be designated as the LSB.*

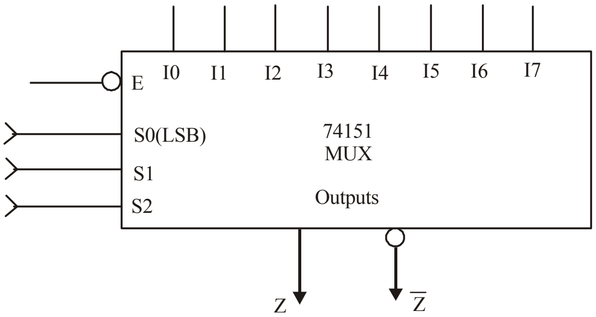


Fig. 9-78.

Solution. Given : The BCD up-counter connected to a decoder shown in Fig. 9-77.

(*a*) We know that P has to be 1 when the BCD counter output is equal and greater than 5. Further Q = 1 when the BCD counter output is an odd number. Using this information, you can prepare a truth table shown in Fig. 9-79. Notice that the outputs P and Q are shown as 'X' (don't care) for the input sequences DCB A = 1010 to 1111. The reason for this is that any number greater than 9 will be an invalid code for BCD counter. Using the truth table, you can write the Boolean expression for P and Q as,

$$P = \overline{D}C\overline{B}A + \overline{D}CB\overline{A} + \overline{D}CBA + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D$$

and

$$Q = \overline{D}\,\overline{C}\,\overline{B}A + \overline{D}\,\overline{C}BA + \overline{D}C\overline{B}A + \overline{D}CBA + D\overline{C}\,\overline{B}A$$

Inputs				*Output*	*Output*
D	***C***	***B***	***A***	***P***	***Q***
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

Fig. 9-79.

(*b*) The logic function for P can be implemented by using 74151 MUX as shown in Fig. 9-80. Notice that CBA are used as select inputs, while D is used on the MUX inputs side.

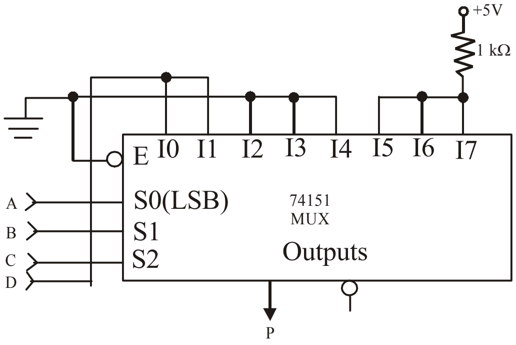


Fig. 9-80.

The logic function for Q can be implemented by using 74151 MUX as shown in Fig. 9-81. Notice that only three variables C, B and A are used as select inputs. D is not required.

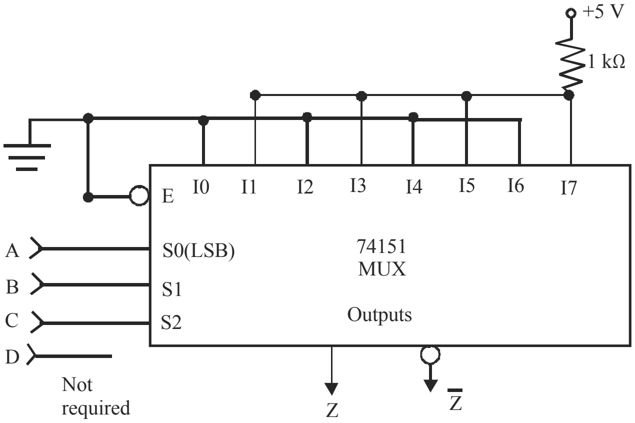


Fig. 9-81.

Example 9-29. *Fig.* 9-82 *shows how a multiplexer can be used to generate logic waveforms with any desirable pattern. The pattern is programmed using eight SPDT (single-pole double-throw) switches and the waveform is repetitively produced by pulsing the MOD-*8 *counter.*

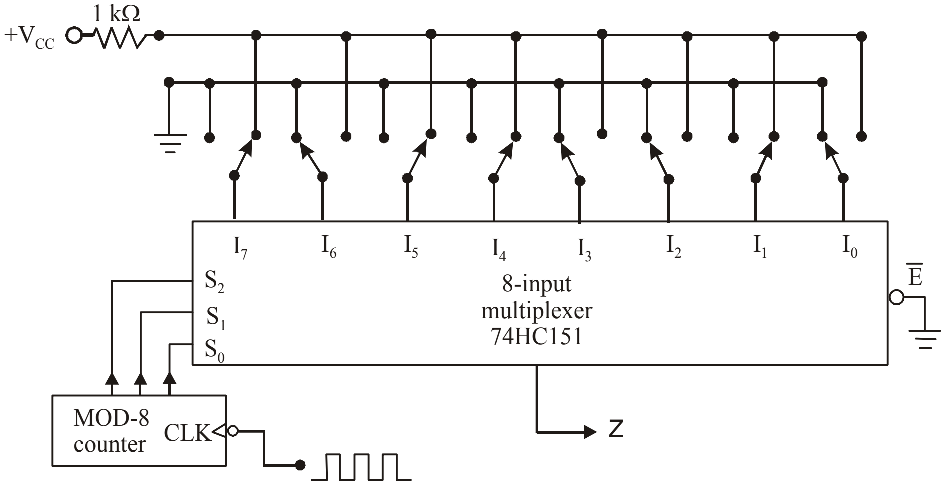


Fig. 9-82.

Draw the waveform at Z for the given switch positions.

Solution. First of all, identify the status of each one of the inputs of the multiplexer (*i.e.* whether it is HIGH or LOW). We find that,

$$I_7 = \text{HIGH}, I_6 = \text{LOW}, I_5 = \text{HIGH}, I_4 = \text{HIGH},$$
$$I_3 = \text{LOW}, I_2 = \text{LOW}, I_1 = \text{HIGH}, I_0 = \text{LOW}$$

Now we recall the operation of the 8-input multiplexer. If the select inputs $S_2\ S_1\ S_0 = 000$, $Z = I_0$, if $S_2\ S_1\ S_0 = 001$, $Z = I_1$, and so on. Applying this logic we can sketch the waveform at the Z output as shown in Fig. 9-83. Notice the arrival of each falling edge of the clock pulse at the MOD-8 counter will change its outputs to the next state. This inturn will change the values of select inputs $S_2\ S_1\ S_0$ from 000 to 111 and then reset to 000 and so on. Thus before the arrival of first falling edge, the counter output is 000, *i.e.* $S_2\ S_1\ S_0$= 000, therefore, the MUX output is I_0 (which is LOW).

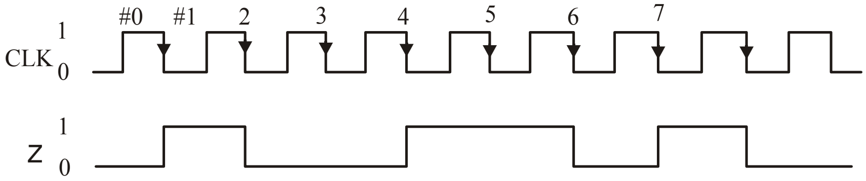


Fig. 9-83.

With the arrival of first falling edge, the counter output changes to 001, So $S_2\ S_1\ S_0 = 001$. This changes the MUX output $Z = I_1$ (which is HIGH). With the arrival of second falling edge, the counter output changes to 010, so $S_2\ S_1\ S_0 = 010$. This changes the MUX output $Z = I_2$ (which is LOW). With the arrival of third NGT, the counter output changes to 011, so $S_2\ S_1\ S_0 = 011$. This changes the MUX output $Z = I_3$ (which is also LOW). In this way, you can determine the MUX output for the subsequent states of MOD-8 counter output.

9-27. Demultiplexers

These are also called *data distributors*. A *demultiplexer* (abbreviated as *DEMUX*) performs the reverse operation of a multiplexer. That is, it takes a single input and distributes it over several outputs. Fig. 9-84 shows a functional diagram for a digital demultiplexer.

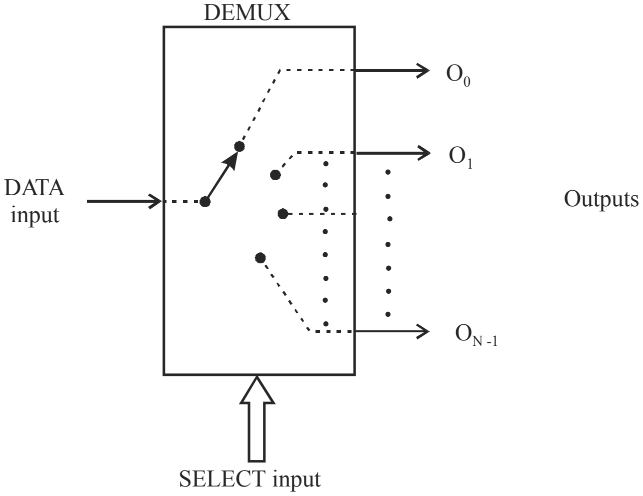


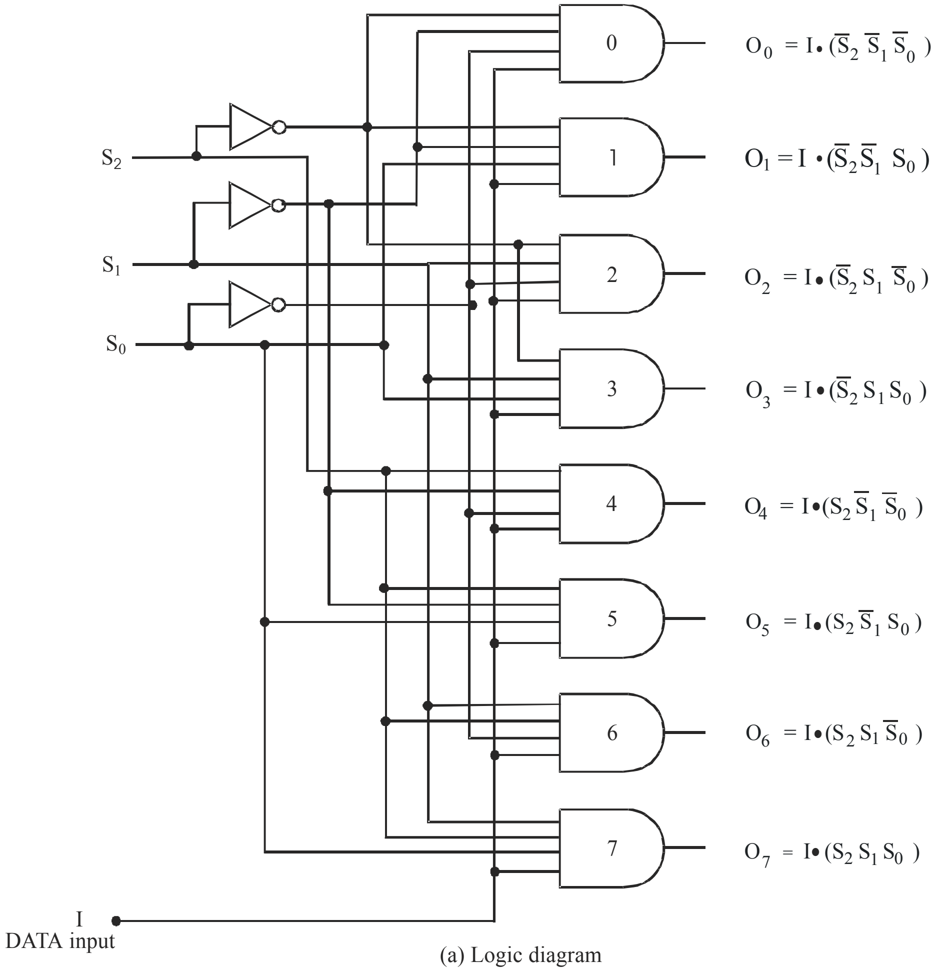
Fig. 9-84.

9-28. 1-Line-to-8-Line Demultiplexer

We have already discussed in the last article that demultiplexer takes a single input and distributes it selectively over several outputs. Fig. 9-85 (*a*) shows a logic diagram for a demultiplexer that distributes one input line to eight output lines.

As seen from the logic diagram of a demultiplexer, the single data input line I is connected to all eight AND gates, but only one of these gates will be enabled by SELECT input lines. For example, with $S_2\,S_1\,S_0 = 000$, only AND gate 0 will be enabled and data input I will appear at output Q_0. Similarly with $S_2\,S_1\,S_0 = 001$, only AND gate 1 will be enabled and data input I will appear at output Q_1 and so on. The complete operation of the demultiplexer is given in the truth table shown in Fig 9-85 (*b*).

It may be carefully noted that the demultiplexer circuit of Fig. 9-85 (*a*) is very similar to the 3-line-to-8-line decoder circuit in Fig 9-5, except that a fourth input (D) has been added to each AND gate. As mentioned in Art 9.3, many IC decoders have an ENABLE input (*e.g.* 74LS138/74 HC138) which is an extra input added to the decoder gates. *This type of a decoder IC can be used as a demultiplexer*. The decoder inputs A, B and C serve as SELECT inputs of the demultiplexer whereas ENABLE input of a decoder serve as data input I of a demultiplexer. It is because of this reason that IC manufactureres often call 3-line-to-8-line decoder IC (74LS 138) as decoder/demultiplexer — *i.e.* it can be used for either function.


S_2
S_1
S_0
0
1
2
3
4
5
6
7
$O_0 = I\bullet(\overline{S}_2\,\overline{S}_1\,\overline{S}_0)$
$O_1 = I\bullet(\overline{S}_2\overline{S}_1\;S_0)$
$O_2 = I\bullet(\overline{S}_2\,S_1\,\overline{S}_0)$
$O_3 = I\bullet(\overline{S}_2\,S_1\,S_0)$
$O_4 = I\bullet(S_2\,\overline{S}_1\,\overline{S}_0)$
$O_5 = I\bullet(S_2\,\overline{S}_1\,S_0)$
$O_6 = I\bullet(S_2\,S_1\,\overline{S}_0)$
$O_7 = I\bullet(S_2\,S_1\,S_0)$
I
DATA input

(a) Logic diagram

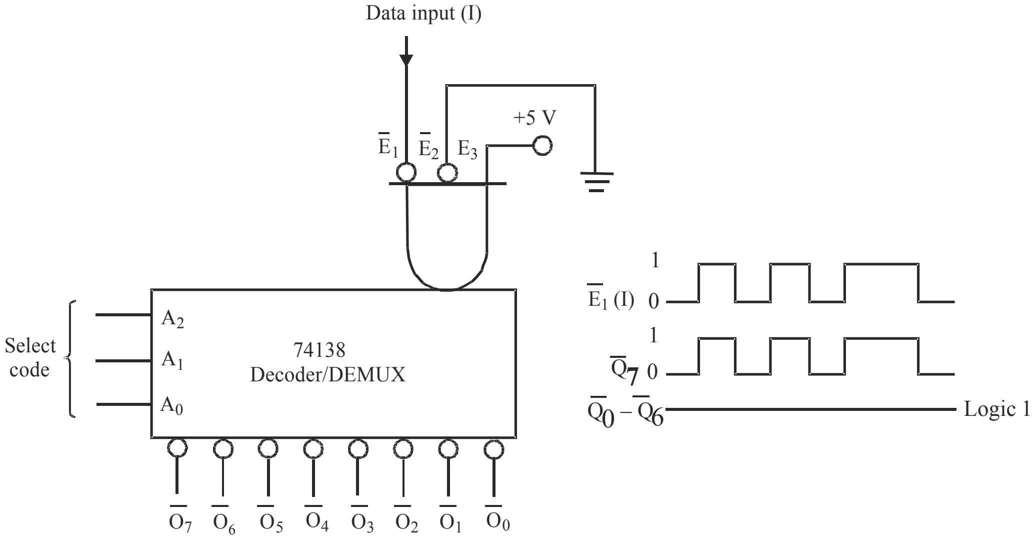
SELECT code			*OUTPUTS*							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

(*b*) Truth table

Fig. 9-85. 1- line - to - 8 - line Demultiplexer

9-29. 74 IC 138 Decoder as a Demultiplexer

We have already discussed in the last article that a 3-line-to-8-line decoder with ENABLE input (74138) can also be used as a demultiplexer. Fig. 9-86 (*a*) shows the use of 74 138 decoder as a demultiplexer. Notice that the enable input $\overline{E}_1$ is used as the data input I_1, while the other two enable inputs are held permanently in their active states. The decoder inputs A_2 A_1 A_0 are used as a SELECT code inputs for the demultiplexer.


Data input (I)
$\overline{E}_1$
$\overline{E}_2$
E_3
+5 V
Select code
A_2
A_1
A_0
74138
Decoder/DEMUX
$\overline{O}_7$ $\overline{O}_6$ $\overline{O}_5$ $\overline{O}_4$ $\overline{O}_3$ $\overline{O}_2$ $\overline{O}_1$ $\overline{O}_0$
1
$\overline{E}_1$ (I) 0
1
$\overline{Q}_7$ 0
$\overline{Q}_0 - \overline{Q}_6$
Logic 1

(a) Circuit diagram

(b) Waveforms for AAA= 111

Fig. 9-86. 74 138 decoder as a demultiplexer

In order to illustrate the operation of 74138 decoder as a demultiplexer, let us assume the select inputs are 111. With this input code, the only output can be activated is $\overline{O}_7$ while all other outputs are HIGH. The output $\overline{O}_7$ will go LOW only if $\overline{E}_1$ goes LOW and will be HIGH. In other words, the output $\overline{O}_7$ will follow the signal on $\overline{E}_1$ (*i.e.*, the data input D) while all other outputs stay HIGH as shown in Fig. 9-86 (*b*). In a similar manner, if we apply a different select code at A_2 A_1 A_0, it will cause the corresponding output to follow the data input, I.

9-30. Applications of Demultiplexer

Demultiplexers have numerous applications in the field of digital electronics. But followings are important from the subject point of view.

1. Clock demultiplexer
2. Security monitoring system and
3. Synchronous data transmission system.

The application of demultiplexer for distributing CLOCK to different devices/components is discussed below.

9-31. Clock Demultiplexer

Fig. 9-87 shows the IC 74138 decoder/demultiplexer chip being used as a clock distributor. Under control of the SELECT lines, the clock signal is routed to the desired destination. For example, if S_2 S_1 S_0 = 000, the clock signal applied to I will appear at output $\overline{O}_0$ a counter circuit is shown connected at this output . With S_2 S_1 S_0 = 001, the clock signal will appear at $\overline{O}_1$ (a shift register is shown connected at this output). Similarly with S_2, S_1, S_0 = 001, the clock signal will appear at $\overline{O}_2$ (where we may have some circuit/device needing a clock signal) and so on.

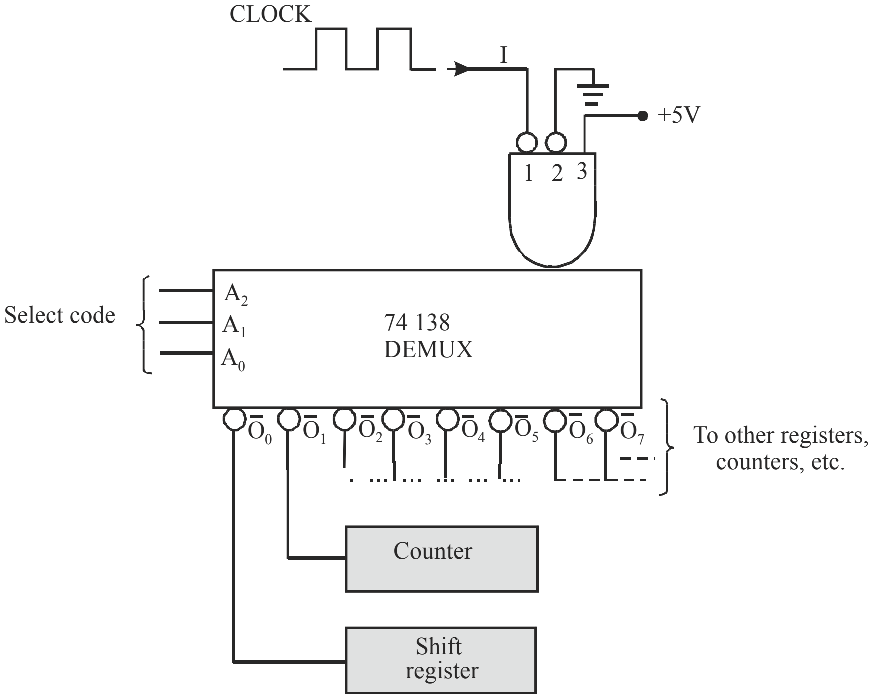


Fig. 9-87. Clock demultiplexer

9-32. Magnitude Comparator – IC 7485

It is a combinational logic circuit that compares two input binary quantities and generates outputs to indicate which one has the greater magnitude. Fig. 9-88 shows the logic symbol and Table 9-6 shows the truth table for a four-bit magnitude comparator. It is available as 7485, 74LS85 and 74HC 85.

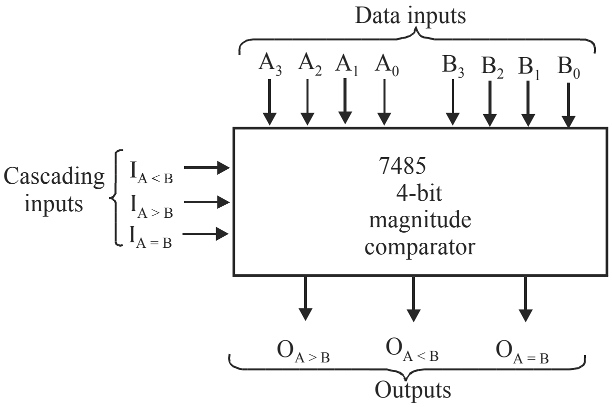


Fig. 9-88. Logic symbol

As seen from the logic symbol, the 7485 compares two unsigned four-bit binary numbers. One of them is $A_3\,A_2\,A_1\,A_0$ and the other is $B_3\,B_2\,B_1\,B_0$. These numbers are referred to as word A and word B respectively. The 7485 has three active-HIGH outputs. The output $O_{A>B}$ will be HIGH when the magnitude of word A is greater than the magnitude of word B. The output A < B will be HIGH when the magnitude of word A is less than the magnitude of word B. The output $O_{A=B}$ will be HIGH when word A and B are identical.

Table 9-6

COMPARING INPUTS				***CASCADING INPUTS***			***OUTPUTS***		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	1	0	0
$A_3<B_3$	X	X	X	X	X	X	0	1	0
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	1	0	0
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	1	0	0	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	1	0	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	1	0	0	1
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	0	0	1	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	1	1	0	0	0	0

In order to illustrate the operation of 7485, let us consider the comparison of two 4-bit numbers, $A = A_3 A_2 A_1 A_0 = 1010$ and $B = B_3 B_2 B_1 B_0 = 1011$. The magnitude comparator will compare A_3 with B_3, A_2 with B_2, A_1 with B_1 and A_0 with B_0. On comparison we find that $A_3 = B_3 = 1$, $A_2 = B_2 = 0$, $A_1 = B_1 = 1$ and $A_0 = 0$ and $B_0 = 1$ So we find that $A_0 < B_0$. As a result $Q_{A<B}$ output of magnitude comparator will go HIGH indicating word A is smaller than word B (*i.e.* A<B).

Example 9-30. *The binary numbers A = 1100 and B = 1011 are applied to the inputs of a 7485 (4-bit magnitude comparator). Determine the outputs.*

Solution. Given, the two binary numbers, A = 1100 and B = 1011.

$$A = A_3, A_2, A_1, A_0 = 1100$$

$$B = B_3, B_2, B_1, B_0 = 1011$$

The magnitude comparator will compare A_3 with B_3, A_2 with B_2, A_1 with B_1 and A_0 with B_0. On comparison we find that $A_3 = 1$ and $B_3 = 1$, $A_2 = 1$ and $B_2 = 0$. Now since $A_2 > B_2$, the magnitude comparator will stop making further comparisons. So the 7485 will make $Q_{A>B}$ output HIGH indicating binary number A > B.

9-33. Cascading of IC 7485 to Compare Two 8-bit Numbers

We have already discussed in the last article that 7485 is used to compare two 4-bit binary numbers. Now we will study as how 7485s can be connected together to compare two 8-bit binary numbers. Fig. 9-89 shows the connections between the two 7485s also called cascading to do the comparison.

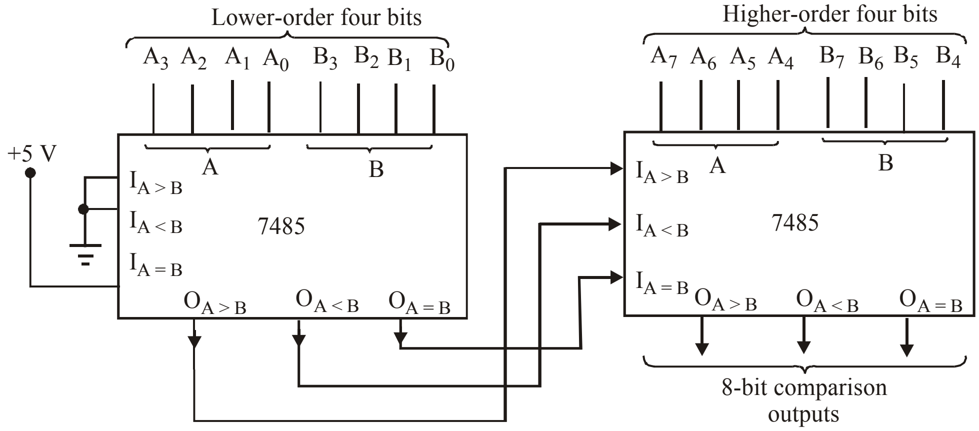

Lower-order four bits
A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0
+5 V
A
B
$I_{A>B}$
$I_{A<B}$
$I_{A=B}$
7485
$O_{A>B}$ $O_{A<B}$ $O_{A=B}$
Higher-order four bits
A_7 A_6 A_5 A_4 B_7 B_6 B_5 B_4
A
B
$I_{A>B}$
$I_{A<B}$
$I_{A=B}$
7485
$O_{A>B}$ $O_{A<B}$ $O_{A=B}$
8-bit comparison outputs

Fig. 9-89.

As seen from the diagram, there are two magnitude comparators. The one on the left is used to compare the lower-order 4-bits of the two 8-bit binary numbers. On the other hand, the comparator on the right is used to compare the higher-order 4-bits of the two 8-bit numbers. Notice the connections of the cascading inputs of the lower-order comparator. As seen, $I_{A>B}$ and $I_{A<B}$ inputs of the lower-order comparator are tied LOW, whereas $I_{A=B}$ input is tied HIGH. Further, the three outputs of lower-order comparator are connected to cascading inputs of higher-order comparator. That is $Q_{A>B}$ of the lower-order comparator is connected to $I_{A>B}$ of the higher-rder comparator. Similarly, $Q_{A<B}$ of the lower-order comparator is connected to $I_{A<B}$ of the higher-order comparator. And $Q_{A=B}$ of the lower-order comparator is connected to $I_{A=B}$ of the higher-order comparator.

In order to understand the operation of the circuit, let us consider an example. Suppose we have two eight-bit numbers, $A_7 A_6 A_5 A_4 A_3 A_2 A_1$ = 10111001 and $B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0$ = 11101111. The high-order comparator compares its inputs $A_7 A_6 A_5 A_4$ = 1011 and $B_7 B_6 B_5 B_4$ = 1110 and produces $Q_{A<B} = 1$ regardless of what levels are applied to its cascade inputs from the lower-order comparator. In other words, once the higher-order comparator senses a difference in the higher-order bits of the two 8-bit numbers, it knows which 8-bit number is greater without having to look at the results of the low-order comparison.

Let us consider another example. Suppose we have two 8-bit numbers: $A_7 A_6 A_5 A_4 A_3 A_2 A_1$= 10101011 and $B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0$=10101001. This time the higher-order comparator finds $A_7 A_6 A_5 A_4$= $B_7 B_6 B_5 B_4$ =1010. So it must look at cascaded inputs to see the result of low-order comparison. The lower-order comparator has $A_3 A_2 A_1 A_0$= 1011 and $B_3 B_2 B_1 B_0$=1001, so it produces a 1 at its $Q_{A>B}$ output and the $I_{A>B}$ input of the higher-order comparator. The higher-order comparator senses this 1 and since its data inputs are equal, it produces a HIGH on its $Q_{A>B}$ ouput to indicate the result of comparison of two eight-bit numbers.

9-34. Applications of Magnitude Comparator

We have already discussed in Art. 9-29 that 7485 – magnitude comparator is used to compare two 4-bit binary numbers. If we cascade, two comparators, it is possible to compare two 8-bit binary numbers. Besides this, magnitude comparators are also useful in control applications where a binary number representing the physical variable being controlled (*e.g.*, position, speed, acceleration or temperature) is compared with a reference value. The comparator outputs are used to actuate circuitry to drive the physical variable toward the reference value.

9-35. Algorithmic State Machines (ASM)

The binary information in digital system can be classified as either data or control information. Data are discrete elements of information that are manipulated to perform arithmetic, logic, shift and other similar data- processing tasks. These operations are implemented with digital components such as adders, decoders, multiplexers, counters and shift registers.

Control information provides command signals that supervise the various operations in the data selection in order to accomplish the desired data-processing tasks. The logic design of a digital system is divided in two parts, they are given below:

1. Data Processor Unit – Design of digital circuits that perform the data-processing operations.
2. Control Logic Circuit – Design of control circuit that supervises the operations and their sequence.

The data processor unit contains registers to manipulate the data according to the system requirements. The control logic sends sequence commands to the data processor by using the status conditions from the data processor unit. The relation between the data processor and control logic in a digital system is shown in Fig. 9-90.

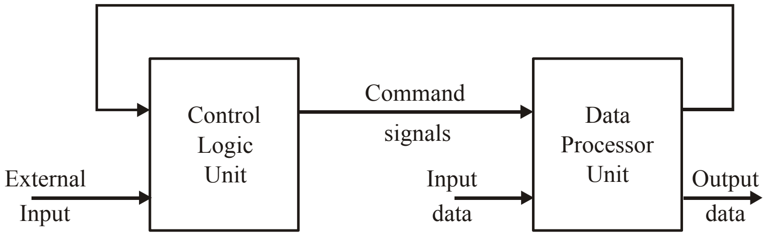


Fig. 9-90.

The control sequence and data- processing tasks of a digital system are specified by algorithm. An algorithm consists of a finite number of procedural steps that specify how to obtain a solution to a problem.

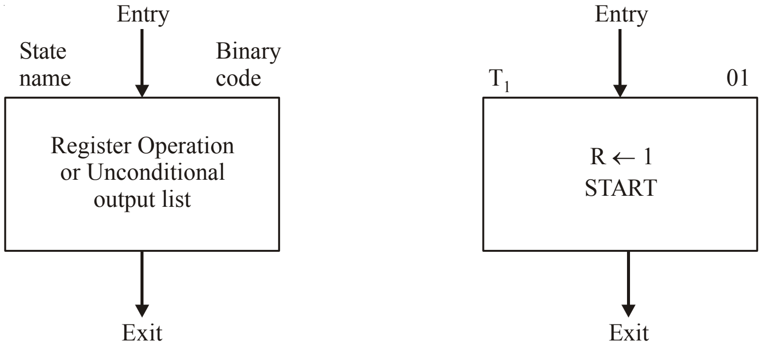
A flow chart is a convenient way to specify the sequence of procedure steps and decision paths for an algorithm. A flow chart for a hardware algorithm translates the word statement into the information diagram which indicates the sequence of operations with necessary conditions for their execution, but in case of sequential circuits, it is necessary to define the timing relationship between sequence of operations conditions required.

A special flow chart that has been developed specifically to define hardware algorithms is called a algorithmic state machine (ASM) chart or SM charts state machine flow charts. This flow chart describe the operations in step by step manner of a digital system to get the desired performance.

9-36. ASM Charts

The ASM chart is a special type of flow chart suitable for describing the sequential operations in a digital system. The chart is composed of three basic elements, they are given below:

1. State Box : state of any system is indicated by a state box. The shape of the state box is a rectangular with in which are written register operations or output signal names that control generates while being in this state. The state is given by a symbolic name, which is placed at the upper left corner of the box. The binary code assigned to the state is placed at the upper right corner as shown in Fig. 9-91 (a). Each state box contains only one entry and one exit path. Fig. 9-91 (b) shows a specific example of a state box.



(a) General Description

(b) Specific Example

Fig. 9-91.

The state has a symbolic name T_1; and binary code assigned to it is 01. Inside the box is written the register operation $R \leftarrow 1$, which indicates the register R is initidized to 1, when the system is in state T_1. The START name inside the box may indicate, an unconditional output signal that states a certain operation.

2. Decision Box : It is a diamond shoped box used to describe the effect of an input on the control subsystem. It has two or more exit paths and only one entry path. The input condition to be tested is written inside the box. One exit path or an expression is taken if the condition is true and another path when the condition is false. When an input condition is assigned a binary value, the two parts are indicated by 0 to 1 as shown in Fig. 9-92 (a).

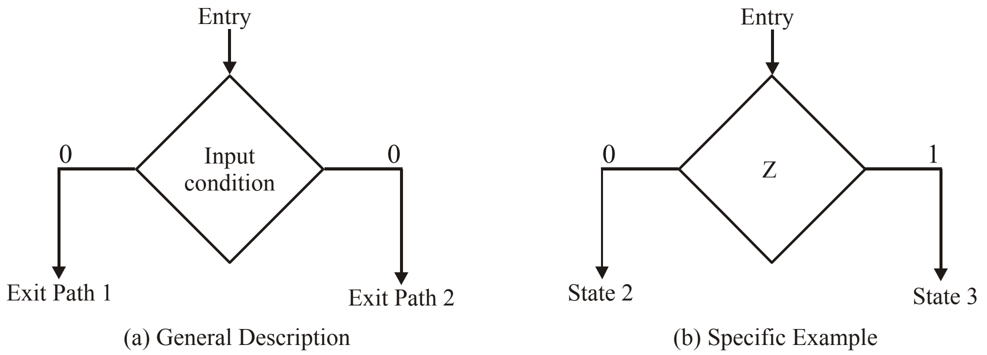


Fig. 9-92.

Fig. 9-92 (b) shows the specific example. Here, the Z is an input, if it is 0, control goes to state 2, else goes to state 3.

3. Conditional Box : It is a unique box of ASM chart. The area shape of the conditional box is shown in Fig. 9-93.. The second corners differentiate it from the state box. It has one entry path and one exit path. The input path to the conditional box must come from one of the exit paths of a decision box. This box is used to represent the register operation or output conditions which are written inside when the control is in that state provided that the input condition is satisfied.

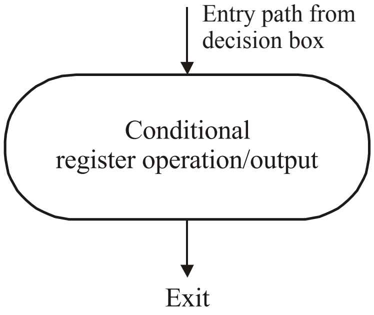

Entry path from decision box
Conditional register operation/output
Exit

Fig. 9-93.

The outputs that are not listed in either the state box or in conditional box in a particular ASM chart are always inactive when the digital system/control is in that state. Fig. 9-94 shows an example

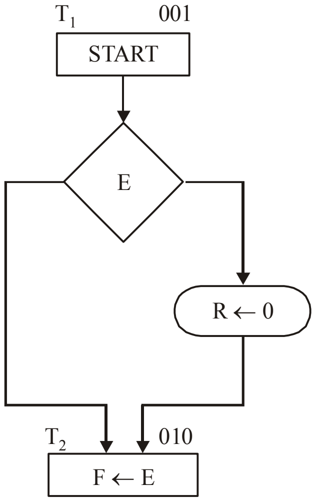


Fig. 9-94.

with a conditional box. The control generates a START output signal which in state T_1 while in state T_1, the control checks the status of input E. If $E = 1$, then R is cleared to 0; otherwise, R remains unchanged. In either case, the next state is T_2.

9-37. ASM Block

An ASM block is a structure consisting of one state box and all the decision and conditional box connected to its exit path. An ASM block has one extrance and any number of exit paths represented by the structure of the decision boxes. An ASM chart consists of one or more interconnected blocks. An example of an ASM block is shown in Fig.9-95. The ASM block consists of one state box and may contain one or more decision and conditional boxes connected to each exit path. This block consist with state T_1 are two decision boxes and one conditional box. The diagram distinguishes the block with dashed lines around the entire structure.

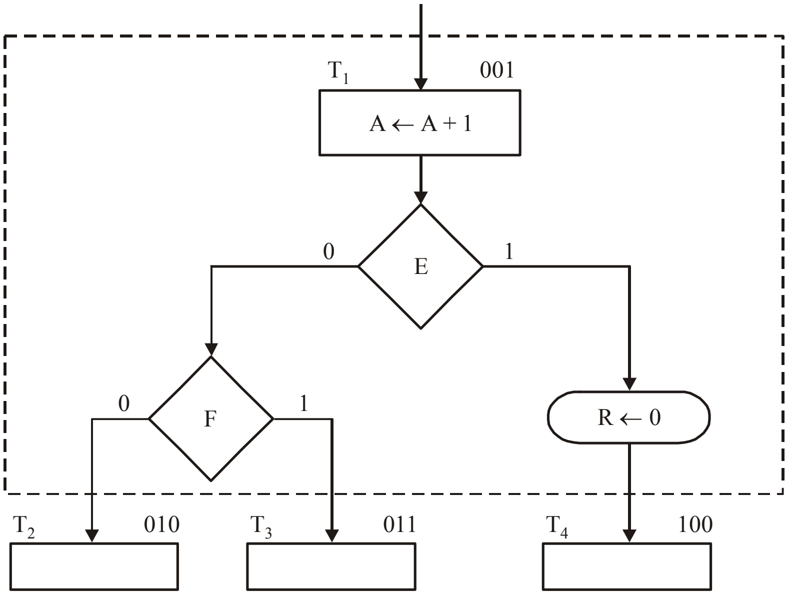


Fig. 9-95. ASM Block

Each block in the ASM chart describes the state of the system during one clock pulse interval. The operations within the state and conditional boxes are executed with a common clock pulse while the system is in state T_1. The same clock pulse also transfers the system controller to one of the next states, T_2, T_3 & T_4 as dictated by the binary values of E and F.

The ASM chart is similar to the state diagram. Each state block is equivalent to a state in a sequential circuit. The decision box is equivalent to the binary information written along the directed lines that connect two states in a state diagram.

9-38. Register Operations

A register includes storage registers, shift registers counters, and single flip-flops. Example of register operations are shift, increment, add clear, and data transfer. It is sometimes convenient to adopt a suitable notation to describe the operations performed among the registers.

Symbolic notation of register operation is shown in Fig. 9-96. A register is designated by one or more capital letters such as A, B or RA. The transfer of data from one register to another is symbolized by a directed arrow that denotes a transfer of contents from the source register to the destination register. Some of the symbolic notation for register operations is shown in table.

Symbolic Notation	***Description***
$A \leftarrow B$	Transfer contents of register B into register A
$R \leftarrow 0$	Clear Resister R
$F \leftarrow 1$	Set flip-flop F to 1
$A \leftarrow A+1$	Increment register A by 1 (count-up)
$A \leftarrow A-1$	Decrement register A by 1 (count-down)
$A \leftarrow A+B$	Add contents of register B to register A

Fig. 9-96.

9-39. Design Example with ASM Chart

Let design a digital system with two flip-flops, *E* and *F*, and one 4-bit binary counter *A*. The individual flip-flops in A are denoted by A_4, A_3, A_2 and A_1, with A_4 holding the most significant bit of the count. *A* start signal *S* initiates the system operation by clearing the counter *A* and flip flop *F*. The counter is then incremented by 1 starting from the next clock pulse and continues to increment until the operation stop. Counter bits A_3 and A_4 determine the sequence of operations.

If $A_3 = 0$, *E* is cleared to 0 and count continues

If $A_3 = 1$, *E* is set to 1; then if $A_4 = 0$ the count continues, but if $A_4 = 1$, *F* is set to 1 on the next clock pulse and the system stops counting.

ASM chart shown in Fig. 9-97, when no operations are performed, the system is the initial state T_0, waiting for the start signal *S*. When input *S* is equal to 1, counter *A* and flip-flop *F* are cleared to 0 and the controller goes to state T_1. Note the conditional box that follows the decision box for *S*. This means that the counter and flip-flop will be cleared during T_0 if $S = 1$, and at the same time, control transfer to state T_1. The block associated with state T_1 has two decision boxes and two conditional boxes. The counter is incremented with every clock pulse. At the same time, one of these possible operations occur during the same clock pulse transition.

Either *E* is cleared and control stays in state $T_1(A_3 = 0)$; or

E is set and control stays in state $T_1(A_3A_4 = 10)$; or

E is set and control goes to state $T_2(A_3A_4 = 11)$.

When the control is in state T_2, flip-flop *F* is set to 1 and the circuit goes back to its initial state, T_0.

The ASM chart consists of three states and three blocks. The blocks associated with T_0 consists of the state box, one decision box and one conditional box. The block associated with T_2 consists of only the state box. The control logic has one external input, *S*, and two status inputs, A_3 and A_4.

Every block is an ASM chart specifies the operations that are to be performed during one common clock pulse. Table 9-7. shows the binary values of the counter and two flip-flops after every clock pulse. The table also shows separately the status of A_3 and A_4 as well as the present state of the controller.

We start with state T_1 right after the input signal *S* has caused the counter and flip-flop *F* to be cleared. The value of *E* is assumed to be 1, because *E* is equal to 1 at T_0 and because *E* does not change during the transition from T_0 to T_1. The system stays in state T_1 during the next thirteen clock pulses. Each pulse increments the counter and either clears of sets *E*. Note the relationship between the time at which A_3 becomes a 1 and the time at which *E* is set to 1, when $A = 0011$, the next clock pulse increments the counter to 0110, but that same clock pulse see the value of A_3 as 0, so E is cleared. The next pulses changes the counter from 0100 to 0101, and now A_3 is initially equal to 1, so E is set to 1. Similarly, E is cleared to 0 not when the counter goes from 0111 to 1000, but when it goes from 1000 to 1001, which is when A_3 is 0 in the present value of the counter.

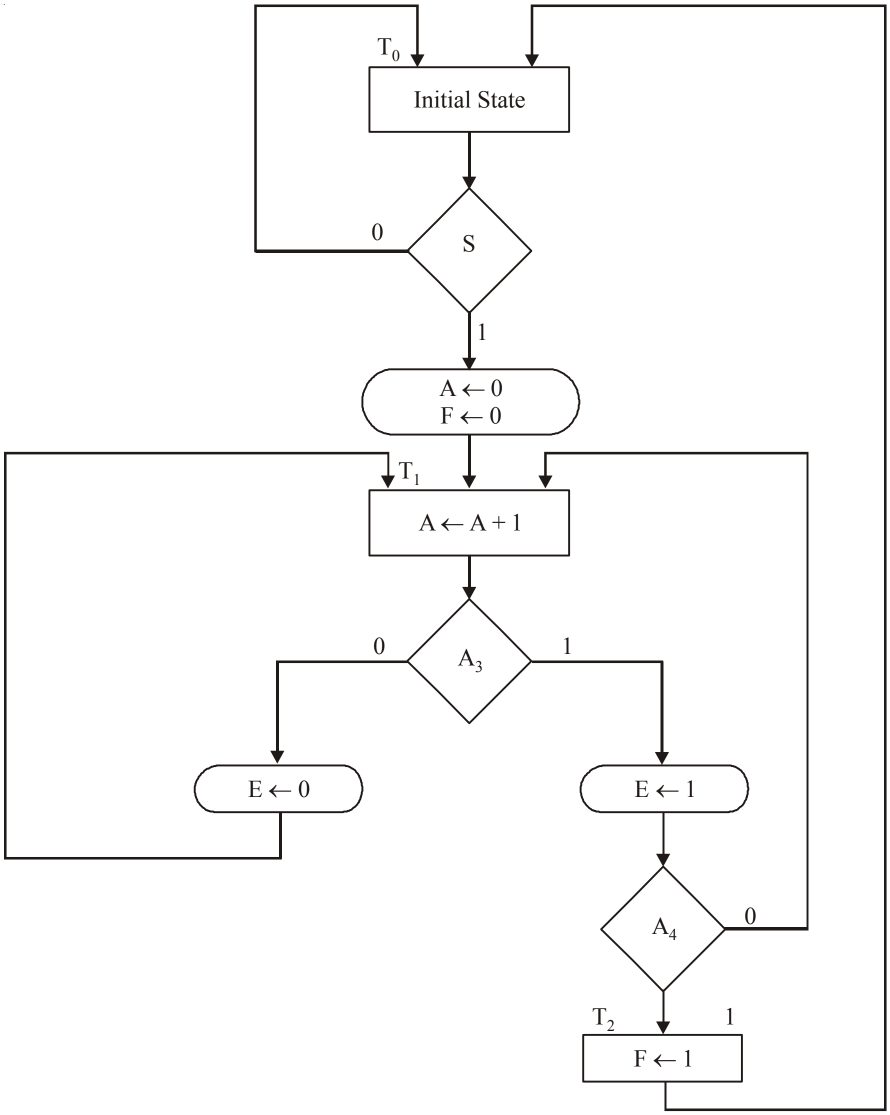


Fig. 9-97.

When the count reaches 1100, both A_3 and A_4 are equal to 1. The next clock pulses increment A by 1, sets E to 1, and transfers control to state T_2 control stays in T_2 for only one clock period. The pulse transition associated with T_2 sets flip-flop F to 1 and transfer control to state T_0. The system stays in the initial state T_0 as long a S is equal to 0.

Table 9-7. Sequence of operation for Design

Counter				***Flip-flops***		***Conditions***	***State***
A_4	A_3	A_2	A_1	E	F		
0	0	0	0	1	0	$A_3 = 0, A_4 = 0$	
0	0	0	1	0	0		T_1
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_3 = 1, A_4 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_3 = 0, A_4 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_3 = 1, A_4 = 1$	
1	1	0	1	1	0		T_2
1	1	0	1	1	1		T_0

From observation of table 9-7 it may seen that the operations performed on *E* are delayed by one clock pulse. This is the difference between and ASM chart and a conventional flow chart.

9-40. Design with Multiplexers

The major goal of control-logic design is the development of a circuit that implements the desired control sequence in a logical and straightforward manner. The attempt to minimize the number of gates tends to produce an irregular network, making it difficult for anyone but the designer to identify the sequence of events the control undergoes.

As a consequence, it is difficult to alter, service, or maintain the equipment after the initial design. The sequence of states in the control should be clearly evident from the circuit configuration even if this requires additional components and results in a nonminimal circuit. The multiplexer method is such an implementation.

The control circuit consists of three components: the flip-flops that hold the binary state value, the decoder that generates the control outputs, and the gates that determine the next state. We now replace the gates with multiplexers and use a register for the individual flip-flops. This design method results in a regular pattern of three of levels of components. They are given below:

1. The first level consists of multiplexers that determine the next state of the register.
2. The second level contains a register that holds the present binary state.
3. The third level has the decoder that provides a separate output for each control state.

Consider, for example, the ASM chart of Fig. 9-98. It consists of four states four control inputs. The state boxes are left empty in this case because we are interested only in the control sequence, which is independent of the register operations. The binary assignment for each state is indicated at the upper right corner of the state boxes. The decision boxes specify the state transitions as a function of the four control inputs, *w*, *x*, *y*, and *z*.

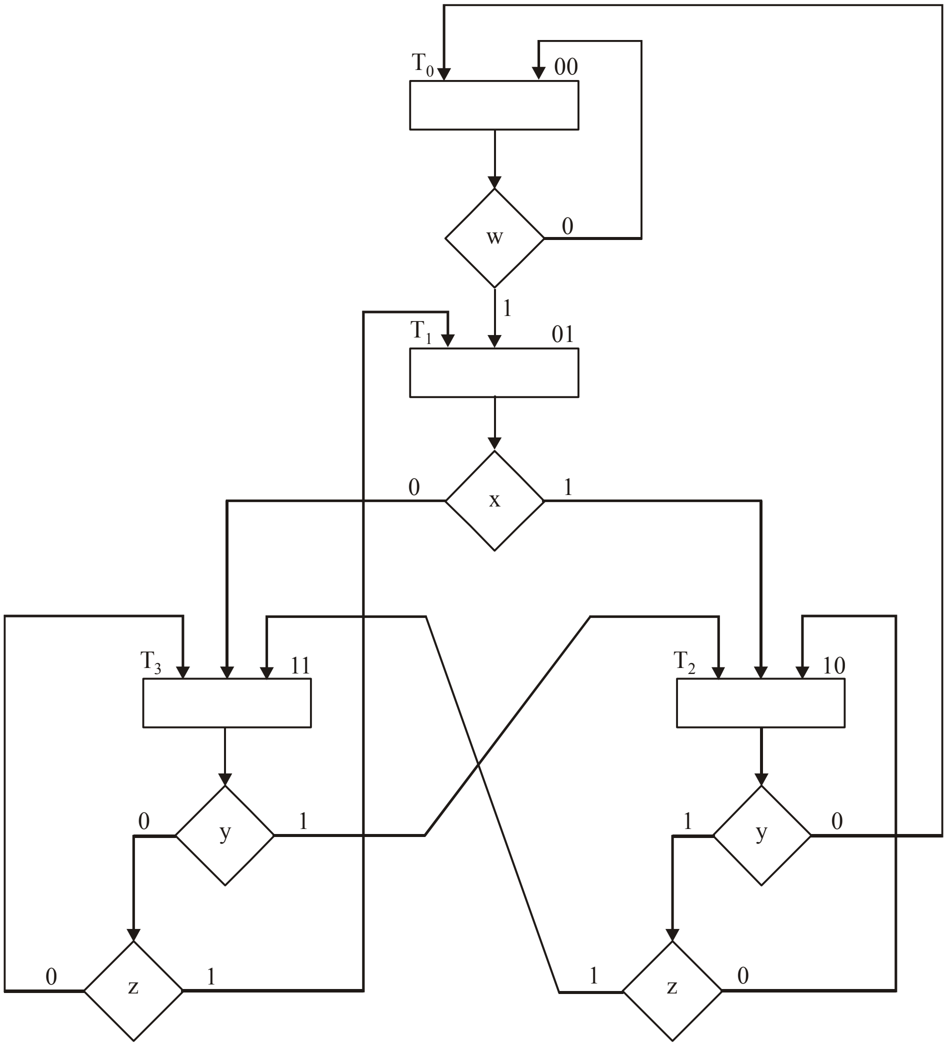


Fig. 9-98. Example of ASM Chart

The three-level control implementation is shown in Fig. 9-99. It consists of two multiplexers, MUX1 and MUX2; a register with two flip-flops, G_1 and G_2 and a decoder with four outputs. The outputs of the register are applied to the decoder inputs and also to the select inputs of the multiplexers. In this way, the present state of the register is used to select one of the inputs from each multiplexer. The outputs of the multiplexers are then applied to the D inputs of G_1 and G_2. The purpose of each multiplexer is to produce an input to its corresponding flip-flop equal to the binary value of the next state.

The inputs of the multiplexers are determined from the decision boxes and state transitions given in the ASM chart. For example, state 00 stays at 00 or goes to 01, depending on the value of input w. Since the next state of G_1 is 0 in either case, we place a signal equivalent to logic 0 in MUX1 input 0. The next state of G_2 is 0 if $w = 0$ and 1 if $w = 1$. Since the next state of G_2 is equal to w, we apply control input w to MUX2 input 0. What this means is that when the select inputs of the multiplexers are equal to present state 00, the outputs of the multiplexers provide the binary value that is transferred

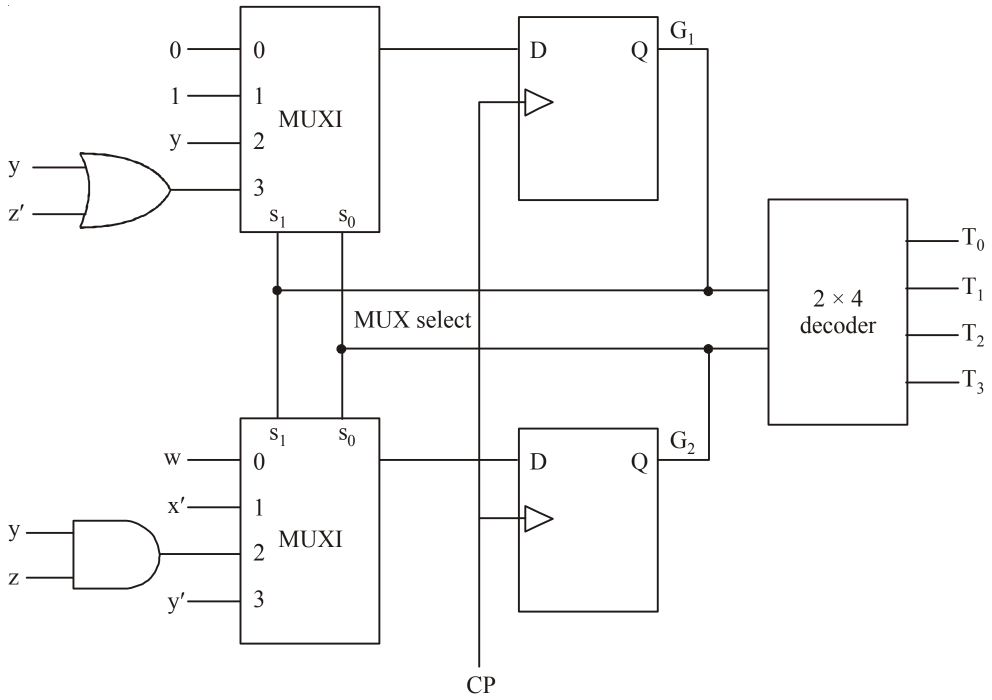


Fig. 9-99. Control implementation with multiplexers

to the register during the next clock pulse. To evaluation of the multiplexer inputs the table showing the input conditions for each possible transition in the ASM charts. Table 9-8 gives this information for the ASM chart of Fig. 9-99.

Table 9-8.

Present State		***Next State***			***Multiplexer Inputs***	
G_1	G_2	G_1	G_2	***Input Conditions***	***MUX1***	***MUX2***
0	0	0	0	w'		
0	0	0	1	w	0	w
0	1	1	0	x		
0	1	1	1	x'	1	x'
1	0	0	0	y'		
1	0	1	0	yz'		
1	0	1	1	yz	$yz' + yz = y$	yz
1	1	0	1	$y'z$		
1	1	1	0	y		
1	1	1	1	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

There are two transitions from present state 00 or 01 and three transitions from present state 10 or 11. These are separated by horizontal lines across the table. The input conditions listed in the table are obtained from the decision boxes in the ASM chart. For example, from Fig. 9-90, we note that present state 01 will go to next state 10 if $x = 1$ or to next state 11 if $x = 0$. In the table, we mark these input conditions as x and x', respectively. The two columns under "multiplexer inputs" in the table specify the input values that must be applied to MUXI and MUX2. The multiplexer input for each present state is determined from the input conditions when the next state of the flip-flop is equal to 1. Thus, after present state 01, the next state of G_1 is always equal to 1 and the next state of G_2 is equal to the complement value of x. Therefore, the input of MUX1 is mad equal to 1 and that of MUX2 to x' when the present state of the register is 01. As another example, after present state 10, the next state of G, must be equal to 1 if the input conditions are yz' or yz. When these two Boolean terms are ORed together and then simplified, we obtain the single binary variable y, as indicated in the table. The next state of G_2 is equal to 1 if the input conditions are $yz = 11$. If the next state of G_1 remains at 0 after a given present state, we place a 0 in the multiplexer input as shown in present state 00 for MUX1. If the next state of G_1 is always 1, we place a 1 in the multiplexer input as shown in present state 01 for MUX1. The other entries for MUX1 and MUX2 are derived in a similar manner. The multiplexer inputs from the table are then used in the control implementation of Fig. 9-91. Note that if the next state of a flip-flop is a function of two or more control variables, the multiplexer may require one or more gates in its input. Otherwise, the multiplexer input is equal to the control variable, or the complement of the control variable, or 0, or 1.

SUMMARY

In this chapter, you have learned that :

1. A decoder is a device that accepts a set of inputs which represent a binary number and activates only the output that corresponds to the input number. IC 74138 is one of the most popular 3-line-to-8-line decoder.
2. Decoder is widely used as a code converter. Some of the major applications are Binary-to-BCD converter, BCD-to-Decimal Decoder/Driver, BCD-to-Seven-Segment Decoder etc.
3. An encoder is a device that has a number of input lines, only one of which is activated at a given time and produces an N-bit output code.
4. A priority encoder is a device that includes the necessary logic to ensure that when two or more inputs of an encoder are activated, the output code will correspond to the highest numbered input.
5. Multiplexer is a device that selects 1 out of N input data sources and transmits the selected data to a single output channel. IC 74151-8-input multiplexer is the most popular multiplexer used in the field of digital electronics. IC 74157 is another useful device. This device has two four-input multiplexers in the same package.
6. Multiplexer is widely used device for logic function generation, selecting data, routing of data, waveform generation etc.
7. Demultiplexer is a device that performs the reverse operation of a multiplexer. It takes a single input and distributes it over several outputs. A decoder can also be used to operate as a demultiplexer.
8. A magnitude comparator-IC 7485 compares two 4-bit binary numbers and generates outputs to indicate which one has a greater magnitude.

GLOSSARY

BCD-to-Decimal decoder/driver. A digital circuit/ device that converts a BCD input into a single equivalent decimal number.

BCD-to-Seven Segment decoder/driver. A digital circuit/ device that takes a four-bit BCD input and activates the required outputs to display the equivalent deciment digit on a seven-segment LED display.

Decoder. A digital circuit/device that converts an input binary code into a corresponding single numeric output.

Demultiplexer. A digital circuit/device that depending on the status of its select inputs will channel its data to one of the several data outputs.

Encoder. A digital circuit/device that produces an output code depending upon which of its inputs is activated.

Multiplexer. A digital circuit/device that depending upon the status of its select inputs, will channel one of several data inputs to its output.

Priority Encoder. A special circuit/digital device that senses when two or more inputs are activated simultaneously and then generates a code coresponding to the highest-numbered input.

DESCRIPTIVE QUESTIONS

1. Design one digit BCD to Binary converter using 74184.
(*Anna University, Apr. /May 2008*)
2. What is the simplest logic circuit for a decoder that produces a 1 output when the BCD input is 0000?
(*Anna University, Nov. /Dec. 2005*)
3. Draw the logic circuit for 3 lines to 8 line decoder.
(*PTU, Dec. 2009*)
4. Give significance of priority encoder.
(*PTU, May 2009*)
5. Realise a BCD to Excess 3 code converter using suitable gates.
(*Nagpur University, 2008*)
6. Give the logic of a 4:1 multiplexer. Draw the gate level diagram.
(*Gauhati University, 2007*)
7. What are the applications of demultiplexers.
(*Mahatma Gandhi University, Jan. 2007*)
8. Design BCD to Excess 3 code converter using NOR gates.
(*VTU, Jul. /Aug. 2005*)
9. Explain with an example how logic functions can be implemented suing multiplexers.
(*Mahatma Gandhi University, Jan. 2007*)
10. Differentiate between encoders, decoders and demultiplexers.
(*Mahatma Gandhi University, Dec. 2007*)
11. How can a decoder are used as a demultiplexer.
(*Mahatma Gandhi University, Nov. 2005*)
12. Implement a full subtractor using 4:1 multiplexers and inverter.
(*Mahatma Gandhi University, Nov. 2005*)
13. Draw the truth table, logic symbol and the completer logic circuit of a 3-to-8 demultiplexer.
(*Mahatma Gandhi University, Nov. 2005*)

14. Prove that a 2-input multiplexer is a universal logic module.
(*Mahatma Gandhi University, May 2005*)

15. Design a 8:1 multiplexer and explain how it works. Distinguish between Multiplexer and Encodrer.
(*Mahatma Gandhi University, May 2005*)

16. Construct 8:1 multiplexer using 2:1 multiplexer
(*VTU, Jul. /Aug. 2005*)

17. Define multiplexer. Show how to convert a decoder into demultiplexer. Indicate how to add a stroke to this system.
(*PTU, May 2008*)

18. What is a demultiplexer? Explain the difference between a DMUX and MUX.
(*PTU, Dec 2008*)

19. Draw a diagram and explain 1 to 16 Demultiplexer circuit.
(*Anna University, May /Jun. 2006*)

20. Write the truth table of 4:1 multiplexer.
(*Anna University, Apr. /May 2008*)

21. Obtain 1 × 32 Demux using suitable number of 1 × 16 Demux.
(*Nagpur University, 2008*)

22. Design a 5 to 32 line decoder using two 4-to-16 line decoders.
(*Nagpur University, 2008*)

23. What is an encoder? What are its uses?
(*Mahatma Gandhi University, Jan. 2007*)

24. Given a 3 × 8 decoder D, show the construction of 4 × 16 decoder using it.
(*Mahatma Gandhi University, Dec. 2007*)

25. With the aid of block diagrams clearly distinguish between a decoder and encoder.
(*VTU, Jan. /Feb. 2004*)

26. What is an encoder? Explain an 8-to-3 line encoder.
(*VTU, Jan. /Feb. 2006*)

27. Design an 8-bit comparator using 7485 ICs only.
(*Nagpur University, 2004*)

28. Write short notes on Multiplexer and Demultiplexer.
(*Gujarat Technological University, Dec. 2009*)

29. Construct a 8 : 1 multiplexer using two 4 : 1 mux and one 2 : 1 mux
(*Gauhati University, 2003*)

30. Design full adder using 8 : 1 Multiplexer.
(*Nagpur University, 2004*)

31. Design BCD to seven segment converter and draw its logic diagram.
(*Nagpur University, 2004*)

32. Design 4 × 16 Decoder using two 3 × 8 decoders. (*Nagpur University, 2004*)

33. Design full adder circuit using suitable Decoder and Gate.
(*Nagpur University, 2004*)

34. Draw a neat diagram and explain the working of 4 × 1 multiplexer.
(*Jamia Millia Islamia University, 2007*)

35. What is the role of multiplexer in the digital electronics? Explain the logic how it selects a one input among several inputs.
(*GBTU/MTU, 2007-08*)

36. Explain 3:8 line decoders. Can we produce a 7 segment display using decoder?
(*GBTU/MTU, 2007-08*)

37. Implement a full adder circuit with 3 × 8 decoder and two OR gate.

(*GBTU/MTU, 2006-07*)

38. Explain the block diagram of combinational logic circuit. Also discuss the combinational circuit design procedure.

(*GBTU/MTU, 2006-07*)

39. Define ASM action blocks.

(*GBTU/MTU, 2006-07*)

40. How do you convert the state diagram to ASM chart?

(*GBTU/MTU, 2006-07*)

41. What is the difference between flow chart and ASM chart? Also draw an ASM chart and state table for a 2-bit UP-DOWN counter having mode control input.

(*GBTU/MTU, 2009-10*)

42. What do you mean by encoder.

(*GBTU/MTU, 2009-10*)

43. Describe the decimal adder add demultiplexer.

(*GBTU/MTU, 2009-10*)

44. Construct a 5 × 32 decoder with four 3 × 8 decoders with enable and one 2 × 4 decoder.

(*GBTU/MTU, 2007-08*)

45. How does an encoder differ from a decoder? Explain briefly.

46. Why is a multiplexer sometimes called a data selector? Explain.

47. What is the function of S_0, S_1 and S_2 pins on the IC 74151 multiplexer. Explain with an example.

48. Define a demultiplexer. Draw the logic block diagram of a 1 to 32 output demultiplexer tree using a trunk with 4 output lines. Indicate the correct addressing.

(*AMIE., Winter 2001*)

49. Design a combinational circuit that accept a three-bit number and generates an output binary number equal to the squares of the input number.

(*GBTU/MTU, 2007-08*)

50. What do you mean by a decoder? Give truth table and logic diagram with one active high enable signal for

(*i*) Active high output 2 to 4 line decoder
(*ii*) Active low output 2 to 4 line decoder

(*GBTU/MTU, 2003-04*)

51. How will you use ROM as decoder? Justify the answer.

(*GBTU/MTU, 2006-07*)

52. Design a parity generator to generate odd parity bit for four bit word.

(*Nagpur University, 2004*)

53. Discuss design and analysis of combinational circuit.

(*RGTU., Dec. 2010*)

54. Explain multiplexer and demultiplexer.

(*RGTU., Dec. 2010*)

55. What is meant by multiplexer ? Explain with diagram and truth table the operation of 4-to 1 line multiplexer.

(*GTU., Dec. 2011*)

56. What is meant by decoder ? Explain 3 to 8 line decoder with diagram and truth table.

(*GTU., Dec. 2011*)

TUTORIAL PROBLEMS

1. A certain application requires that a 5-bit number to be decoded. Use 74154 (4-line-to-16-line) decoder to implement the logic. The binary number is represented by $A_4 A_3 A_2 A_1 A_0$.

2. Fig. 9-100 shows the waveforms applied at the A, B, C and D inputs of 7442 (BCD-to - Decimal) decoder. Sketch the output waveforms.

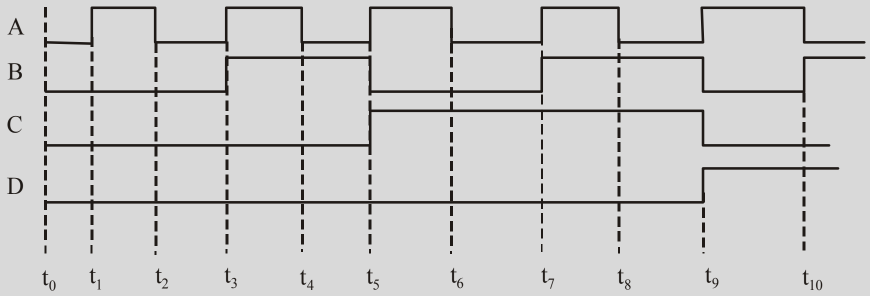


Fig. 9-100.

3. The timing diagram shown in Fig. 9-101 is applied to a two-input multiplexer. Sketch the output waveform.

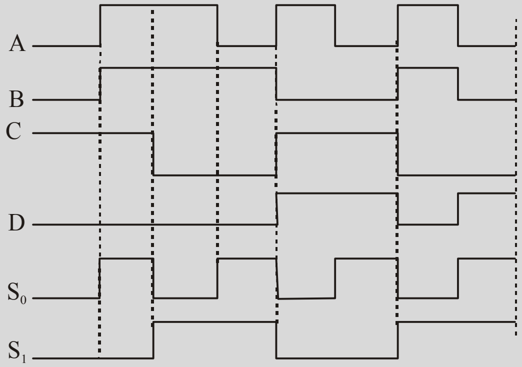


Fig. 9-101.

4. Fig. 9-102 shows a logic circuit for a multipexer. Its three inputs I_1, I_2 and I_3 are HIGH (1), while I_0 is 'C'. Determine the Boolean expression for the circuit output.

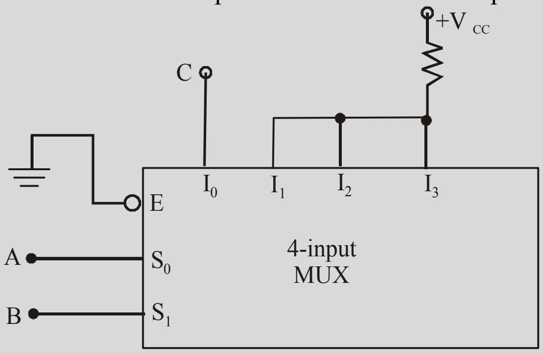


Fig. 9-102.

(*Civil Services EE. 1997*)

(**Ans.** A + B + C)

5. Determine the logic circuit realized by the circuit shown in Fig 9-103

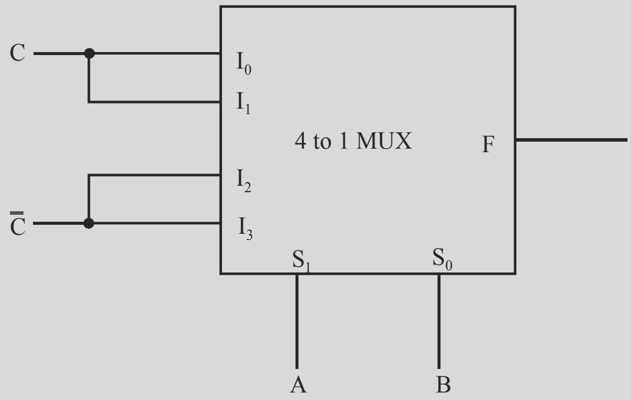


Fig. 9-103.

(*UPSC Engg. services 1999*)

(**Ans.** $C\bar{A}+\bar{C}A$)

6. Determine the function 'F' implemented by the multiplexer chip shown in Fig. 9-104

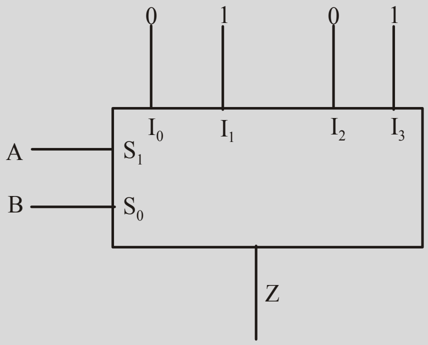


Fig. 9-104.

(*UPSC Engg. Services 1998*)

(**Ans.** B)

7. You are given (*i*) a seven-segment display with anodes connected together and (*ii*) a BCD-to-7-segment decoder with 4 input lines D, C, B and A and 7 output lines a, b, c, d, e, f and g.

(*a*) Draw a truth table showing decimal numbers and their binary codes and the state of each segment.

(*b*) Write the logic expression in terms of the inputs DCBA when the segment 'a' will be 'OFF'.

(*AMIE., Summer* 2000)

8. Fig. 9-105 shows an 8-bit magnitude comparator with logic levels specified for all inputs. Determine the outputs of the magnitude comparator.

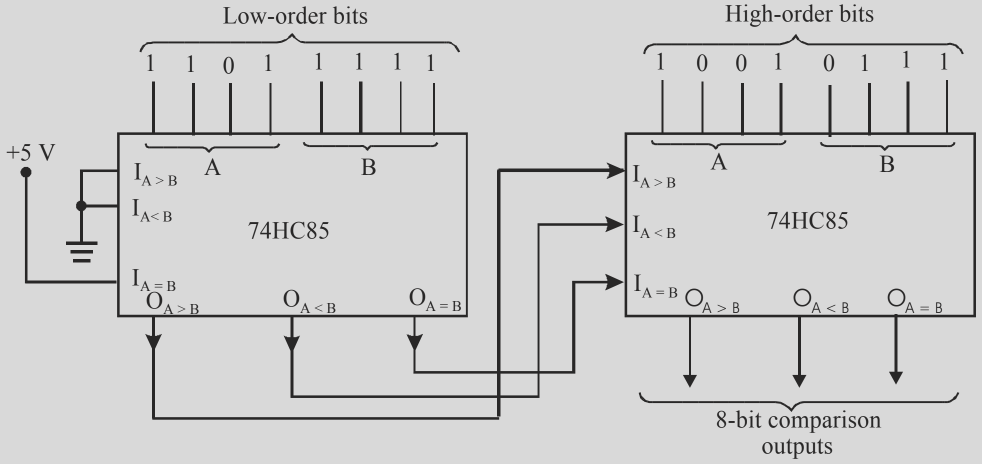


Fig. 9-105.

9. Fig. 9-106 shows 74 HC 85 4-bit magnitude comparator with its input logic levels specified. Determine the output logic levels. What information does the diagram tell us about any previous stages of the cascaded comparator.

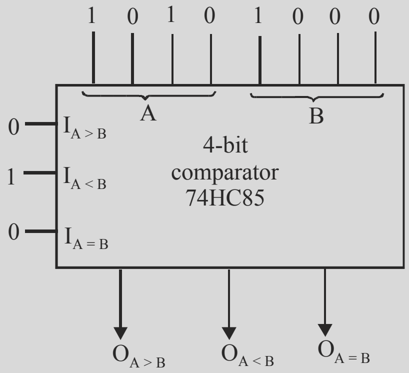


Fig. 9-106.

MULTIPLE CHOICE QUESTIONS

1. Which of the following device has more number of inputs than outputs?

(*a*) encoder (*b*) decoder

(*c*) multiplexer (*d*) none of these

2. Which of the following device is used to convert key actuations to a binary code?

(*a*) decoder (*b*) multiplexer (*c*) encoder (*d*) none of these

3. Which of the following device allows only one output to be activated at one time?

(*a*) multiplexer (*b*) decoder/demultiplexer

(*c*) encoder (*d*) none of these

4. Which of the following device can be used to interface a BCD input to an LED display?

(*a*) encoder (*b*) decoder

(*c*) BCD-to-7-segment decoder (*d*) multiplexer

5. In general, a multiplexer has

(*a*) one data input, several data outputs and selection inputs

(*b*) one data input, one data output and one selection input

(*c*) several data inputs, several data outputs and selection inputs

(*d*) several data inputs, one data output and selection inputs.

6. Data selectors are basically the same as

(*a*) decoders (*b*) demultiplexers (*c*) multiplexers (*d*) encoders

7. Which one of the following devices has more inputs than outputs?

(*a*) decoder (*b*) encoder (*c*) multiplexer (*d*) demultiplexer

8. Which one of the following devices can be used to generate arbitrary logic functions?

(*a*) decoder (*b*) encoder (*c*) multiplexer (*d*) demultiplexer

9. The following device selects one of the several inputs and transmits to a single output :

(*a*) decoder (*b*) multiplexer (*c*) demultiplexer (*d*) counter

(*AMIE., Summer* 2002)

ANSWERS

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

10

D/A AND A/D CONVERTERS

OUTLINE

Objectives

After completing this chapter, you should be able to

- ❍ understand the digital-to-analog (D/A) conversion
- ❍ calculate the resolution (or step-size) of a D/A converter
- ❍ list the various types of D/A conversion methods
- ❍ understand the operation of the binary-weighted D/A conversion method
- ❍ explain the operation of the R-2R ladder D/A conversion method

- describe the analog-to-digital (A/D) conversion method
- list the various types of A/D conversion methods
- explain the operation of digital-ramp A/D converter
- describe the operation of successive approximation A/D converter
- know the operation of sample and hold circuit
- understand the application of A/D and D/A converters in digital storage oscilloscopes, digital signal processors and analog multiplexing circuits

10-1. Introduction

A DVD player.

Usually all the portable DVD players have the following information printed on the body, “2-bit Dual D/A converter”. You may be curious to know the reason for 1-bit Dual D/A converter. As a matter of fact, in a DVD (or any other digital recording technology), the goal is to create a recording with very high fidelity and perfect reproduction. Fidelity here means very high similarity between the original signal and the reproduced signal and the perfect reproduction means the recording sounds the same every time you play it, no matter how many times you play it. To accomplish these two goals, namely, very high fidelity and perfect reproduction, digital recording converts the analog wave into a stream of numbers and records the numbers instead of the wave. The conversion of the analog wave into numbers is done by a device called an *analog-to-digital converter* (abbreviated as A/D converter or ADC). Then to play back the music, the stream of numbers is converted back to an analog wave by a digital-to-analog converter (abbreviated as D/A converter or DAC). The analog wave produced by the DAC is amplified and fed to the speakers to produce the sound.

Similarly, many of the sensors used to monitor the real* world industry applications produce analog or continuous voltage signals. These signals must be converted to a digital form before the computer can manipulate them. The device used to perform this operation is an A/D converter. There is a similar device which is used to convert the digital data from the computer to an analog form. This device is known as a D/A converter. Let us study the behaviour, construction and their applications in detail. We shall study D/A converter first, followed by A/D converter.

10-2. Digital-To-Analog (D/A) Conversion

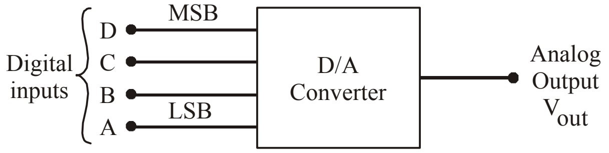
Most of the D/A and A/D converters utilize the D/A conversion process. Therefore we will study the D/A conversion first.

Basically, D/A conversion is the process of taking a value represented in digital code (such as BCD or a straight binary number) and converting it to a voltage or current that is proportional to the digital value.

Fig. 10-1(*a*) shows the symbol for a typical 4-bit D/A converter. We will not worry about the internal circuitry of the D/A converter at present. Rather let us focus on the relationship between various inputs and outputs. As seen from this figure, it has 4 digital inputs represented by A, B, C and D respectively. These outputs are obtained from the output register of a digital system. We know that these 4 inputs produce $2^4 = 16$ different binary numbers as shown in Fig. 10-1(*b*). Notice that for each digital input number the D/A converter output voltage is a unique value. In this case, the analog output voltage (V_{out}) is equal in volts to the binary number. In actual practice, it could have been twice

* The real world industry applications require variables such as temperature, pressure, light intensity, audio signals, position rotation speed, flow rate etc. to be measured for process control.

the binary number or any other value. In fact, the same idea would hold true if the D/A output were a current (I_{out}) instead of voltage.



(*a*)

D	C	B	A	V_{out} (Volts)
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
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

(*b*)

Fig. 10-1.

In general,

$$\text{Analog output} = K \times \text{Digital input}$$

where K = Proportionality factor. It is a constant value for a given D/A converter. Its value will be in voltage units when the D/A output is a voltage, and in current units when the D/A output is a current.

For a D/A converter in the present case, the value of K = 1 volt, therefore,

$$\begin{aligned} V_{out} &= K \times \text{Digital input} \\ &= (1) \times \text{Digital input} \qquad \ldots(i) \end{aligned}$$

We can use equation (*i*) to calculate V_{out} for any value of digital input. For example, with a digital input of 1010 (decimal 12), the output voltage,

$$V_{out} = 1 \times 10 = 10\text{V}$$

Example 10-1. *A 4-bit D/A converter gives an output voltage of 4.5V for an input code of 1001. Determine the output voltage for an input code of 0110.* (*U.P.S.C. Engg. Services, 1996*)

Solution. Given: Output voltage = 4.5V for an input code of 1001.

We know that the decimal equivalent of binary 1001 is 9. Therefore the output voltage of D/A converter,

$$\begin{aligned} 4.5 &= K \times \text{Digital input} \\ &= K \times 9 \qquad \ldots(i) \end{aligned}$$

where K = Proportionality constant.

From equation (*i*) we find that,

$$K = \frac{4.5}{9} = 0.5$$

We also know that the decimal equivalent of binary 0110 is 6. Therefore the output voltage of D/A converter is,

$$\begin{aligned} V_{out} &= K \times \text{Digital input} \\ &= 0.5 \times 6 = 3\text{V} \end{aligned}$$ **Ans.**

Example 10-2. *A five-bit D/A converter has a voltage output. For a digital input of 10100, an output voltage 10V is produced. What will V_{out} be for a digital input of 11101?*

Solution. Given: A D/A converter with digital input of 10100 producing an output voltage 10V.

We know that a digital input of 10100_2 is equivalent to 20_{10}. Now since V_{out} = 10 V for this case, therefore the proportional factor (K) can be determined by using the equation,

$$10\text{V} = \text{K} \times \text{Digital input}$$
$$= \text{K} \times 20_{10}$$
$$\therefore \qquad \text{K} = \frac{10}{20_{10}} = 0.5$$

$\therefore$ The output voltage for a digital input of 11101 (equivalent to 29_{10}) is,

$$V_{out} = \text{K} \times 20_{10} = (0.5\text{V}) \times 29_{10} = 14.5\text{V} \textbf{ Ans.}$$

Example 10-3. *What is the largest value of output voltage from an eight-bit D/A converter that produces 1V for a digital input of 00110010?*

Solution. Given: An 8-bit D/A converter that produces 1V for a digital input of binary 00110010. We know that the decimal equivalent of binary 00110010 is 50_{10}. Therefore, the output voltage of D/A converter, (V_{out}),

$$1\text{V} = \text{K} \times 50_{10}$$
$$\therefore \qquad \text{K} = \frac{1}{50_{10}} = 0.02\text{V} = 20 \text{ mV}$$

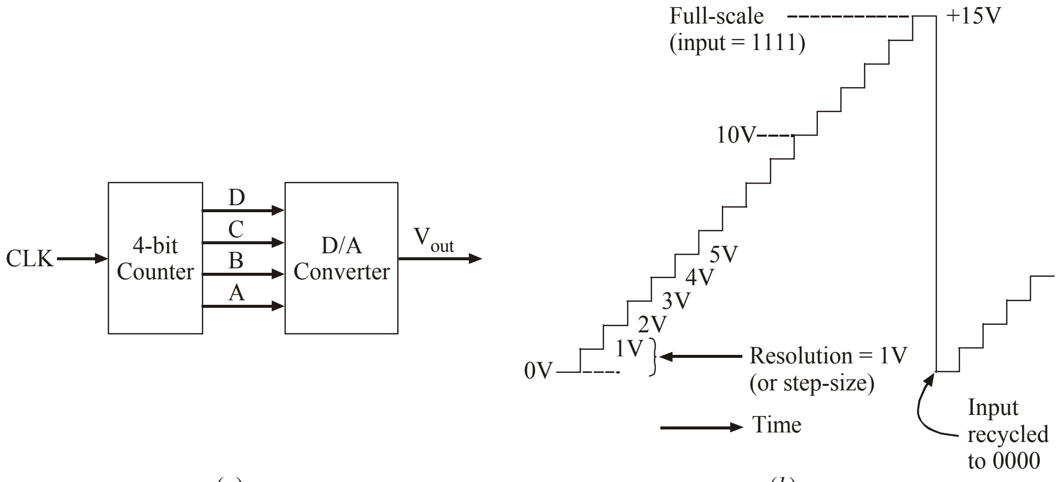
We also know that the largest value of an output voltage will correspond to the maximum value of digital input (*i.e.* the 8-bit binary input). The maximum value of 8-bit binary is 11111111. This corresponds to 255_{10}.

Hence the largest value of an output voltage from an 8-bit D/A converter,

$$V_{out} = \text{K} \times \text{Max. Digital input}$$
$$= (0.02\text{V}) \times 255_{10}$$
$$= 5.10 \text{ V} \textbf{ Ans.}$$

10-3. Resolution (or Step-Size) of a D/A Converter

It is defined as the smallest change that can occur in the analog output as a result of a change in the digital input. The resolution is always equal to the weight of the LSB (least-significant bit). It is also called the step size because it is the amount by which the V_{out} of a D/A converter will change as



(*a*) (*b*)

Fig. 10-2.

the digital input value is changed from one step to the next. It order to illustrate this point further refer to the diagram shown in Fig. 10-2 (*a*) and (*b*).

Fig. 10-2(*a*) shows a 4-bit counter connected to a D/A converter. The function of the counter is to provide the inputs to D/A converter. As the counter is continually cycled through its 16 states by the clock signal, the D/A converter output produces a *staircase* waveform as shown in Fig. 10-2(*b*). Notice that the staircase waveform goes up 1V per step. When the counter reaches to its maximum count value 1111, the D/A output is at its maximum value of 15V (full-scale output). When the counter recycles to 0000, the D/A converter output returns to 0V. The resolution (or step-size) is the size of the jumps in the staircase waveform. In the present case, each step is 1V.

It may also be carefully noted that the staircase has 16 levels corresponding to the 16 input states but there are only 15 steps (or jumps) between the 0V level and full-scale. In general, for an N-bit D/A converter,

The number of different levels, $= 2^N$

and the number of steps $= 2^N - 1$

Thus an 8-bit D/A converter has 2^8 (= 256 different levels) and 255 (= 256 – 1) different steps. Similarly, a 10-bit D/A converter has 2^{10} (= 1024 different levels) and 1023 (= 1024 – 1) different steps.

It is easy to check that resolution (or step-size) of a D/A converter is the same as the proportionality factor (K) in the input/output relationship, *i.e.*,

$$\text{Analog output} = \text{K} \times \text{Digital input.}$$

The above equation can be expressed in a different way by observing that the digital input of a D/A converter is equal to the number of steps, the proportionality factor (K) is the amount of voltage (or current) per step and the analog output is the product of the above two quantities, *i.e.*,

$$\text{Analog output} = \text{Resolution} \times \text{Number of steps}$$

or Analog full-scale output = Resolution × $(2^N - 1)$

$$\therefore \quad \text{Resolution} = \frac{\text{Analog full-scale output}}{2^N - 1}$$

Example 10-4. *What is the resolution in volts of a 10-bit D/A converter whose full-scale output is 5V?*

Solution. Number of bits, N = 10, Full-scale output = 5V.

We know that,

$$\text{resolution} = \frac{\text{Analog full-scale output}}{2^N - 1}$$

$$= \frac{5\text{V}}{2^{10} - 1} = \frac{5\text{V}}{1023} = 0.0049\text{V}$$

$$= 4.9 \text{ mV } \textbf{Ans.}$$

Example 10-5. *How many bits are required for D/A converter so that its full-scale output is 10 mA and its resolution is less than 40 μA?*

Solution. Given: Full-scale output = 10 mA = 10×10^{-3}; Resolution = 40μA = 40×10^{-6}A.

Let N = number of bits to produce resolution less than 40 μA.

We know that resolution,

$$40 \times 10^{-6} = \frac{\text{Full-scale output}}{2^N - 1} = \frac{10 \times 10^{-3}}{2^N - 1}$$

$$\therefore \quad 2^N - 1 = \frac{10 \times 10^{-3}}{40 \times 10^{-6}} = 250$$

$$\text{or} \quad 2^N = 250 + 1 = 251$$

Taking logarithms both sides, we get

$$\log_{10}(2^N) = \log_{10} 251$$

or $$N \log_{10} 2 = \log_{10} 251 \qquad \ldots(\because \log m^n = n \log m)$$

$$\therefore \quad N = \frac{\log_{10} 251}{\log_{10} 2} = \frac{2.4}{0.3010} = 7.97 \approx 8. \quad \ldots\text{(Use log function on the calculator)}$$

Therefore we should choose a D/A converter *with not less than 8 bits* to produce a resolution less than 40 μA. **Ans.**

Example 10-6. *Assuming a 12-bit D/A converter with perfect accuracy, how close to 250 rpm can the motor speed be adjusted in a digital control system shown in Fig. 10-3? What is the resolution of the D/A converter?*

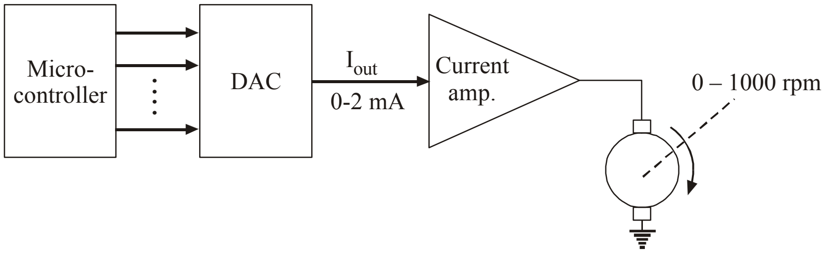


Fig. 10-3.

Solution. Given: The number of bits, N = 12. Full-scale output = 2 mA = 2 × 10^{-3}A; Motor speed = 0 to 1000 rpm.

Resolution of D/A Converter

With 12 bits, there are $(2^{12} - 1) = 4095$ steps. Thus the motor speed will go up in steps of 1000 rpm/4095 = 0.244 rpm. The number of steps needed to reach 250 rpm is 250/0.244 = 1024.59. Notice that it is not the whole number of steps. So we can round it to 1025. The actual motor speed on the 1025th step will be 250.1 rpm which is quite accurate (≈ 0.04%) **Ans.**

Motor Speed

We know that the resolution,

$$\text{Resolution} = \frac{\text{Full-scale output}}{2^N - 1} = \frac{10\text{mA}}{2^{12} - 1} = \frac{10\text{mA}}{4096 - 1}$$

$$= 2.44 \times 10^{-3}\text{mA or } 2.44\ \mu\text{A. } \textbf{Ans.}$$

Example 10-7. *Fig. 10-4 shows a microcontroller controlling the speed of a industrial robot motor. The microcontroller generates a digital number for a D/A converter which in turn produces an analog value of current in the range of 0-2 mA. The current amplifier boosts up the magnitude of current which drives the motor winding at a speed ranging from 0 to 1000 rpm (i.e. revolutions per minute).*

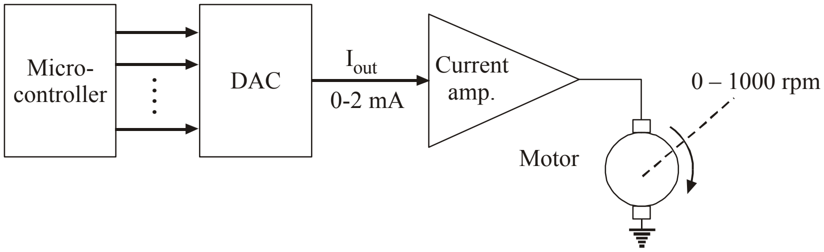


Fig. 10-4.

How many bits should be used if the microcontroller is to be able to produce a motor speed that is within 2 rpm of the desired speed?

Solution. Given: The full-scale output of D/A converter = 2 mA. The max. speed of the motor = 1000 rpm.

We know that the motor speed will range from 0 to 1000 rpm as the D/A converter goes from zero to full scale. Each step in the D/A converter output will produce a step in the motor speed. Since we want the step size to be no greater than 2 rpm, therefore we need 1000/2 (= 500) steps. Now in order to determine the number of bits of D/A converter, we know that if N is the number of bits, then number of steps,

$$= 2^N - 1$$

In order to meet the required condition,

$$2^N - 1 \geq 500$$

or

$$2^N \geq 500 + 1$$

$$\geq 501$$

Since $2^8 = 256$ and $2^9 = 512$, therefore the smallest number of bits that will produce at least 500 steps is 9. It may be noted that we could use more than 9 bits (say 10-bit or 12-bit D/A converter). But this will add to the cost of D/A converter and ultimately the entire electronic system will be more costlier. A good engineer should be full aware of economic considerations.

10-4. Percentage Resolution of D/A Converter

We have already mentioned in the last article that resolution of a D/A converter is the smallest change that can occur in the analog output as a result of a change in the digital input. Sometimes it is convenient to express it as a percentage of the full-scale output. For example, if a D/A converter has a full-scale output of 15V when the digital input is 1111 (= 15_{10}), then the step-size is 1V. This gives a percentage resolution of,

$$\% \text{ resolution} = \frac{\text{Step-size}}{\text{Full-scale output}} \times 100\%$$

$$= \frac{1\text{V}}{15\text{V}} \times 100\% = 6.67\%$$

For example, if an 8-bit D/A converter has a step-size of 10 mV, then there will be $2^8 - 1 = 256 - 1 = 255$ steps of 10 mV each. The full-scale output will be,

$$= 255 \times 10 \text{ mV} = 2550 \text{ mV or } 2.55 \text{ V.}$$

∴ Percentage resolution,

$$= \frac{10\text{mV}}{2.55\text{V}} \times 100\% \approx 0.39\%$$

Notice that if we use a 10-bit D/A converter with a same step-size (*i.e.* 10 mV), the percentage resolution,

$$= \frac{10\text{mV}}{(2^{10} - 1) \times 10 \text{ mV}} \times 100 \approx 0.1\%$$

It is evident from the above discussion that percentage resolution becomes smaller as the number of input bits is increased. As a matter of fact, the percentage resolution can also be calculated by using the equation,

$$\% \text{ resolution} = \frac{1}{\text{total number of steps}} \times 100\%$$

Thus for an 8-bit converter, the percentage resolution,

$$\% \text{ resolution} = \frac{1}{2^8 - 1} \times 100\%$$

$$= \frac{1}{255} \times 100\% \approx 0.39\%$$

and for a 10-bit converter,

$$\% \text{ resolution} = \frac{1}{2^{10} - 1} \times 100\%$$

$$= \frac{1}{1023} \times 100\% \approx 0.1\%$$

This means that it is only the number of bits that determines the percentage resolution of a D/A converter. Increasing the number of bits increases the number of steps to reach full scale, so that each step is a smaller part of the full-scale voltage. Because of this reason most manufacturers of D/A converters specify resolution as number of bits.

Example 10-8. *A 10-bit D/A converter has a step-size of 10 mV. Determine the full-scale output voltage and the percentage resolution.*

Solution. Given: Number of bits, N = 10, step-size = 10 mV.

Full-scale output voltage

We know that the resolution (or step-size) of D/A converter,

$$K = 10 \text{ mV} = 0.01 \text{ V}$$

and the number of steps,

$$= 2^N - 1 = 2^{10} - 1 = 1023$$

We also know that the resolution of a D/A converter,

$$0.01 = \frac{\text{Analog full-scale output}}{2^N - 1} = \frac{\text{Analog full-scale output}}{1023}$$

Percentage Resolution

∴ Analog full-scale output,

$$= 0.01 \times 1023 = 10.23 \text{ V}$$ **Ans.**

We know that percentage resolution,

$$\% \text{ resolution} = \frac{1}{2^N - 1} \times 100\% = \frac{1}{1023} \times 100\% = 0.098\%$$ **Ans.**

Example 10-9. *A certain 12-bit BCD digital-to-analog converter has a full-scale output of 9.99V. Determine (a) the percentage resolution and (b) converter's step-size.*

Solution. Given: Number of input bits, N = 12. Analog full-scale output = 9.99V.

(*a*) Percentage Resolution

We know that percentage resolution of a D/A converter,

$$\% \text{ resolution} = \frac{\text{Analog full-scale output}}{\text{Number of steps}} \times 100\%$$

$$= \frac{9.99}{2^N - 1} \times 100\% = \frac{9.99}{2^{12} - 1} \times 100\% = 0.24\%$$ **Ans.**

(*b*) D/A Converter's step-size

We know that D/A converter's percentage resolution,

$$0.24\% = \frac{\text{Step-size}}{\text{Analog full-scale output}} \times 100\%$$

$$= \frac{\text{Step-size}}{9.99} \times 100\%$$

$$\therefore \quad \text{Step-size} = \frac{0.24\% \times 9.99}{100\%}$$

$$\approx 0.024\text{V or } 24 \text{ mV } \textbf{Ans.}$$

10-5. What Does Resolution of a D/A Converter Mean?

We have already mentioned in Article 10-3 that a D/A converter cannot produce a continuous range of output values. So in that sense the output is truly analog. In fact, a D/A converter produces a finite set of output values. For example, in a temperature monitoring system in modern cars, the microcontroller generates a digital output to provide an output voltage between 0 and 5V to an electronic display as well as an electronically controlled value for controlling the fan speed of the car airconditioner (in summer) and heater (in winter).

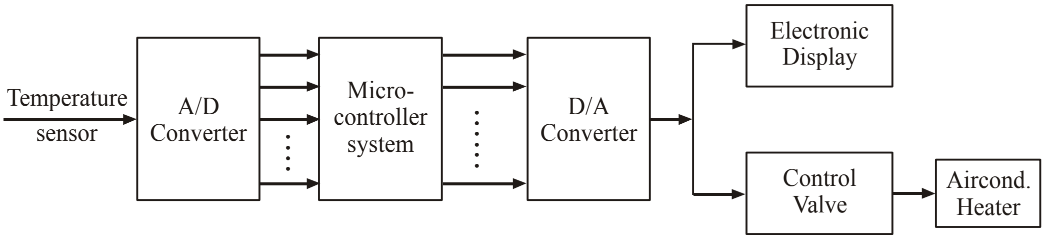


Fig. 10-5.

The resolution (*i.e.*, number of bits) of a D/A converter determines how many possible voltage values the microcontroller can send to the display/control valve. Note that if an 8-bit D/A converter is used, there will be 255 possible steps of 0.039V (39 mV) between 0 and 5V. On the other hand, if a 10-bit D/A converter is used, there will be 1023 possible steps of 0.00097V (0.97 mV) between 0 and 5V. It is evident from this discussion that greater the number of bits, the finer the resolution, *i.e.* smaller the step-size.

The digit system design engineer must decide what resolution is needed on the basis of required system performance. The resolution (or the step-size) limits how close the D/A converter output can approach to a desired analog valve. Generally, the cost of D/A converter increases with the number of bits. Hence the system design engineer will use a D/A converter with optimum number of bits.

10-6. D/A Converter Output : Analog or Digital

Strictly speaking, the output of a D/A converter is not an analog quantity. It is due to the fact that the output can take on only specific values such as 16 possible voltage levels as shown in Fig. 10-2(b). These values are valid as long as V_{ref} is constant. So in that sense you may argue that the output of D/A converter is actually digital. However, as we will study later that the number of different possible output values can be increased. As a result, the difference between successive values decreases as we increase the number of input bits. This approach will allow us to produce an output that is more and more like an analog quantity that varies continuously over a range of values. In other words, the output of a D/A converter is a *Pseudo-analog* quantity. Hence we will continue to refer the D/A converter output as analog, keeping at the back of our head that it is an approximation to a pure analog quantity.

10-7. D/A Converter—Input Weights

For a D/A converter shown in Fig. 10-1(*a*), each digital input contributes a different amount to the analog output. This is easily seen if we examine the cases where only one input is high (refer to Table 10-1). The contributions of each digital input are weighted according to their position in the binary number. Thus, A, which is the LSB, has a weight of 1V, B has a weight of 2V, C has a weight of 4V and D, the MSB, has the largest weight of 8V. The weights are successively doubled for each bit beginning with the LSB. Thus we can consider V_{out} to be weighted sum of the digital inputs. For example, to find V_{out} for the digital input 0101, we can add the weights of C and A bits to obtain 4V + 1V = 5V.

Table 10-1

D	C	B	A		V_{out} (Volts)
0	0	0	1	⟶	1
0	0	1	0	⟶	2
0	1	0	0	⟶	4
1	0	0	0	⟶	8

Example 10-10. *A 5-bit D/A converter produces V_{out} = 0.2V for a digital input of 00001. (a) Find the value of V_{out} for an input of 11111.*

Solution. Given: A 5-bit D/A converter that produces V_{out} = 0.2 V for a digital input of 00001.

We know that the binary number 00001 is equal to decimal 1. Therefore, the output voltage, V_{out},

$$0.2\text{ V} = \text{K} \times \text{Digital input}$$

$$= \text{K} \times 1$$

$$\therefore \qquad \text{K} = 0.2\text{V}$$

We also know that the binary number 11111 is equal to decimal 31. Therefore the output voltage,

$$V_{out} = \text{K} \times \text{Digital input}$$

$$= 0.2\text{V} \times 31 = 6.2\text{V} \textbf{ Ans.}$$

Alternatively, we know that 0.2V represents the weight of LSB (least-significant bit) of the D/A converter. Thus the weights of other 4 bits are 0.4V, 0.8V, 1.6V and 3.2V respectively. Thus the output voltage, V_{out}, for a digital input of 11111 is,

$$V_{out} = 3.2\text{V} + 1.6\text{V} + 0.8\text{V} + 0.4\text{V} + 0.2\text{V}$$

$$= 6.2\text{V} \textbf{ Ans.}$$

10-8. D/A Converter with Bipolar Inputs

So far we have assumed that the binary input to a D/A converter has been an unsigned number and the D/A converter output has been a positive voltage or current. However, some D/A converters are designed to produce both positive and negative values, such as –5V to +5V or –10V to +10V. This is generally done by using the binary input as the signed number with the most-significant bit (MSB) as the sign bit. Recall that we use 0 for representing + and 1 for '–' numbers. In most of D/A converters negative input values are often represented in 2's complement form. However, in same D/A converters, the negative values are represented in true-magnitude form also.

For example, suppose we have an 8-bit D/A converter that uses the 2's complement system and has a resolution of 0.0392V (or 39.2 mV). The binary input values range from 10000000 (–128) to 01111111 (+127) to produce analog outputs in the range of –5V to +5V. There are 255 steps (2^8–1) of 39.2 mV between these negative and positive limits.

Example 10-11. *The control of a positioning device may be achieved using a *servomotor. Fig. 10-6 shows a simple servo-controlled system that is controlled by a digital input that could be coming directly from a computer or from an output medium such as magnetic tape. The lever arm is moved vertically by the servomotor. The motor rotates clockwise or counterclockwise depending on whether the voltage from the power amplifier is positive or negative. The motor stops when power amplifier output is 0.*

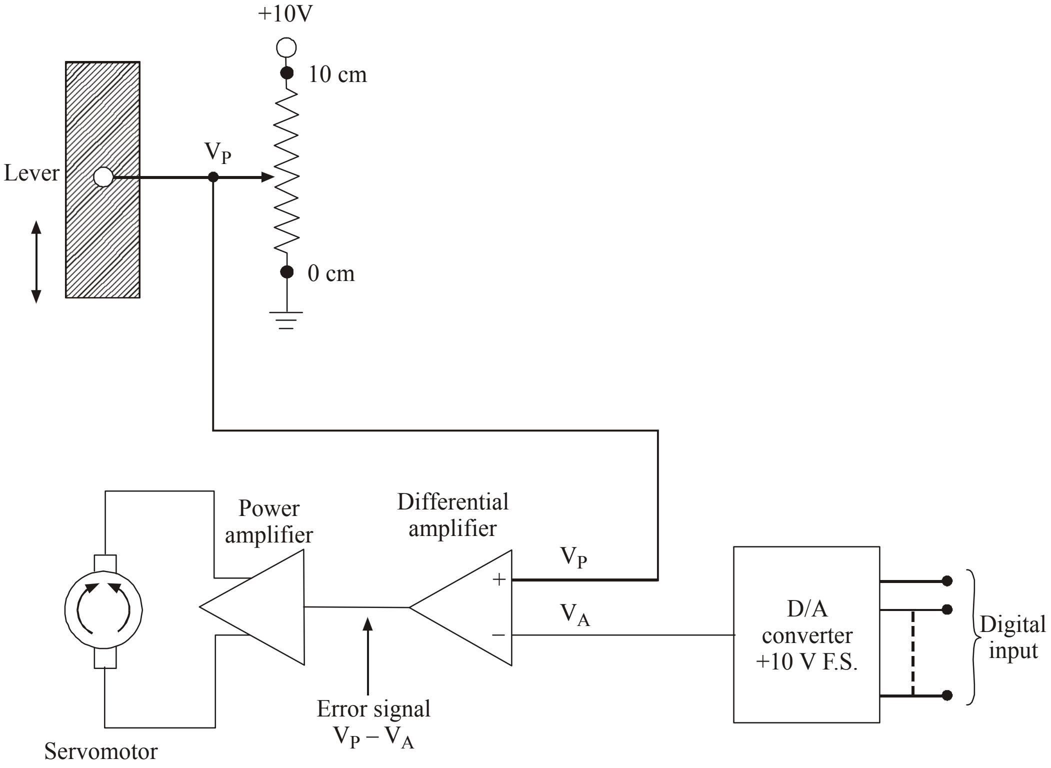


Fig. 10-6.

The mechanical position of the lever is converted to a dc voltage by the potentiometer arrangement shown in the figure. When the lever is at its 0 reference point, $V_P = 0V$. The value of V_P increases at the rate of 1V/cm until the lever is at its highest point (10 cm) and $V_P = 10V$. The desired position of the lever is provided as a digital code from the computer and is then fed to a DAC, producing V_A. The ***difference*** *between V_P and V_A (called error) is produced by the* ***differential*** *amplifier and is amplified by the power amplifier to drive the motor in the direction that causes the error signal to decrease to 0, i.e. moves the lever until $V_P = V_A$.*

(a) If it is desired to position the lever within a resolution of ***0.1 cm,*** *what is the number of bits needed in the digital input code?*

(b) In actual operation, the lever arm might oscillate slightly around the desired position, especially if ***wire-wound*** *potentiometer is used. Explain it why?*

Solution. Given: Full-scale output of D/A converter = 10V.

Resolution of the potentiometer = 0.1 cm.

*The servomotor is a motor designed to drive a mechanical device as long as an error signal exists.

(*a*) Number of bits needed

Let N-number of inputs required for D/A converter. We know that the step-size of the potentiometer

$$= \frac{1\text{V}}{1\text{ cm}} \times 0.1\text{ cm} = \mathbf{0.1V}$$

and the resolution of the D/A converter should be the same as the resolution (or step-size) of the potentiometer. Therefore resolution of D/A converter,

$$0.1\text{V} = \frac{\text{Full-scale output}}{2^{\text{N}} - 1}$$

$$= \frac{10\text{V}}{2^{\text{N}} - 1}$$

$$\therefore \qquad 2^{\text{N}} - 1 = \frac{10\text{V}}{0.1\text{V}} = 100$$

$$\text{or} \qquad 2^{\text{N}} = 101$$

If we choose N = 6; 2^{N} = 64 which is less than 101. However, if we choose N =7, 2^{N} = 128 which is greater than 101. Therefore the optimum choice for number of bits, N = 7. **Ans.**

10-9. Different Types of D/A Conversion

Strictly speaking, there are several methods and circuits for producing the D/A operation. The most common types of D/A converter's techniques are :

1. Pulse width modulator method
2. Oversampling method
3. The binary-weighted method
4. The R-2R ladder scheme
5. The thermometer coded D/A method
6. The segmented D/A method
7. Hybrid method

It is not important to be familiar with all the various circuit schemes because D/A converters are available as ICs in encapsulated packages that do not require any circuit knowledge. Rather it is more important to know the significant performance characteristics of D/A converters so that you can use them intelligently in designing electronic systems for various applications. Now we shall briefly review all the D/A conversion methods. However we will focus relatively more on the binary weighted method and the R-2R ladder scheme.

10-10. Pulse Width Modulator Method

In this method, a stable current or voltage is switched into a low pass analog filter with a duration determined by the digital input code. This method is often used for electric motor speed control, and is now becoming common in high-fidelity audio.

10-11. Oversampling Method

In this method, a circuit uses a pulse density conversion technique. The oversampling technique allows for the use of a lower resolution D/A converter internally. A simple 1-bit D/A converter is often chosen because the oversampled result is inherently linear. The D/A converter is driven with a pulse density modulated signal, created with the use of a low-pass filter, step nonlinearity (the actual 1-bit

D/A converter), and negative feedback loop, in a technique called *delta-sigma modulation*. This results in an effective high-pass filter acting on the quantization (signal processing) noise, thus steering this noise out of the low frequencies of interest into the high frequencies of little interest, which is called noise shaping (*very* high frequencies because of the oversampling). The quantization noise at these high frequencies are removed or greatly attenuated by use of an analog low-pass filter at the output (sometimes a simple RC low-pass circuit is sufficient). Most very high resolution DACs (greater than 16 bits) are of this type due to its high linearity and low cost. Speeds of greater than 100 thousand samples per second (for example, 192kHz) and resolutions of 24 bits are attainable with Delta-Sigma D/A converters.

10-12. The Binary-Weighted Method

In this method, the circuit contains one resistor or current source for each bit of the D/A converter connected to a summing point. These precise voltages or currents sum to the correct output value. This is one of the fastest conversion methods but suffers from poor accuracy because of the high precision required for each individual voltage or current. Such high-precision resistors and current-sources are expensive, so this type of converter is usually limited to 8-bit resolution or less.

Fig. 10-7 shows a 4-bit D/A converter using binary weighted conversion technique. As seen from this figure, the circuit makes use of an op-amp in the inverting amplifier configuration. It has 4 resistors with one of their ends connected to four inputs A, B, C and D and there other ends connected together to the inverting input. The op-amp acts as a summing amplifier which produces the weighted sum of its input voltages.

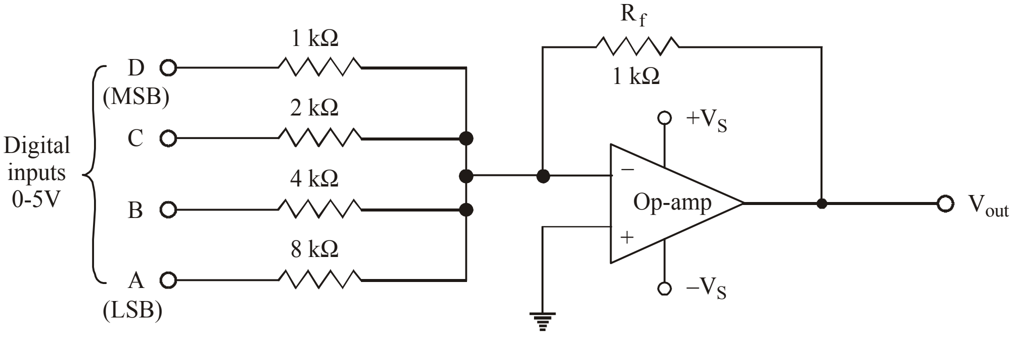


Fig. 10-7. 4-bit D/A converter employing binary weighted method.

It may be recalled that the summing amplifier multiplies each input voltage by the ratio of R_f/R_{in} (*i.e.*, the ratio of feedback resistor to the corresponding input resistor). Notice that in the present case, $R_f = 1$ kΩ and R_{in} range from 1 kΩ to 8 kΩ. The D input has $R_{in} = 1$ kΩ, so the summing amplifier passes the voltage at D with no attenuation. The C input has $R_{in} = 2$ kΩ, so this will be attenuated by 1/2. Similarly, the B input has $R_{in} = 4$ kΩ, so this will be attenuated by 1/4 and likewise the A input will be attenuated by 1/8. Thus the summing amplifier output may be expressed as :

$$V_{out} = -\left(V_D + \frac{1}{2}V_C + \frac{1}{4}V_B + \frac{1}{8}V_A\right)$$

The negative sign is there because of the 180°-phase shift introduced by op-amp being used in the inverting amplifier configuration.

Table 10-2

Input Code				V_{out} (Volts)
D	C	B	A	
0	0	0	0	0
0	0	0	1	– 0.625
0	0	1	0	– 1.250
0	0	1	1	– 1.875
0	1	0	0	– 2.500
0	1	0	1	– 3.125
0	1	1	0	– 3.750
0	1	1	1	– 4.375
1	0	0	0	– 5.000
1	0	0	1	– 5.625
1	0	1	0	– 6.250
1	0	1	1	– 6.875
1	1	0	0	– 7.500
1	1	0	1	– 8.125
1	1	1	0	– 8.750
1	1	1	1	– 9.375 ← Full-scale output voltage

Table 10-2 lists all the possible input conditions and the resultant summing amplifier output voltage. The output is evaluated for any input condition by setting the appropriate inputs to either 0 or 5V. For example if the digital input is 1101, then $V_D = V_C = V_A = 5V$ and $V_B = 0$. Then using equation, we get,

$$V_{out} = -\left(5V + \frac{1}{2} \times 5V + 0V + \frac{1}{8} \times 5V\right)$$
$$= -(5V + 2.5V + 0V + 0.625V)$$
$$= -(5V + 2.5V + 0V + 0.625V)$$
$$= -\,8.125V$$

The resolution of this 4-bit D/A converter is equal to the weighting of the least-significant bit (LSB) which is $\frac{1}{8} \times 5V$ = 0.625 V. As seen in the table, the analog output voltage increase in steps of 0.625V as the binary input number advances by one step.

It is evident from the resistor values shown in Fig. 10.7 that they are binary weighted (*i.e.*, 1,2, 4 and 8). In other words, starting with the most-significant bit (MSB) resistor, the resistor values increase by a factor of 2. This of course produces the desired weighting in the voltage output.

The values of analog output voltage (V_{out}) shown in Table 10-2 are the ideal values for the various inputs. How close the circuit shown in Fig. 10-7 comes to producing these values depends upon two factors :

1. The precision of the input and feedback resistors.
2. The precision of the input voltages.

We know that the resistors can be made very accurate (within 0.01% of the desired values) by trimming. But the input voltage levels have to be handled differently. It may be carefully noted that the digital inputs cannot be taken directly from flip-flops or logic gates. It is because of the fact that the output logic levels of these devices are not exactly 0V and 5V but vary within certain ranges. Because of this reason, it is necessary to add some more circuitry between each digital input and its input resistor to the summing amplifier as shown in Fig. 10-8.

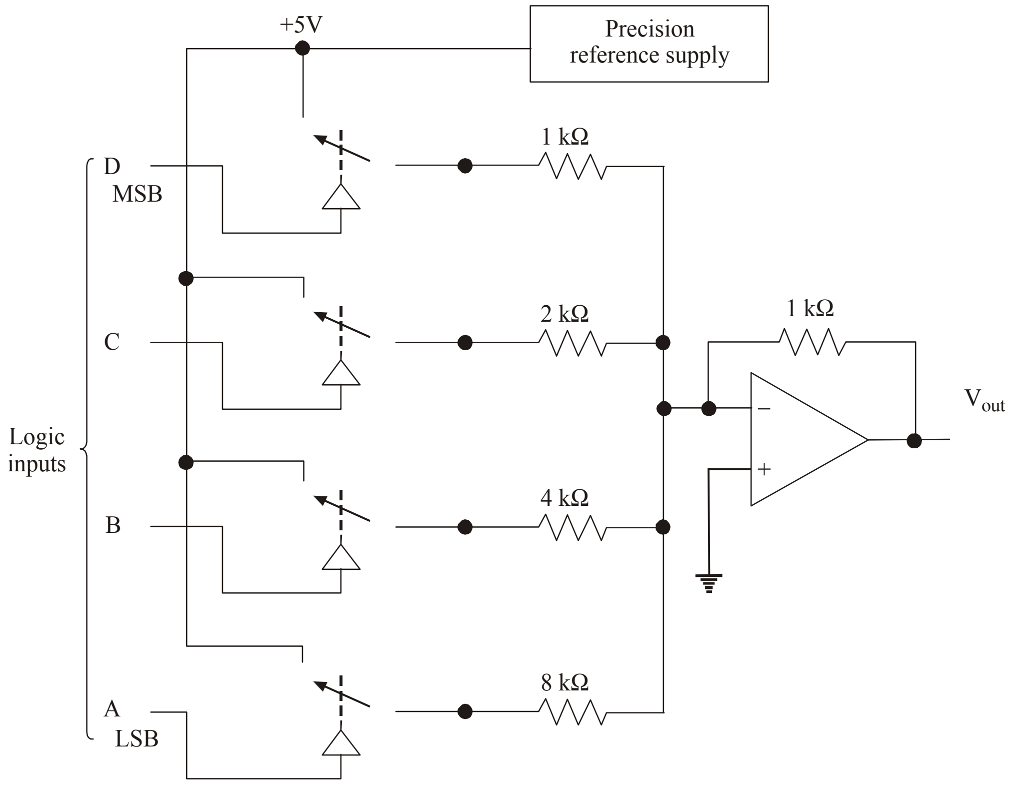


Fig. 10-8.

As seen from this figure, each digital input controls a semiconductor switch. When the input is **HIGH**, the switch closes and connects a precision reference supply to the input resistor. When the input is **LOW**, the switch is open. The reference supply produces a very stable, precise voltage needed to generate an accurate analog output.

Fig. 10-9 shows a circuit for generating an analog output current proportional to a binary input. As seen from this circuit, it is a 4-bit D/A converter with binary weighted resistors. The circuit uses four parallel current paths, each controlled by a semiconductor switch. The state of each switch is controlled by logic levels at the binary inputs. The current through each path is determined by an accurate reference voltage, V_{ref} and a precision resistor in the path. The resistors are binary weighted so that the various currents will be binary weighted and the total current (I_{out}) will be the sum of the individual currents.

Notice that the most-significant bit (MSB) path in the circuit has the smallest value of resistor R, the next path has a resistor of twice the value and so on.

Let V_{ref} = the precision reference supply voltage,

R = the value of precision resistor,

Then, I_0, i.e. the reference value of current,

$$= \frac{V_{ref.}}{R}$$

and $$I_{out} = B_3 \times I_0 + B_2 \times \frac{I_0}{2} + B_1 \times \frac{I_0}{4} + B_0 \times \frac{I_0}{8}$$

For example, if V_{ref} = 5V and R = 10 kΩ; then I_0 = 5V/10kΩ = 0.5 mA. Then if the digital input is 1010, then,

$$I_{out} = 1 \times 0.5\,\text{mA} + 0 \times \frac{0.5\text{ mA}}{2} + 1 \times \frac{0.5\text{mA}}{4} + 0 \times \frac{0.5\text{mA}}{8}$$

$$= 0.625\text{ mA.}$$

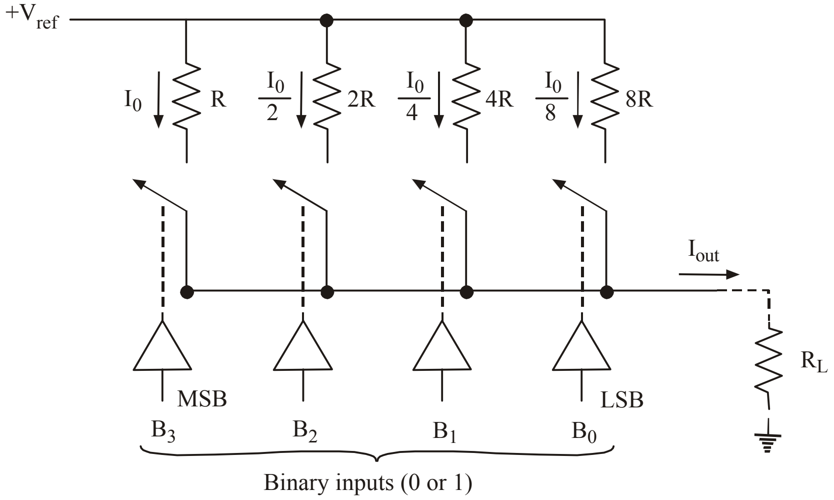


Fig. 10-9.

The output current (I_{out}) can be made to flow through a load, R_L, which is much smaller than the value of R, so that it has no effect on the value of current. Ideally speaking, R_L should be a **short** to ground.

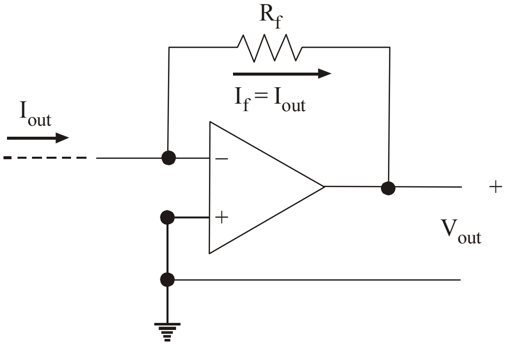


Fig. 10-10.

A more practical way is to replace R_L with an op-amp based current-to-voltage converter as shown in Fig. 10-10. Here the output current (I_{out}) from the D/A converter is connected to the op-amp's inverting input (–) terminal which is virtually at ground. The op-amp's negative feedback forces a current equal to I_{out} to flow through h_F to produce $V_{out} = -I_{out} \times R_f$. Thus V_{out} will be an analog voltage that is proportional to the binary input to the D/A converter. This analog output can drive a wide range of loads without being loaded down.

Example 10-12. *In a 4-bit weighted resistor D/A converter, the resistor value corresponding to LSB is 32 kΩ. Determine the resistor value corresponding to MSB.*

Solution. Given: a 4-bit weighted resistor D/A converter; the resistor value corresponding to LSB = 32 kΩ. Then the value of the resistor for the MSB can be determined as follows:

We know that the value of resistors in the weighted resistor D/A converter is 8R, 4R, 2R, R respectively. The value 8R is for the LSB and R for the MSB. Since the value of resistor for MSB is (1/8) × value of resistor for LSB, therefore the value of resistor for MSB is (1/8) × 32 k = 4 kΩ**Ans.**

Example 10-13. *Fig. 10-11 shows a simple D/A converter. Using an Op-Amp summing amplifier with binary weighted resistors :*

(*a*) *Determine the weight of each input bit.* (*b*) *Change* R_F *to 250Ω and determine the full-scale output.*

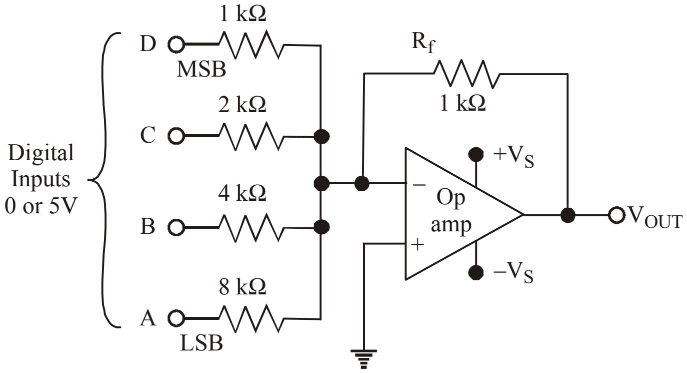


Fig. 10-11

Solution. Given $R_f = 1\ k\Omega = 1000\ \Omega$;

(*a*) Weight of each input bit

The weight of each input bit for the given values of input resistors and R_f, is determined as below. We know that, weight of input bit A (LSB),

$$= \frac{R_f}{R_r} = \frac{1\ k}{1\ k} = \frac{1}{8}$$

Weight of input bit B,

$$= \frac{R_f}{R_i} = \frac{1\ k}{4\ k} = \frac{1}{4}$$

Similarly the weight of input bit C,

$$= \frac{R_f}{R_i} = \frac{1\ k}{2\ k} = \frac{1}{2}$$

and the weight of input D (MSB)

$$= \frac{R_f}{R_i} = \frac{1\ k}{1\ k} = 1$$

(*b*) Full-scale output if R_f is changed to 250 Ω

We know that if R_f is changed from 1 kΩ to 250 Ω; the weights of every input bit will change. Thus the new weights of input bits are :

Weight of input bit A, (LSB),

$$= \frac{R_f}{R_i} = \frac{250}{8000} = \frac{1}{32}$$

and the weight of input bit B,

$$= \frac{R_f}{R_i} = \frac{250}{4000} = \frac{1}{16}$$

Similarly the weight of input bit C,

$$= \frac{R_f}{R_i} = \frac{250}{2000} = \frac{1}{8}$$

and the weight of input bit D (MSB),

$$= \frac{R_f}{R_i} = \frac{250}{1000} = \frac{1}{4}.$$

Using these values the full-scale output voltage will be,

$$V_{out} = -\left(\frac{1}{4}V_D + \frac{1}{8}V_C + \frac{1}{16}V_B + \frac{1}{32}V_A\right)$$

$$= -\left(\frac{1}{2}\times 5 + \frac{1}{8}\times 5 + \frac{1}{16}\times 5 + \frac{1}{32}\times 5\right)$$

$$= -(2.5 + 0.6250 + 0.3125 + 0.1563)$$

$$= -3.5938\text{V} \approx -3.6\text{V} \textbf{ Ans.}$$

10-13. Limitations of Binary-Weighted Method

The D/A converter based on binary-weighted resistors is required to produce proper weighting of each bit. While the method works well in theory, it has practical limitations. The major problem is the large difference in resistor values between the LSB and MSB, especially in high resolution D/A converters (*i.e.* 10-bit, 12-bit and 16-bit D/A converters). For example, if the MSB resistor is 1 kΩ in a 10-bit D/A converter, the LSB resistor will be 512 kΩ. With the current IC fabrication technology, it is very difficult to produce resistance values over a wide resistance range that maintains an accurate ratio especially with variation in temperature.

10-14. The R-2R Ladder Scheme

We have already discussed in the last article about the limitation of D/A converter using binary weighted resistors. For this reason, it is preferable to have a circuit that uses resistances that are fairly close in value. One of the most widely used circuits is the R-2R Ladder network.

In this method, the circuit uses a repeating cascaded structure of resistor values R and 2R. This improves the precision due to the relative ease of producing equal valued matched resistors (or current sources). However, wide converters perform slowly due to increasingly large RC-constants for each added R-2R link.

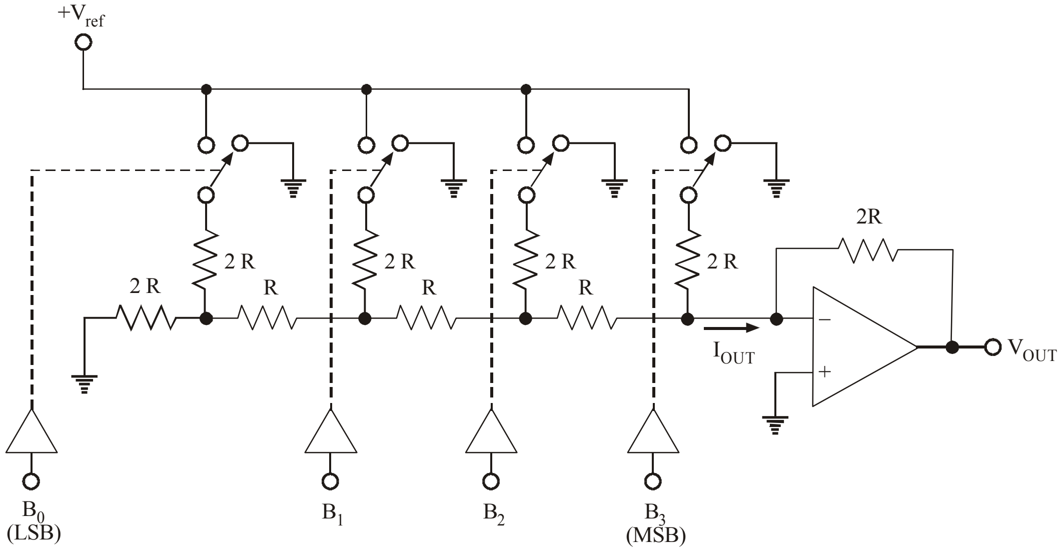


Fig. 10-12.

Fig. 10-12 shows a 4-bit D/A converter using R-2R ladder network scheme. Notice how the two values of resistorly (R and 2R) are arranged. The current output (I_{out}) depends upon the positions of the four switches. The position of the switches in turn is controlled by the binary inputs $B_3B_2B_1B_0$. The output current (I_{out}) flows through an op-amp current-to-voltage converter to develop the output voltage (V_{out}).

The value of output voltage is given by the equation,

$$V_{out} = \frac{V_{ref}}{8} \times (\text{Binary input})$$

The binary input for a 4-bit D/A converter may range from 0000 to 1111 (*i.e.* 0 to 15).

Example 10-14. *Assume that V_{ref} = 5V for the D/A converter shown in Fig. 10-12. What are the resolution and full-scale output of this converter?*

Solution. Given : V_{ref} = 5V; Number of bits of D/A converter, N = 4.

Resolution

We know that resolution of a D/A converter is equal to the weight of LSB. This can be determined by setting binary input = 0001 $\left(= \frac{1}{10}\right)$.

We also know that the output voltage,

$$V_{out} = -\frac{V_{ref}}{8} \times \text{Binary input}$$

$$= -\frac{5V}{8} \times 1 = -0.625V$$

The value of the output voltage corresponding to 0001 input is the resolution of D/A converter. Hence, resolution of D/A converter shown in Fig. 10-12, is –0.625 V **Ans.**

Full-scale output

We know that full-scale output will correspond to a binary input 1111 $(=15_{10})$. Thus the full-scale output voltage,

$$V_{out} = -\frac{V_{ref}}{8} \times \text{Binary input}$$

$$= -\frac{5V}{8} \times 15 = -9.375 \text{ V } \textbf{Ans.}$$

10-15. The Thermometer Coded D/A Method

This contains an equal resistor or current source segment for each possible value of D/A converter output. An 8-bit thermometer D/A converter would have 255 segments, and a 16-bit thermometer D/A converter would have 65,535 segments. This is perhaps the fastest and highest precision D/ A converter architecture but at the expense of high cost. Conversion speeds of higher than *1 billion samples per second* have been reached with this type of D/A converter. A further discussion on the thermometer coded D/A method is beyond the scope of this method.

10-16. The Segmented D/A Method

This method combines the thermometer coded principle for the most-significant bits and the binary weighted principle for the least-significant bits. In this way, a compromise is obtained between precision (by the use of the thermometer coded principle) and number of resistors or current sources (by the use of the binary weighted principle). The full binary weighted design means 0% segmentation, the full thermometer coded design means 100% segmentation.

10-17. Hybrid D/A Method

This method use a combination of the above techniques in a single converter. Most D/A converter integrated circuits are of this type due to the difficulty of getting low cost, high speed and high precision in one device.

10-18. An Integrated Circuit D/A Converter

The DAC 0808, a TTL IC available from National Semiconductor, is an 8-bit D/A converter that uses an R-2R ladder network. Its block symbol is shown in Fig. 10-13(*a*).

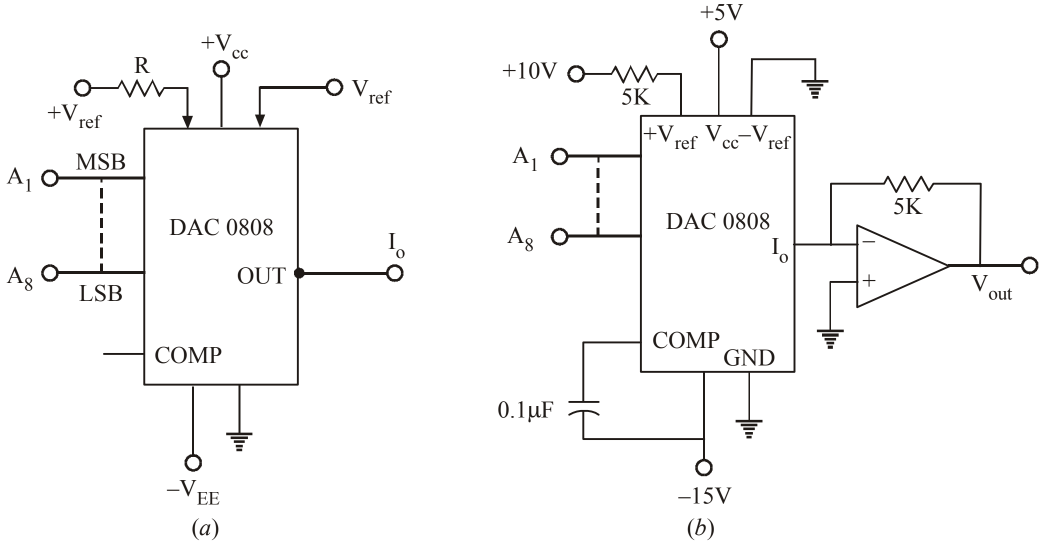


Fig. 10-13.

The device needs three supply voltages : V_{CC}, R_{ref} and V_{EE}. For normal operation, $V_{CC} = +5V$, $V_{EE} = -15V$ and $V_{ref} = +10.00V$. Notice that reference voltage need to be very precisely set to the exact value. The device produces output current

$$I_0 = \frac{V_{ref}}{R}\left(\frac{A_1}{2}+\frac{A_2}{4}+\frac{A_3}{8}+\frac{A_4}{16}+\frac{A_5}{32}+\frac{A_6}{64}+\frac{A_7}{128}+\frac{A_8}{256}\right)$$

For example, if $V_{ref} = +10V$, and the digital input is 0110 0100, then the output current,

$$I_0 = \frac{10}{5k}\left(\frac{0}{2}+\frac{1}{4}+\frac{1}{8}+\frac{0}{16}+\frac{0}{32}+\frac{1}{64}+\frac{0}{128}+\frac{0}{256}\right)$$

$$= \frac{10}{5k}\left(\frac{1}{4}+\frac{1}{8}+\frac{1}{64}\right) = \frac{10}{5k}\left(\frac{16+8+1}{64}\right) = \frac{10\times 25}{64\times 5k}$$

$$= 0.78125 \text{ mA}$$

∴ The output voltage,

$$V_0 = R_f \times I_0 = -5k \times 0.78125 \text{ mA} = -3.906V$$

Notice that the value of reference resistor (R) and the feedback resistor (R_f) both are equal to 5kΩ. This has been done purposely to keep the calculation of output current (I_0) and hence the output voltage (V_{out}) simple.

If R = R_f, you can use the following equation straightaway to calculate the output voltage,

$$V_{out} = V_{ref}\left(\frac{A_1}{2}+\frac{A_2}{4}+\frac{A_3}{8}+\frac{A_4}{16}+\frac{A_5}{32}+\frac{A_6}{64}+\frac{A_7}{128}+\frac{A_8}{256}\right)$$

where A_1 = Most significant bit of digital input and
A_8 = Least significant bit.

10-19. D/A Converter Specifications

There is a wide variety of D/A converters available in the market. In order to evaluate D/A converter for any particular application, it is important for an electronics engineer to be familiar with the specifications of D/A converters. Although a datasheet of a typic D/A converter lists a number of specifications (refer to the datasheet of DAC 0808 Fig. 10-14), yet the following are important from the subject point of view:

1. **Resolution :** This is the number of possible output levels the D/A converter is designed to reproduce. This is usually stated as the number of bits it uses. The number of output levels is given by 2^N where N is the number of bits. For instance, a 1-bit D/A is designed to produce 2 output levels while an 8-bit D/A converter is designed for 256 levels. Resolution is related to the effective number of bits. Higher the effective number of bits, higher is the resolution. A 10-bit D/A converter has a better resolution than an 8-bit D/A converter and a 12-bit D/A converter has a better resolution than a 10-bit D/A converter. However, a D/A converter with more number of bits is expensive than the one with lower number.
2. **Accuracy :** The manufacturers specify the accuracy of D/A converter in several ways. The two most common are (1) full-scale error and (2) linearity error. Both are normally expressed as a percentage of the converter's full-scale output (% F.S.).

DAC0808 8-Bit D/A Converter

General Description : The DAC0808 is an 8-bit monolithic digital-to-analog converter (DAC) featuring a full-scale output current settling time of 150 ns while dissipating only 33 mW with

Block and Connection Diagrams

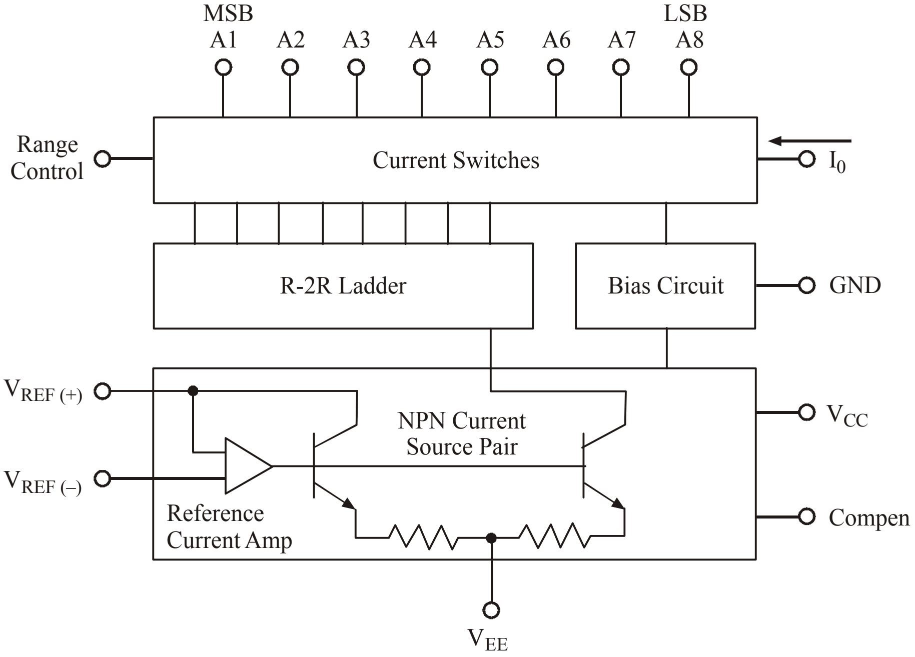
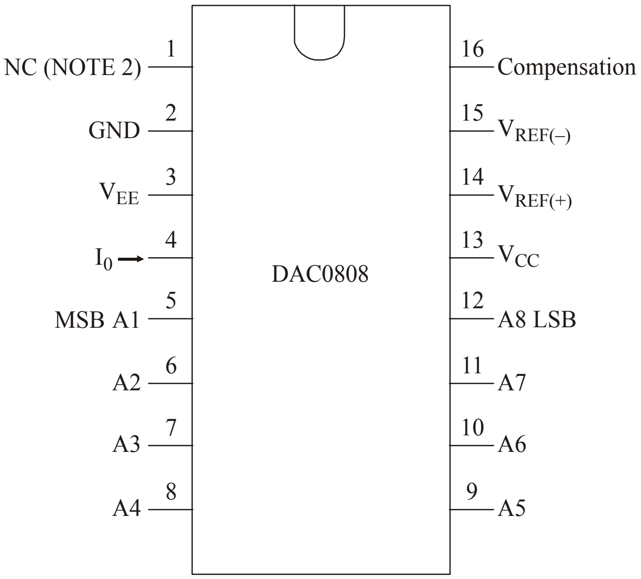


Fig. 10-14 (Contd.)

Dual-In-Line Package



Top View
Order Number DAC0808
See NS Package M16A or N16A

± 5V supplies. No reference current (I_{REF}) trimming is required for most applications since the full-scale output current is typically ± 1 LSB of 255 I_{REF}/256. Relative accuracies of better than ± 0.19% assure 8-bit monotonicity and linearity while zero level output current of less than 4 µA provides 8-bit zero accuracy for $I_{REF} \geq 2$ mA. The power supply currents of the DAC0808 is independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the MC1508/MC1408. For higher speed applications, see DAC0800 datasheet.

Features

- ❍ Relative accuracy : ± 0.19% error maximum
- ❍ Full scale current match : ± 1 LSB typ
- ❍ Fast settling time : 150 ns typ
- ❍ Noninverting digital inputs are TTL and CMOS compatible
- ❍ High speed multiplying input slew rate : 8 mA/µs
- ❍ Power supply voltage range: ± 4.5V to ± 18V
- ❍ Low power consumption : 33 mW @ ± 5V

Absolute Maximum Ratings (Note 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Power Supply Voltage	
V_{CC}	+ 18V_{DC}
V_{EE}	– 18V_{DC}
Digital Input Voltage, P_{in} 5–P_{in} 12	–10 V_{DC} to + 18 V_{DC}
Applied Output Voltage, V_O	–11 V_{DC} to + 18 V_{DC}
Reference Current, I_{14}	5 mA
Reference Amplifier Inputs, P_{in} 14, P_{in} 15	V_{CC}, V_{EE}
Power Dissipation (Note 4)	1000 mW
ESD Susceptibility (Note 5)	TBD
Storage Temperature Range	–65°C to + 150°C
Lead Temp. (Soldering, 10 seconds)	
Dual-In-Line Package (Plastic)	260°C
Dual-In-Line Package (Ceramic)	300°C
Surface Mount Package	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C

Operating Ratings

Temperature Range	$T_{MIN} \le T_A \le T_{MAX}$
DAC0808	$0 \le T_A \le + 75°C$

Electrical Characteristics

(V_{CC} = 5V, V_{EE} = –15 V_{DC}, V_{REF}/R14 = 2 mA, and all digital inputs at high logic level unless otherwise noted.)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
E_r	Relative Accuracy (Error					%
	Relative to Full Scale I_0)					
	DAC0808LC (LM 1408-8)				**±0.19**	%
	Settling Time to Within ½ LSB	T_A=25°C (Note 7),		150		ns
	(Includes t_{PLH})					
t_{PLH}, t_{PHL}	Propagation Delay Time	T_A = 25°C, (*Figure 5*)		30	100	ns
TCI_0	Output Full Scale Current Drift			±20		ppm/°C
MSB	Digital Input Logic Levels					
V_{IH}	High Level, Logic "1"		2			V_{DC}
V_{IL}	Low Level, Logic "0"				0.8	V_{DC}
MSB	Digital Input Current					
	High Level	V_{IH} = 5V		0	0.040	mA
	Low Level	V_{IL} = 0.8V		–0.003	–0.8	mA
I_{15}	Reference Input Bias Current			–1	–3	µA
	Output Current Range					
		V_{EE} = –5V	0	2.0	2.1	mA
		V_{EE} = –15V,T_A = 25°C	0	2.0	4.2	mA
I_O	Output Current	V_{REF} = 2.000V,				
		R_{14} = 1000Ω,				
			1.9	1.99	2.1	mA
	Output Current, All Bits Low			0	4	µA
	Output Voltage Compliance					
	(Note 3)	$E_r \le$ 0.19%, T_A = 25°C				
	V_{EE} = –5V, I_{REF}=1 mA				–0.55, +0.4	V_{DC}
	V_{EE} Below –10V				–5.0, +0.4	V_{DC}
SRI_{REF}	Reference Current Slew Rate		4	8		mA/µs

Electrical Characteristics (continued)
(V_{CC} = 5V, V_{EE} = –15 V_{DC}, V_{REF}/R14 = 2 mA, and all digital inputs at high logic level unless otherwise noted.)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
	Output Current Power Supply Sensitivity	$-5V \le V_{EE} \le -16.5V$		0.05	2.7	μA/V
	Power Supply Current (All Bits Low)					
I_{CC}				2.3	22	mA
I_{EE}				–4.3	–13	mA
	Power Supply Voltage Range	$T_A = 25°C$				
V_{CC}			4.5	5.0	5.5	VDC
V_{EE}			–4.5	–15	–16.5	V_{DC}
	Power Dissipation					
	All Bits Low	V_{CC}=5V, V_{EE}=–5V		33	170	mW
		V_{CC}= 5V, V_{EE}=–15V		106	305	mW
	All Bits High	V_{CC}=15V, V_{EE} =–5V		90		mW
		V_{CC}=15V, V_{EE}=–15V		160		mW

Note 2: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 3: Range control is not required.

Typical Application

$$V_0 = 10V\left(\frac{A1}{2} + \frac{A2}{4} + \ldots \frac{A8}{256}\right)$$

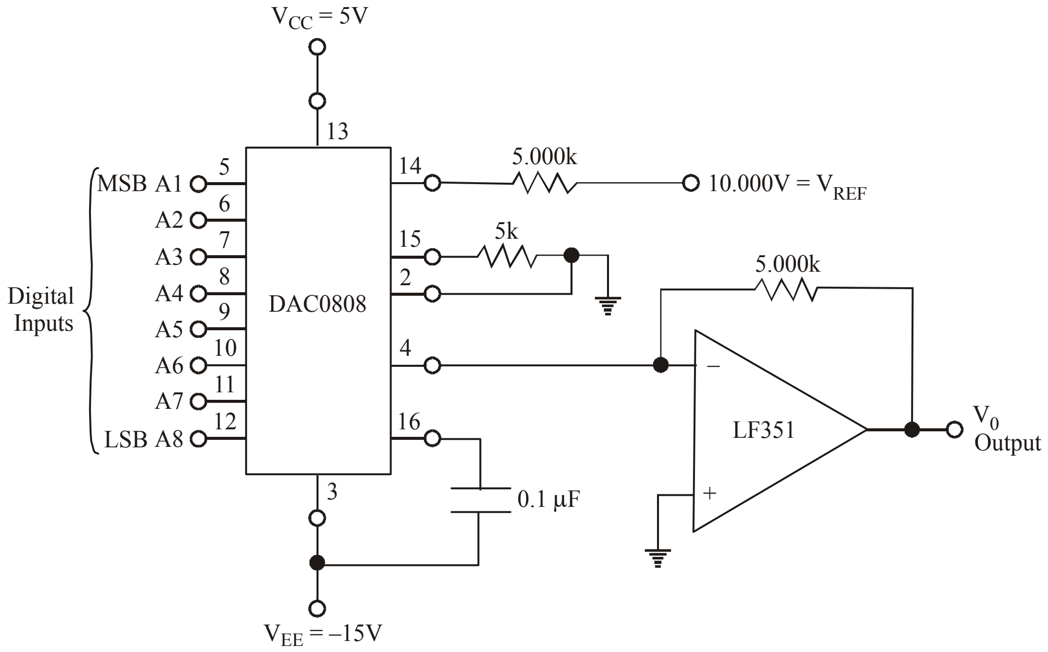


Fig. 10-15. +10V Output Digital to Analog Converter (Note 8)

Note 4: The maximum power dissipation must be derated at elevated temperatures and is dictated by T_{JMAX}, θ_{JA}, and the ambient temperature, TA. The maximum allowable power dissipation at any temperature is $P_D = (T_{JMAX} - T_A)/\theta_{JA}$ or the number given in the Absolute Maximum Ratings, whichever is lower. For this device, T_{JMAX} = 125°C, and the typical junction-to-ambient thermal resistance of the dual-in-line J package when the board mounted is 100°C/W. For the dual-in-line N package, this number increases to 175°C/W and for the small outline M package this number is 100°C/W.

Note 5: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Note 6: All current switches are tested to guarantee at least 50% of rated current.

Note 7: All bits switched.

Note 8: Pin-out numbers for the DAL080X represent the dual-in-line package. The small outline package pinout differs from the dual-in-line package.

3. **Full-scale error** is the maximum deviation of the D/A converter's output from its expected value, expressed as a percentage of full-scale. For example, DAC0808 has a maximum full-scale error of ± 0.19%. Since this converter has a typical full-scale current output of 2mA, therefore the percentage full-scale error converts to,

$$\pm 0.19\% \times 2 \text{ mA} = \pm 0.0038 \text{ mA or } \pm 3.8 \,\mu\text{A}$$

   This means that the output of DAC 0808 D/A converter can at any time be off by as much as 3.8 mA from its expected value.

4. **Linearity error** is the maximum deviation in step-size from the ideal step-size. For example, the DAC 0808 has an expected step size of 0.0078 mA (or 7.8 mA). If this converter has a linearity error of ± 0.19% F.S., this would mean that the actual step could be off as much as 7.8 mA.

   Note : It is very important to understand that accuracy and resolution of a D/A converter must be compatible. It is illogical to have a resolution of, say, 2 per cent and an accuracy of 0.02% or vice versa. In order to illustrate this point, a D/A converter with a resolution of 2% and a full-scale output of 2mA can produce an analog output current within 0.0004 mA of a desired value, assuming perfect accuracy. It makes no sense to have a costly accuracy of 0.02% of full-scale (*i.e.*, 0.004 mA) if the resolution already limits the closeness of the desired value to 0.04 mA. A similar argument can be presented for having a resolution that is very small (large no. of bits) while the accuracy is poor.

5. **Off-set error.** Ideally, the output of D/A converter will be 0 when all the inputs are 0s. However in actual practice, there will be a very small output voltage (or current) which is called an offset error. For instance, DAC 0808 has an offset error of less than 4 μA which is almost negligible. However in some D/A converters, this offset error could be significant. In such cases, if not corrected, the offset error will be added to the expected D/A output for all input cases. Many D/A converters will have an external offset adjustment that allows you to zero (or cancel) the offset. This is usually done by applying 0 input at all the inputs of D/A converter and monitoring the output while an offset adjustment potentiometer is adjusted until the output is as close to 0 as required.

6. **Maximum sampling frequency (Settling time).** This is a measurement of the maximum speed at which the D/A converter circuitry can operate and still produce the correct output. According to Shanon-Nyquist sampling theorem, a signal must be sampled at over twice the bandwidth of the desired signal. For instance, to reproduce signals in all the audible spectrum, which includes frequencies of up to 20 kHz, it is necessary to use D/A converter that operater it over 40 kHz. The CD standard samples audio at 44.1 kHz, thus D/A converters of this frequency are often used. A common frequency in cheap computer sound cards is 48 kHz — many work only at this frequency, offering the use of other sample rates only through (often poor) internal resampling.

The operating speed of a D/A converter is usually specified by giving the sampling time. The settling time is the time required for the D/A converter output to go from zero to full-scale as the binary input is changed from all 0s to all 1s. As a matter of fact, the settling time is measured as the time for D/A converter output to settle within ± ½ step size of its final value. For example, if a D/A converter has a resolution of 10 mV, settling time is measured as the time it takes the output to settle within 5 mV of it full-scale.

Typical values of settling time range from 100 ns to 10 μs. Generally speaking, the D/A converters with a current output will have shorter settling times than those with voltage outputs. For instance, the D/A converter 1280 can operate as either current output or voltage output. In the current output mode, its settling time is 300 ns whereas in the voltage output mode, its settling time is 2.5 μs. The main reason for this difference in the settling time is the response time of the Op-Amp that is used as the current-to-voltage converter.

7. **Monotonicity.** This refers to the ability of D/A converter's analog output to increase with an increase in digital code or converse. This characteristic is very important for D/A converters used as a low frequency signal source or as a digitally programmable trim element.

Example 10-15. *A D/A converter has 5V full scale output voltage and an accuracy of ± 0.2%. Determine the maximum error for any output voltage.*

(*U.P.S.C. Engg. Services 1996*)

Solution. Given: Full-scale output voltage = 5V and accuracy = ± 0.2% = ± 0.002.

We know that the maximum error for any output voltage,

$$= 0.002 \times 5 = 0.010 \text{ V} = 10 \text{ mV } \textbf{Ans.}$$

Example 10-16. *A six-bit ladder type D/A converter has a digital input : 101001. Find the analog voltage output for this D/A converter.*

Solution. Given: The number of D/A converter input bits = 6.

The digital input = 101001.

Let us assume the reference voltage, $V_{ref} = 10$ V.

We know that the analog output voltage,

$$V_{out} = V_{ref}\left(\frac{A_0}{2}+\frac{A_1}{4}+\frac{A_2}{8}+\frac{A_3}{16}+\frac{A_4}{32}+\frac{A_5}{64}\right)$$

$$= 10\left(\frac{1}{2}+\frac{0}{4}+\frac{1}{8}+\frac{0}{16}+\frac{0}{32}+\frac{1}{64}\right)$$

$$= 10\left(\frac{1}{2}+\frac{1}{8}+\frac{1}{64}\right) = 10\left(\frac{32+8+1}{64}\right) = \frac{10\times 41}{64}$$

$$= 6.40625 \text{ V } \textbf{Ans.}$$

Example 10-17. *A 10-bit D/A converter provides an analog output which has a maximum value of 10.23 volts. Find the resolution of this D/A converter.*

Solution. Given : The number of digital inputs, N = 10, Max. value of analog output voltage, $V_{out\,(max)} = 10.23$.

We know that resolution of a D/A converter,

$$\text{Resolution} = \frac{\text{Full-scale output}}{2^N - 1}$$

$$= \frac{10.23}{2^{10}-1} = 0.01\text{V or } 10 \text{ mV } \textbf{Ans.}$$

10-20. Applications of D/A Converter

D/A converters are used extensively in the field of electronics to convert the digital input to an analog voltage (or current). There are numerous applications, but some of the most common which are important from the subject point of view are given below.

1. **Control.** D/A converters are commonly used in control engineering applications. In these applications, the D/A converter receives a digital input from a microcontroller (or a computer) and converts to an analog control signal to adjust the process variable to a desired value. The process variable could be the temperature of a furnace, volume flow rate of a liquid gas through a pipe, speed of a motor etc.
2. **Automatic Test Equipment (ATE).** In this application, the microcontrollers (or computers) are programmed to generate some standard digital signals for testing any equipment. These signals are converted by D/A to their equivalent analog signals to test analog circuitry. The test circuit's analog output response will normally be converted to a digital value by an A/D converter and fed into a microcontroller system (or a computer) for storage, display and analysis.
3. **Signal Reconstruction.** In many applications such as audio and video recording, digital storage oscilloscope, audio CD systems, an analog signal (voice, picture or both) is digitized; that is, successive points on the signal are converted to their digital equivalents and stored in memory. This conversion is performed by an electronic device called analog-to-digital (A/D) converter. A D/A converter can then be used to convert the stored digital data back to analog—one point at a time—thereby reconstructing the original signal.
4. **A/D Conversion.** The D/A converter is used as a part of circuitry in many A/D converters. We shall study A/D converters in the next section.
5. **Serial DACs.** We know that most of the D/A applications involve the use of microcontroller (or a microprocessor). The main problem with using the parallel-data (8-bit/10-bit/16-bit etc.). D/A converters that have been mentioned so far is that they occupy many port bits of the microcontroller. In some cases, where speed of data transfer is of main concern, we have to use the required number of port bits of the microcontroller to connect it to D/A converter. However, in those cases where speed of data transfer is **NOT** a main concern, a microcontroller can output the digital value to a D/A converter over a serial interface. Serial D/A converters are now readily available with a built-in serial in/parallel out shift register. Many of these devices have more than one D/A converter on the same chip. The digital data, along with a code that specifies which D/A converter you want, are sent to a D/A device one at a time. As each bit is presented on the D/A input, a pulse is applied to the serial clock input to shift the bit in. After the proper number of clock pulses, the data value is latched and converted to its analog value.

10-21. Analog-to-Digital (A/D) Conversion

Basically, an A/D conversion is a process of taking an analog voltage and converting to its equivalent digital code. The A/D conversion process is generally more complex and time-consuming than that of D/A process. The electronic device that does the A/D conversion is called A/D converter.

Many different methods have been developed for A/D conversion. We shall study some of these methods in detail. It may be carefully noted that although knowing different methods of A/D conversion may never be necessary to design or construct A/D converters, but they are used to provide a greater insight into the factors that determine the performance of A/D converters.

Fig. 10-15 (*a*) shows a block diagram of an A/D converter that utilizes a D/A converter as a part of their circuitry. As seen from this diagram, the A/D converter consists of an Op-Amp, a control unit, a register, and a D/A converter. The timing for the operation is provided by the input clock signal. The control unit contains the logic circuit for generating the proper sequence of operations in response

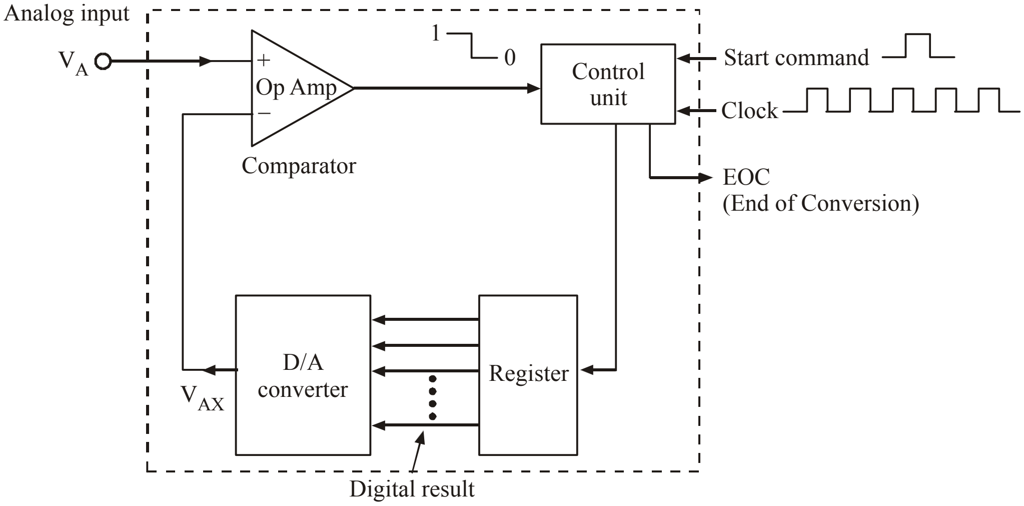


Fig. 10-15 (*a*)

to the *start command* which initiates the conversion process. The op-amp is used as a comparator. It has two analog inputs and a digital output. The digital output switches the states depending upon which analog input is greater.

The basic operation of the A/D converter utilizing the D/A converter may be described as follows :

1. The **"Start Command"** pulse initiates the operation.
2. The control unit continually modifies the binary number stored in the register. The rate at which the number is modified is determined by the clock input.
3. The binary number in the register is converted to an analog voltage, V_{AX}, by the D/A converter.
4. The comparator compares V_{AX} with the analog input voltage V_A. As long as $V_{AX} < V_A$, the comparator output stays "High". When V_{AX} exceeds V_A by at least an amount equal to V_T (the threshold voltage), the comparator output goes "low" and stops the process of modifying the number in the register. At this point, V_{AX} is a close approximation to V_A. The digital number in the register, which is the digital equivalent of V_{AX}, is also the approximate digital equivalent of V_A, within the resolution and accuracy of the D/A converter.
5. The control logic activates the **"End-of-conversion"** signal, EOC, when the conversion is complete.

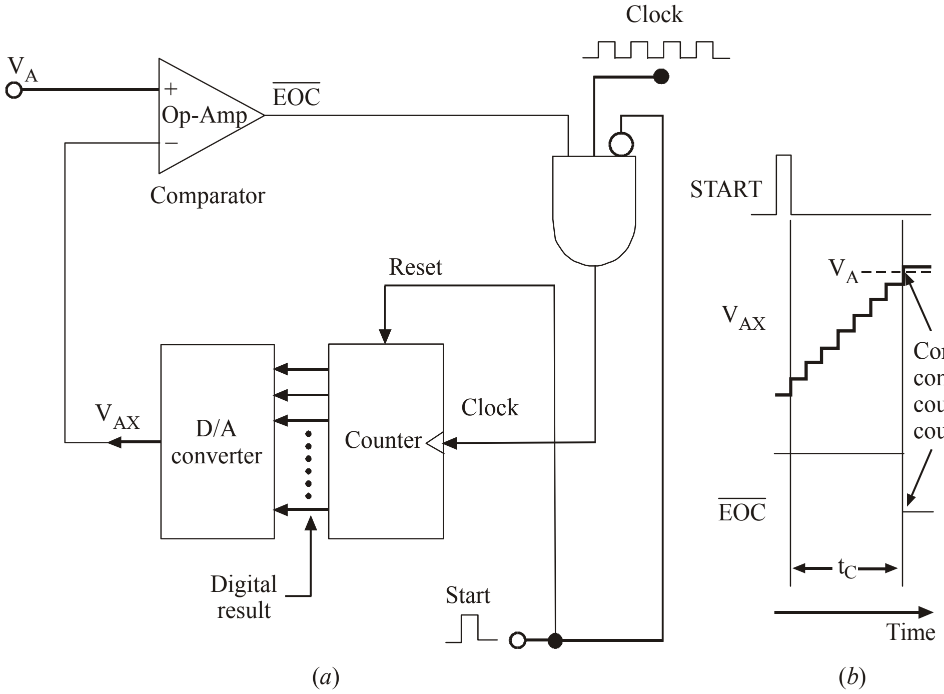
As a matter of fact, there could be several variations of this A/D conversion scheme. Every scheme may differ mainly in the manner in which the control section continually modifies the numbers in the register. However, the basic idea is the same with the register holding the required digital output.

10-22. Digital-Ramp A/D Converter

This is one of the most simplest versions of general A/D converter discussed in the last article (refer to Fig. 10-15). The digital-ramp A/D converter makes use of a binary counter as the register. It allows the clock to implement the counter one step at a time until $V_{AX} \geq V_A$. It is called a digital-ramp A/D converter because the waveform at V_{AX} is a step-by-step ramp (actually a staircase) like the one shown in Fig. 10-16(*b*). It is also known as a counter-type A/D converter.

Fig. 10-16 shows a block diagram for a digital ramp A/D converter. As seen from this diagram, it contains an op-amp comparator, a counter, a D/A converter and a 3-input logic AND gate. Notice that one of the AND gate inputs is an inverted input. The op-amp comparator output serves as the *active-low* end-of-conversion signal $(\overline{\text{EOC}})$. Notice that the $\overline{\text{EOC}}$ is connected to one of the normal inputs of AND gate, the clock pulses to the other normal input. The "start" pulse input is applied to the inverted input of AND gate. The "start" pulse input is also used to "reset" the counter. In order to explain the A/D conversion for the digital-ramp converter, let us assume that the analog voltage to be converted (*i.e.* V_A) is positive. Then the conversion operation is as follows :

1. A **"start"** pulse is applied to reset the counter to '0'.
2. With all the 0's at its input, the D/A converter's output will be zero (*i.e.* $V_{AX} = 0$).
3. Since the analog input voltage $V_A > V_{AX}$, the comparator output $(\overline{\textbf{EOC}})$ will be "high" (*i.e.* logic 1).
4. When the **"start"** pulse returns "low" (i.e. logic 0), the AND gate is enabled. As a result, the clock pulses get into the counter and the counter starts counting.
5. As the counter starts counting, the D/A converter output V_{AX} increases one step at a time as shown in Fig. 10-16(*b*). This continues until V_{AX} reaches a step that exceeds V_A by an amount equal to or greater than the threshold voltage of op-amp comparator (typically 10 to 100 μV). The moment $V_{AX} > V_A$, $\overline{\text{EOC}}$ will go "low". This inhibits the flow of clock pulses into the counter. As a result, the counter will stop counting.
6. The conversion process is now complete as signalled by the "high-to-low" transition at $\overline{\text{EOC}}$. At this moment, the counter output represents a binary number equivalent to the analog input voltage V_A.
7. The counter will hold this binary number at its output until the next "start" pulse initiates a new conversion.



(*a*) (*b*)

Fig. 10-16.

10-23. Successive Approximation A/D Converter

This is one of the most widely used types of A/D converter. Strictly speaking, it has more complex circuitry than the digital-ramp A/D converter but a much shorter conversion time. As a matter of fact, the conversion time of a successive approximation A/D converter has a fixed value. In other words, it is *not dependent* on the value of the input signal.

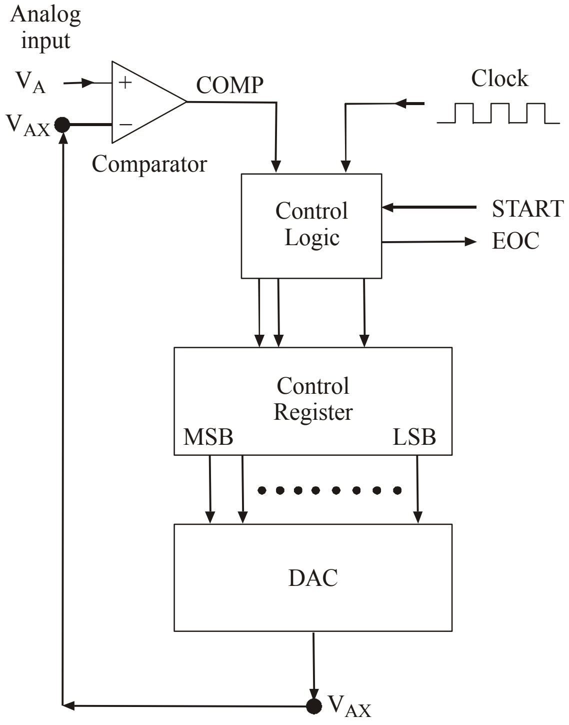


Fig. 10-17.

Fig. 10-17 shows a basic arrangement of a successive approximation type A/D converter. Notice that the arrangement is quite similar to the digital-ramp type A/D converter. However, it may be carefully noted that the successive approximation type A/D converter does not use a counter to produce an input to the D/A converter block, rather it uses a register.

The digit circuit inside the control logic block modifies the content of the register bit by bit until the register data are the digital equivalent of the analog input (V_A) within the resolution of the converter. The basic sequence of operations for the complete conversion process is illustrated through a flowchart shown in Fig. 10-18. We will use this flowchart to explain the A/D conversion through successive approximation method.

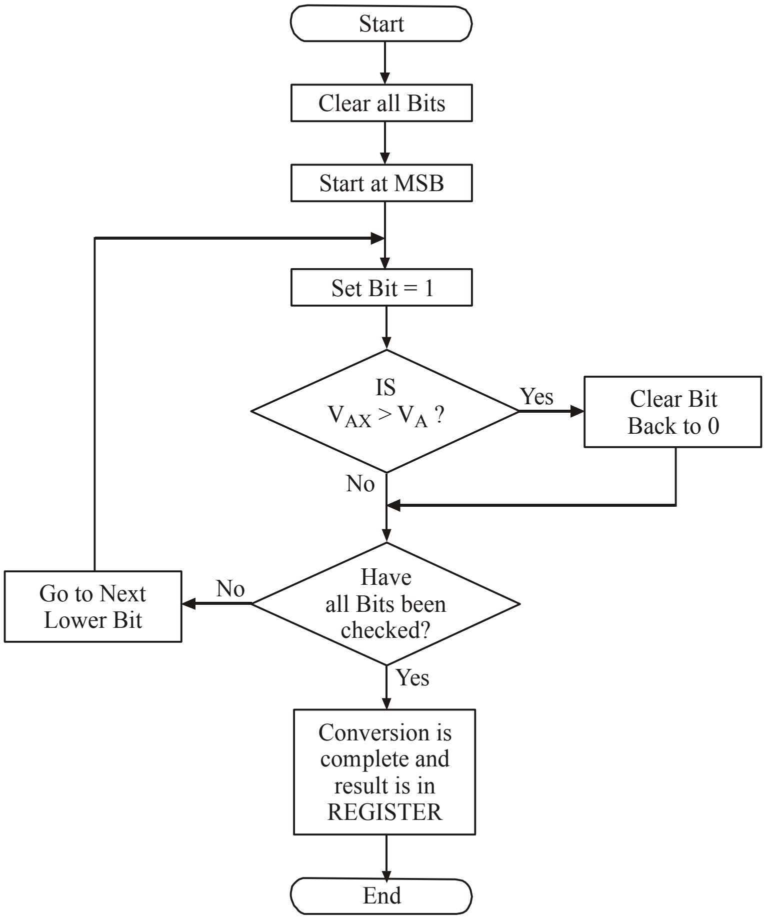


Fig. 10-18.

In order to understand the successive approximation method, let us consider a 4-bit D/A converter with a step-size of 1V connected to a comparator, control logic circuitry and a register as shown in Fig. 10-19. Let us assume that the analog input signal is 10.4 V. The operation begins with the control logic clearing all the register bits to 0, *i.e.*, $Q_3 = Q_2 = Q_1 = Q_0 = 0$. We can express this output state as $[Q] = 0000$. This state causes the D/A converter to produce an output, $V_{AX} = 0V$. This is indicated at time ‘ t_0 ’ on the timing diagram shown in Fig. 10-20.

At the next step, time t, the control logic sets the most-significant bit (MSB) of the register to 1, so that the output state is $[Q] = 1000$. This causes the D/A converter to produce $V_{AX} = 8V$. Since the analog input voltage (V_A) is 10.4V and the D/A converter output (V_{AX}) is 8V, (*i.e.*, $V_{AX} < V_A$), the comparator output (COMP) is still logic “high”. The logic “high” state tells the control logic that the setting of MSB did not make V_{AX} exceed V_A, so that the MSB is kept at 1.

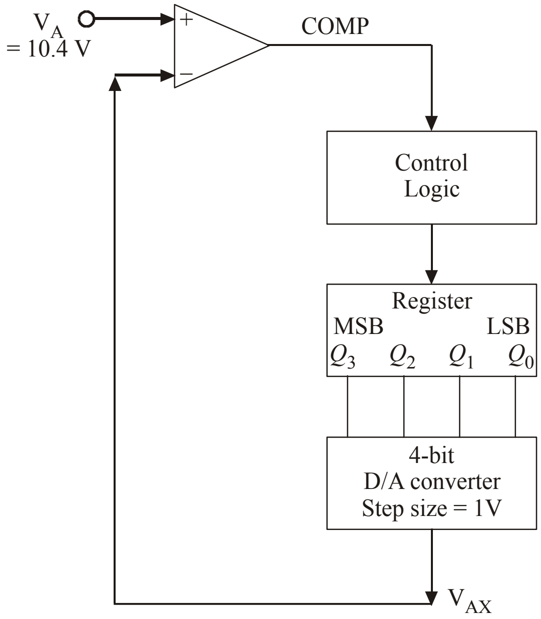


Fig. 10-19.

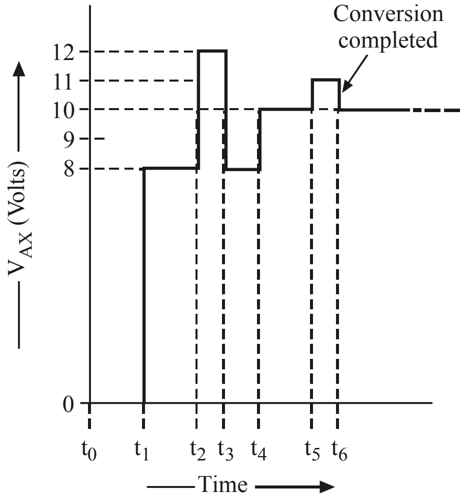


Fig. 10-20.

The control logic now proceeds to the next lower bit, Q_2. It sets Q_2 to '1'. As a result of this the output state [Q] = 1100. This causes the D/A converter to produce V_{AX} = 12V. This situation is indicated at time t_2 in the timing diagram [Fig. 10-20]. Since $V_{AX} > V_A$, the comparator output (COMP) goes "low". This logic 'low' state signals the control logic that the value of V_{AX} is too large. As a result, the control logic clears Q_2 back to 0 at time t_3. Thus at t_3, the register contents are back to 1000 and V_{AX} is back to 8V.

At the next step *i.e.* at t_4, the control logic sets the next lower bit Q_1. As a result of this the output state of the register [Q] = [1010]. This causes the D/A converter to produce V_{AX} = 10V. With $V_{AX} < V_A$, the comparator output (COMP) is logic 'high'. This situation tells the control logic to keep Q_1 at 1.

The final step occurs at t_5 where the control logic sets the next lower bit Q_0 of the shift register. As a result, the register output, [Q] = [1011]. This causes the D/A converter to produce V_{AX} = 11V. Since $V_{AX} > V_A$, the comparator output (COMP) goes "low". This situation tells the control logic to clear Q_0 back to 0 at time t_6.

Notice that at this point all the register bits have been processed. Therefore the A/D conversion process is complete. As a result, the control logic activates its "End-of-conversion" $(\overline{EOC})$ output to signal that the digital equivalent of the analog input signal (V_A) is now in the register.

It may be carefully noted that 1010 is actually equivalent to 10V. This value is less than the value of analog signal input (10.4V). This is a characteristic of successive approximation method. This is different from a digital-ramp method where the digital output was always equivalent to a voltage that was on the step above V_A.

Example 10-18. *An 8-bit successive approximation type A/D converter has a resolution of 20 mV. Calculate the value of digital output if an analog input signal is 2.51V.*

Solution. Given : Resolution of an 8-bit A/D converter = 20 mV = 20×10^{-3} V; V_A = 2.51 V.

We know that the number of steps required,

$$= \frac{2.51}{20 \times 10^{-3}} = 125.5$$

We also know that step 125 would produce $V_{AX} = 2.50V$ and step 126 will produce 2.52V. The successive approximation type A/D converter always produces a final V_{AX} that is at the step *below* V_A. Therefore for the case of $V_A = 2.51V$, the digital output would be 125_{10}. The 8-bit binary equivalent of this number is 01111101_2 **Ans.**

10-24. Conversion Time of Successive Approximation A/D Converter

We have already discussed in the last article about the operation of successive approximation type A/D converter. We mentioned there that the control logic goes to each register bit, sets it to 1, decides whether or not to keep it at 1 and goes on to the next bit. The processing of each bit takes one clock cycle, so that the conversion time for an N-bit A/D converter will be,

$$t_c = \text{Number of bits} \times \text{time-period of one clock cycle.}$$

$$= N \times T$$

Let us illustrate it with an example. Suppose we have an 8-bit A/D converter with a clock frequency of 640 kHz. Then we know that time period of one clock cycle,

$$T = \frac{1}{640 \text{ kHz}} = \frac{1}{640 \times 10^3} = 1.56 \text{ ms}$$

∴ The conversion time for 8-bit A/D converter,

$$t_c = N \times T$$

$$= 8 \times 1.56 \text{ ms} = 12.5 \text{ ms}$$

It may be carefully noted that for the successive approximation type A/D converter, the conversion time will be the same regardless of the value of analog input signal (V_A). This is because of the fact that the control logic has to process each bit to see whether a '1' is needed or not.

Example 10-19. *Compare the maximum conversion time of a 12-bit digital-ramp type A/D converter and a 12-bit successive approximation A/D converter if both utilize a 1 MHz clock frequency.*

Solution. Given: Resolution of A/D converters = 12-bit, clock frequency (f) = 1 MHz = 1×10^6 Hz.

We know that time period of the clock,

$$T = \frac{1}{f} = \frac{1}{1 \times 10^6} = 1 \times 10^{-6} s \text{ or } 1 \,\mu\text{s}.$$

We also know that for a digital-ramp converter, the maximum conversion time,

$$= (2^N - 1) \times T$$

$$= (2^{12} - 1) \times 1 \,\mu\text{s} = 4095 \,\mu\text{s}$$

$$= 4.095 \text{ ms}$$

and the conversion time for successive approximation type A/D converter

$$= N \times T = 12 \times 1 \,\mu\text{s} = 12 \,\mu\text{s}.$$

Note. A quick comparison of the conversion times of two A/D converter types indicates that the 12-bit successive approximation A/D converter is 4095/12 = 341 times faster than the digital-ramp type converter.

10-25. An Actual IC : The ADC 0808 Successive Approximation A/D Converter

There is a large variety of A/D converters available off-the-shelf from several IC manufacturers. These A/D converters have a wide range of operating characteristics and features. Let us study one of the most popular devices to get a better idea of the characteristics and features. The understanding of characteristics and features is very important from the selection and application of the device in digital system design.

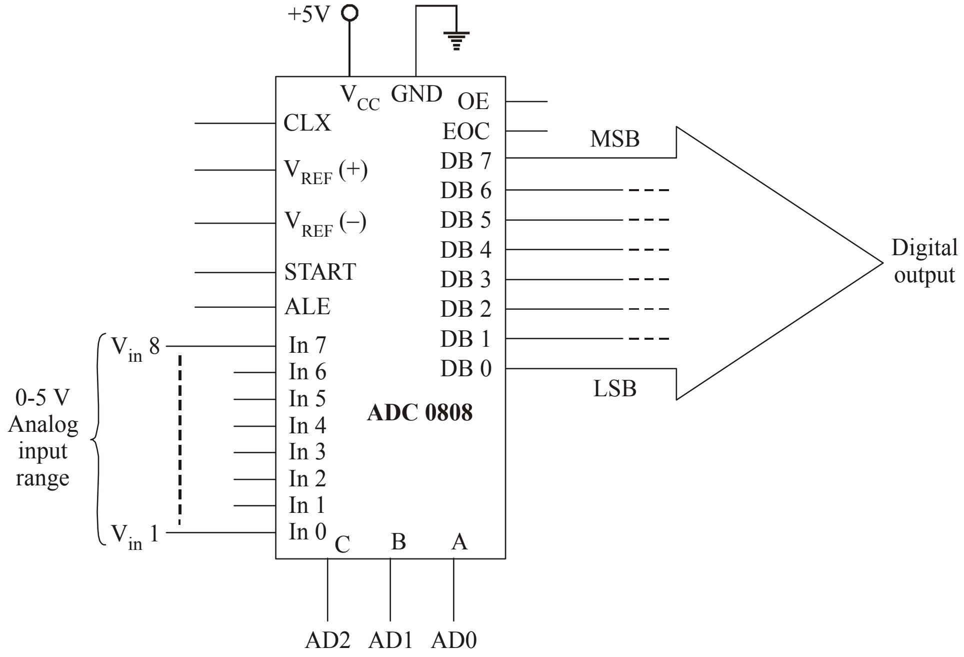


Fig. 10-21.

Fig. 10-21 shows the block symbol for ADC 0808 A/D converter. This device is a 28-pin, CMOS IC that performs A/D conversion using successive approximation technique. Although ADC 0808 has several characteristics but the following are important from subject point of view.

1. It has 8 analog inputs (In 0, In 1 ... In 7). All these inputs are converted to an internal 8-channel single-ended analog multiplexer. A particular input channel is selected through the internal address decoder. The address decoder has 3 inputs labelled as A, B and C. Table 10-3 shows the selection of the analog input based on the input state of the address lines A, B and C.

Table 10-3

Selected Analog Channel	Address Line		
	C	B	A
IN 0	0	0	0
IN 1	0	0	1
IN 2	0	1	0
IN 3	0	1	1
IN 4	1	0	0
IN 5	1	0	1
IN 6	1	1	0
IN 7	1	1	1

As seen from this table, if A = B = C = 0, the analog input connected to IN0 will be selected for its conversion to its 8-bits digital equivalent number.

2. Since the device converts the analog input voltage to its 8-bit digital output, therefore the resolution of A/D converter is 5V/255 = 19.6 mV. All the digital outputs are tri-state buffered.

3. The device can be operated at a clock frequency ranging from 10 kHz to 1280 kHz. A typical clock frequency is 640 kHz. The external clock signal is connected through "clock" pin.
4. Using an external clock frequency of 640 kHz, the conversion time is 100 μs.
5. The device has only one common ground pin for all analog and digital signals.

The device is designed to be easily interfaced to a microprocessor databus. For this reason the names of some of the ADC 0808 inputs and outputs are based on functions that are common to microprocessor based systems. The functions of these inputs and outputs may be explained with the help of Fig. 10-22.

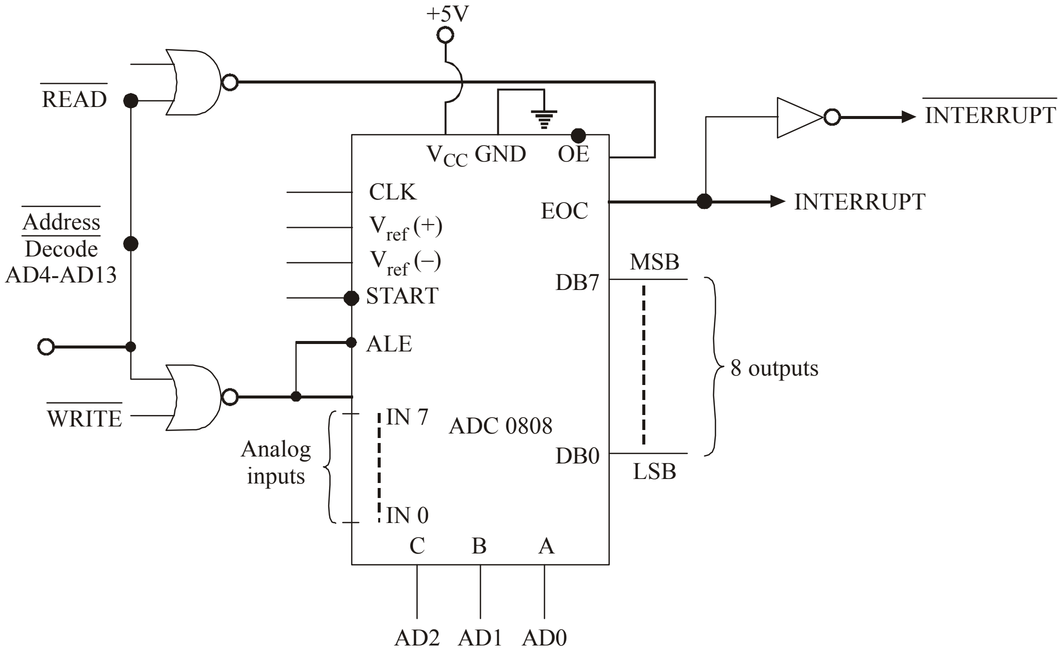


Fig. 10-22.

ALE (Address Latch Enable) : A positive-going pulse is obtained through a 2-input NOR gate with one of its inputs connected to any one of the Address Bus (AD4-AD13) and the other input firm $\overline{\text{WRITE}}$ signal.

START : A positive-going pulse is obtained from the same NOR gate as that for ALE (Address Latch Enable).

$\overline{\textbf{READ}}$ **:** This input, along with any one of the $\overline{\text{Address Decode}}$ (AD4-AD15) lines connected through a NOR gate, is used to enable the digital output buffers with $\overline{\text{Address Decode}} = \overline{\text{WRITE}} = \text{LOW}$, the digital output pins will have logic levels representing the results of the last A/D conversion. Then the microcomputer (or microprocessor) can read (or fetch) this digital data value over the system databus.

EOC (End of Conversion) : This output signal will go "low" at the start of conversion and will return "high" to signal the end of conversion. This signal is also called "INTERRUPT" because in a typical situation, it is sent to a microprocessor's interrupt input to get the microprocessor's attention and let it know that ADC's data are ready to be read.

CLK (Clock) : This input is used to feed the clock pulses to the device necessary for its operation.

V_{ref} (+) : This is the voltage to be applied to the top of ladder network. In most of the cases, this value is the same as that of V_{CC} (Refer to page 2 and 3 of the ADC 0808 datasheet).

V_{ref} (–) : This is the voltage to be applied to the bottom of the ladder network. In most of the cases, this value is zero (Refer to page 2 and 3 of the ADC 0808 datasheet).

Fig. 10-23 shows the device specifications datasheet of ADC 0808. You are strongly advised to study the block diagram, connection diagram, general description, features, key specifications and the electrical characteristics of ADC 0808.

ADC0808/ADC0809

8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 datasheet. (See AN-247 for more information.)

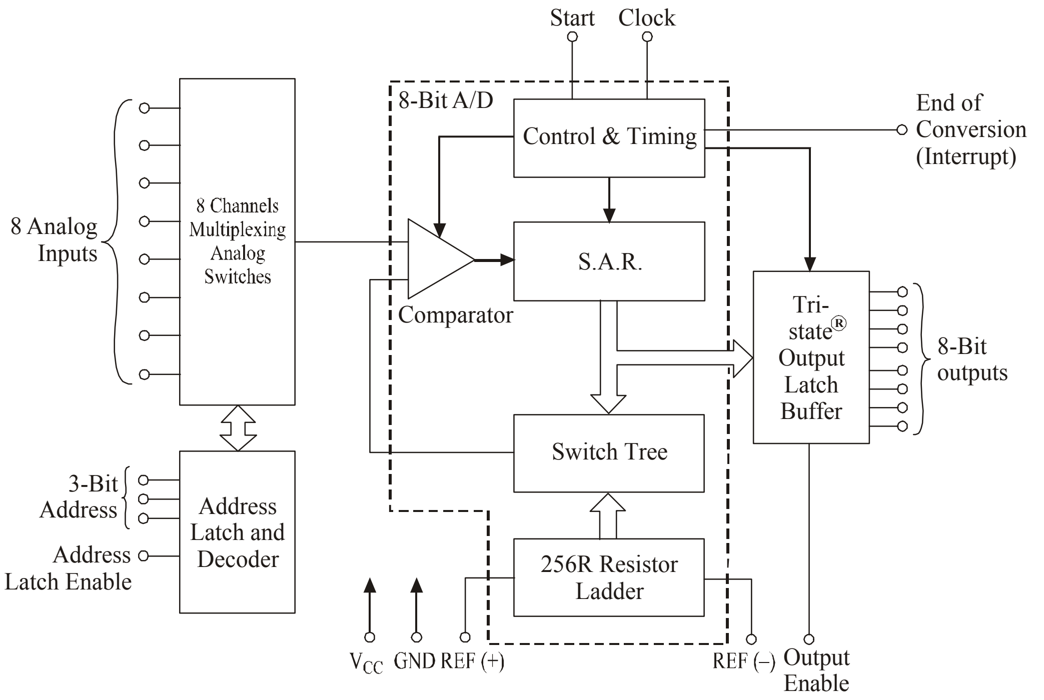
Features

- Easy interface to all microprocessors
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- No zero or full-scale adjust required
- 8-channel multiplexer with address logic
- 0V to V_{CC} input range
- Outputs meet TTL voltage level specifications
- ADC0808 equivalent to MM74C949
- ADC0809 equivalent to MM74C949-1

Key Specifications

• Resolution	8 Bits
• Total Unadjusted Error	±Va LSB and ±1 LSB
• Single Supply	5 V_{DC}
• Low Power	15 mW
• Conversion Time	100 µs

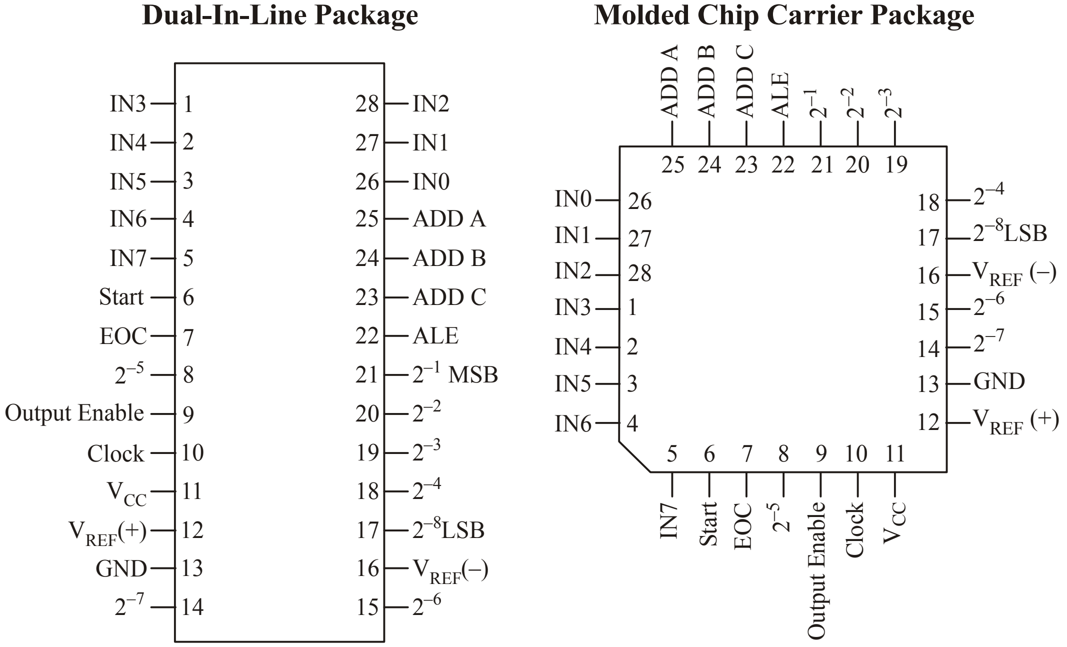
Block Diagram



See Ordering Information

Fig. 10-23 (Contd.)

Connection Diagrams



Order Number ADC0808CCN or ADC0809CCN
See NS Package J28A or N28A

Order Number ADC0808CCV or ADC0809CCV
See NS Package V28A

Fig. 10-23 (Contd).

Ordering Information

Temperature Range		**–40°C to +85°C**	
Package Outline		N28A Molded DIP	V28A Molded Chip Carrier
Error	± ½ LSB Unadjusted	ADC0808CCN	ADC0808CCV
	± 1 LSB Unadjusted	ADC0809CCN	ADC0809CCV

Absolute Maximum Ratings

(Notes 2, 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC}) (Note 3)	6.5 V
Voltage at Any Pin Except Control Inputs	–0.3V to (V_{CC} + 0.3V)
Voltage at Control Inputs (START, OE, CLOCK, ALE, ADD A, ADD B, ADD C)	–0.3V to +15V
Storage Temperature Range	–65°C to +150°C
Package Dissipation at T_A= 25°C	875 mW
Lead Temp. (Soldering, 10 seconds)	
Dual-In-Line Package (plastic)	260°C
Molded Chip Carrier Package	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C
ESD Susceptibility (Note 8)	400V

Operating Conditions (Note 1, 2)

Temperature Range (Note 1)	$T_{MIN} \leq T_A \leq T_{MAX}$ –40°C ≤ T_A ≤ + 85°C
Range of V_{CC} (Note 1)	4.5 V_{DC} to 6.0 V_{DC}

Electrical Characteristics - Converter Specifications

Converter Specifications : V_{CC}= 5 V_{DC}=V_{REF+1} $V_{REF(-)}$=GND, $T_{MIN} \leq T_A \leq T_{MAX}$ and f_{CLK}= 640 kHz unless otherwise stated.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
	ADC0808 Total Unadjusted Error (Note 5)	25°C T_{MIN} to T_{MAX}			± ½ ± ¾	LSB LSB
	ADC0809 Total Unadjusted Error (Note 5)	0°C to 70°C T_{MIN} to T_{MAX}			± 1 ± 1¼	LSB LSB
	Input Resistance	From Ref(+) to Ref(–)	1.0	2.5		kΩ
	Analog Input Voltage Range	(Note 4) V(+) or V(–)	GND–0.10		V_{CC}+0.10	V_{DC}
$V_{REF(+)}$	Voltage, Top of Ladder	Measured at Ref(+)		V_{CC}	V_{CC}+0.1	V
$\frac{V_{REF(+)} + V_{REF(-)}}{2}$	Voltage, Center of Ladder		V_{CC}/2–0.1	V_{CC}/2	V_{CC}/2+0.1	V
$V_{REF(-)}$	Voltage, Bottom of Ladder	Measured at Ref(–)	–0.1	0		V
I_{IN}	Comparator Input Current	f_c=640 kHz, (Note 6)	–2	± 0.5	2	μA

Electrical Characteristics - Digital Levels and DC Specifications

Digital Levels and DC Specifications: ADC0808CCN, ADC0808CCV, ADC0809CCN and ADC0809CCV, $4.75 \leq V_{CC} \leq 5.25V$, $-40°C \leq T_A \leq +85°C$ unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
ANALOG MULTIPLEXER						
$I_{OFF(+)}$	OFF Channel Leakage Current	V_{CC}= 5V, V_{IN}= 5V, T_A= 25°C T_{MIN} to T_{MAX}		10	200 1.0	nA µA
$I_{OFF}(-)$	OFF Channel Leakage Current	V_{CC}= 5V, V_{IN}= 0, T_A=25°C T_{MIN} to T_{MAX}	–200 –1.0	–10		nA µA
CONTROLINPUTS						
$V_{IN(1)}$	Logical "1" Input Voltage		V_{CC}–1.5			V
$V_{IN(0)}$	Logical "0" Input Voltage				1.5	V

Electrical Characteristics - Digital Levels and DC Specifications (continued)

Digital Levels and DC Specifications: ADC0808CCN, ADC0808CCV, ADC0809CCN and ADC0809CCV, $4.75 \leq V_{CC} \leq 5.25V$, $-40°C \leq T_A \leq +85°C$ unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
CONTROL INPUTS						
$I_{IN(1)}$	Logical "1" Input Current (The Control Inputs)	V_{IN} = 15V			1.0	µA
$I_{IN(O)}$	Logical "0" Input Current (The Control Inputs)	V_{IN}= 0	–1.0			µA
I_{CC}	Supply Current	f_{CLK}= 640 kHz		0.3	3.0	mA
DATA OUTPUTS AND EOC (INTERRUPT)						
$V_{OUT(1)}$	Logical "1" Output Voltage	V_{CC} = 4.75V I_{OUT} = –360µA I_{OUT} = –10µA	2.4 4.5			V(min) V(min)
$V_{OUT(0)}$	Logical "0" Output Voltage	I_0=1.6 mA			0.45	V
$V_{OUT(0)}$	Logical "0" Output Voltage EOC	I_0=1.2 mA			0.45	V
I_{OUT}	TRI-STATE Output Current	V_0 = 5V V_0= 0	 –3		3	µA µA

Electrical Characteristics - Timing Specifications

Timing Specifications $V_{CC}=V_{REF(+)}= 5V$, $V_{REF(-)}$ =GND, $t_r = t_f=20$ ns and $T_A=25°C$ unless otherwise noted.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{ws}	Minimum Start Pulse Width	(*Figure 5*)		100	200	ns
t_{WALE}	Minimum ALE Pulse Width	(*Figure 5*)		100	200	ns
t_s	Minimum Address Set-Up Time	(*Figure 5*)		25	50	ns
t_H	Minimum Address Hold Time	(*Figure 5*)		25	50	ns
t_D	Analog MUX Delay Time From ALE	R_S=OΩ (*Figure 5*)		1	2.5	µs
t_{H1}, t_{H0}	OE Control to Q Logic State	C_L=50 pF, R_L=10k (*Figure 8*)		125	250	ns
t_{lH}, t_{0H}	OE Control to Hi-Z	C_L=10 pF, R_L=10k (*Figure 8*)		125	250	ns
t_c	Conversion Time	f_c=640 kHz, (*Figure 5*) (Note 7)	90	100	116	µs
f_c	Clock Frequency		10	640	1280	kHz
t_{EOC}	EOC Delay Time	(*Figure 5*)	0		8+2µS	Clock Periods
C_{IN}	Input Capacitance	At Control Inputs		10	15	pF
C_{OUT}	TRI-STATE Output Capacitance	At TRI-STATE Outputs		10	15	PF

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 2: All voltages are measured with respect to GND, unless otherwise specified.

Note 3: A Zener diode exists, internally, from V_{CG} to GND and has a typical breakdown voltage of 7 V_{DC}.

Note 4: Two on-chip diodes are tied to each analog input which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the V_{CC}n supply. The spec allows 100 mV forward bias of either diode. This means that as long as the analog V_{IN} does not exceed the supply voltage by more than 100 mV, the output code will be correct. To achieve an absolute $0V_{DC}$ to $5V_{DC}$ input voltage range will therefore require a minimum supply voltage of 4.900 V_{DC} over temperature variations, initial tolerance and loading.

Note 5: Total unadjusted error includes offset, full-scale, linearity, and multiplexer errors. See *Figure 3*. None of these A/Ds requires a zero or full-scale adjust. However, if an all zero code is desired for an analog input other than 0.0V, or if a narrow full-scale span exists (for example: 0.5V to 4.5V full-scale) the reference voltages can be adjusted to achieve this. See *Figure 13*.

Note 6: Comparator input current is a bias current into or out of the chopper stabilized comparator. The bias current varies directly with clock frequency and has little temperature dependence (*Figure* 6). See paragraph 4.0.

Note 7: The outputs of the data register are updated one clock cycle before the rising edge of EOC.

Note 8: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

10-27. Quantization Error of an A/D Converter

As a matter of fact, it is very important to understand the errors associated with making any kind of measurements. An unavoidable source of error in the digital-ramp method if the step-size (or resolution) of the internal D/A converter is the smallest unit of measure. Imagine trying to measure the cricket players' heights by standing them next to a staircase with a 12″ steps and assigning them the height of the first step higher than their head. Anyone over 6′ would be measured as 7′ tall. Likewise the output voltage (V_{AX}) of the internal D/A converter is a staircase waveform that goes up in discreet steps until it exceeds the analog input voltage, V_A. By making the step-size smaller, we can reduce the potential error. But in practice, there will always be a difference between the actual (analog) quantity and the digital value assigned to it. This difference between the two values is called **quantization error.**

Let us assume the step-size of the internal D/A converter in a digital-ramp A/D converter is 10 mV. Since V_{AX} is an approximation to the value of V_A, therefore the best we can expect is that V_{AX} is within 10 mV of V_A. So the quantization error in this case is 10 mV. This quantization error can be reduced by increasing the number of bits in the counter and the D/A converter. The quantization error is sometimes specified as an error of +1 LSB indicating that could be off by as much as the weight of the LSB.

It may be carefully noted that the input V_A can take an infinite number of values ranging from 0 to full-scale. However, the approximation V_{AX} can take on only a finite number of discreet values. This means that a small range of V_A values will have the same digital representation.

10-28. Accuracy of an A/D Converter

As in D/A converter, the accuracy of an A/D converter is not related to the resolution but is dependent on the accuracy of the circuit components, such as the comparator, the D/A converter's precision resistors and current switches, the reference supplies and so on. For instance, an error specification of 0.01% full-scale, owing to nonideal components. This error is in addition to the quantization error due to the resolution. These two sources of error are usually of the same order of magnitude for a given A/D converter.

10-29. Sample and Hold Circuits

We have already discussed in this chapter that analog-to-digital converters are used to convert the electrical signals in analog form to their digital equivalent. In most of the applications in real world, the signals produced by sensors or actuators are continuously changing with time in response to the changing real world variables (e.g. pressure, volume, density, speed, distance etc.) For such situations, when the sensor or actuator output is directly connected as an input to an A/D converter, the conversion process will be affected due to continuous change in analog inputs. This problem is overcome by placing a sample and hold circuit before the A/D converter.

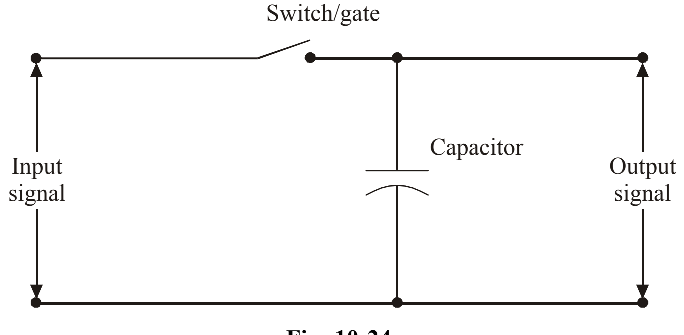


Fig. 10-24.

In most of the circuits, a capacitor is used to store the analog voltage and an electronic switch or logic gate is used to connect or disconnect the capacitor from the analog inputs. The rate at which the switch or logic gate is operated defines the number of samples per second taken from a continuous signal to a discrete signal. It is also called as sampling rate. Fig. 10-24 shows the arrangement for the basic sample and hold circuit.

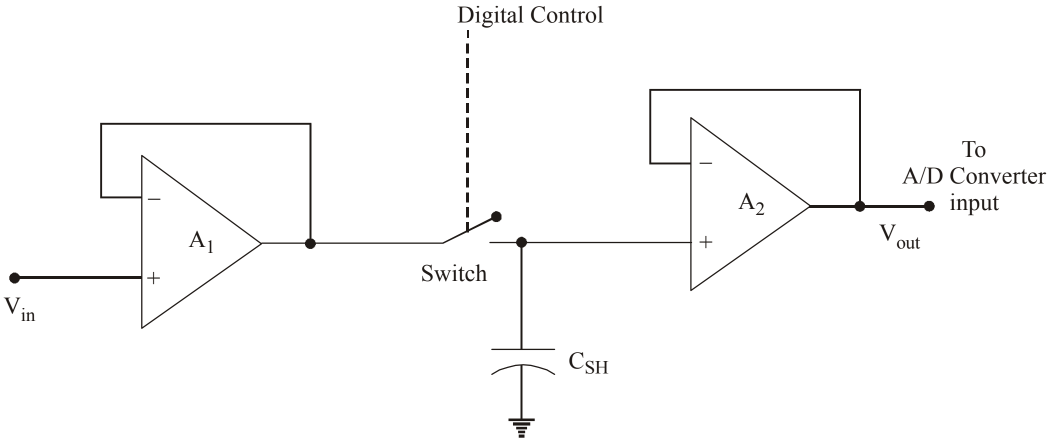


Fig. 10-25.

Fig. 10-25 shows a typical circuit diagram of the sample and hold operation. Here analog input is given to the unity gain buffer amplifier A_1 that gives the high impedance to the analog signal and a low output impedance to charge the capacitor C_{SH}. When the switch is closed, the capacitor C_{SH} is connected to the output of amplifier A_1 and will be charged. The switch will be controlled by a digital input. The capacitor gets charged as long as the switch is in closed position. This is called sample operation. When the switch is opened, the capacitor C_{SH} will hold this voltage so that the output of amplifier A_2 will apply the voltage to A/D converters. The unity gain amplifier A_2 gives high input impedance that will not discharge the capacitor during the conversion time of the A/D conversion operation. Thus the A/D converters receive the steady DC input voltage. This operation is called as 'hold' operation.

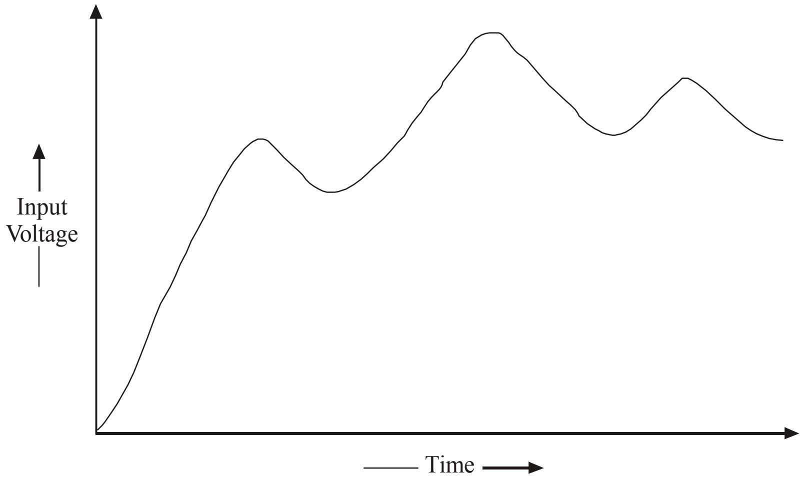


Fig. 10-26(*a*)

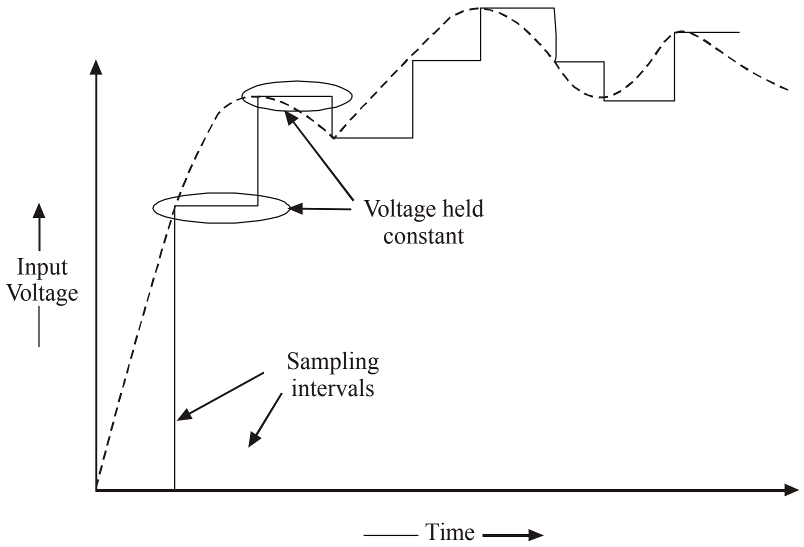


Fig. 10-26(*b*). Input signal and sample and hold output.

Fig. 10-26(*a*) shows the waveform of an input signal given to the sample and hold circuit while Fig. 12-26(*b*) the waveform of the circuit output. The vertical lines represent the instances at which the sampling is taking place. In between the two consecutive sampling intervals, the voltage is held constant.

10-30. Digital Storage Oscilloscope (DSO)

The digital storage oscilloscope (just like a normal oscilloscope) is a "test and measurement" equipment. It makes use of A/D and D/A converters internally to take advantage of processing of signals in digital form. Fig. 10-27 shows the block diagram of a digital storage oscilloscope (DSO).

The DSO operation is controlled and operated by a comparator block in the microprocessor. The data acquisition system contains a sample-and-hold circuit and an A/D converter that takes the samples and digitizes the input signal at a rate determined by the clock. This digitized data will be stored in the memory. If the memory becomes full, the first stored data will be overwritten by the latest data. This data acquisition and storage process will happen till the external trigger receives at the comparator circuit. Once the trigger occurs, the data acquisition will be stopped and will not acquire any new data. At this point of time the saved data will be shown in the LCD.

Advantages : The digital storage oscilloscope has many advantages than that of normal oscilloscope. Some of the advantages are as listed below :

1. Since the DSO uses digital memory, it can store the waveforms for longer time. But in the normal CRO this cannot happen.
2. In the DSO, we can store and view the part or full waveforms before the actual trigger happens. But this is not possible in the conventional CRO.
3. In the DSO, the stored waveform can be positioned anywhere in the screen. We can actually adjust the vertical and horizontal scales of the waveform. This is not possible in the normal CRO.
4. Nowadays most of the DSOs can store as many waveforms in the memory and if needed we can take the printouts of the waveforms by connecting a normal standard printer into the DSO.

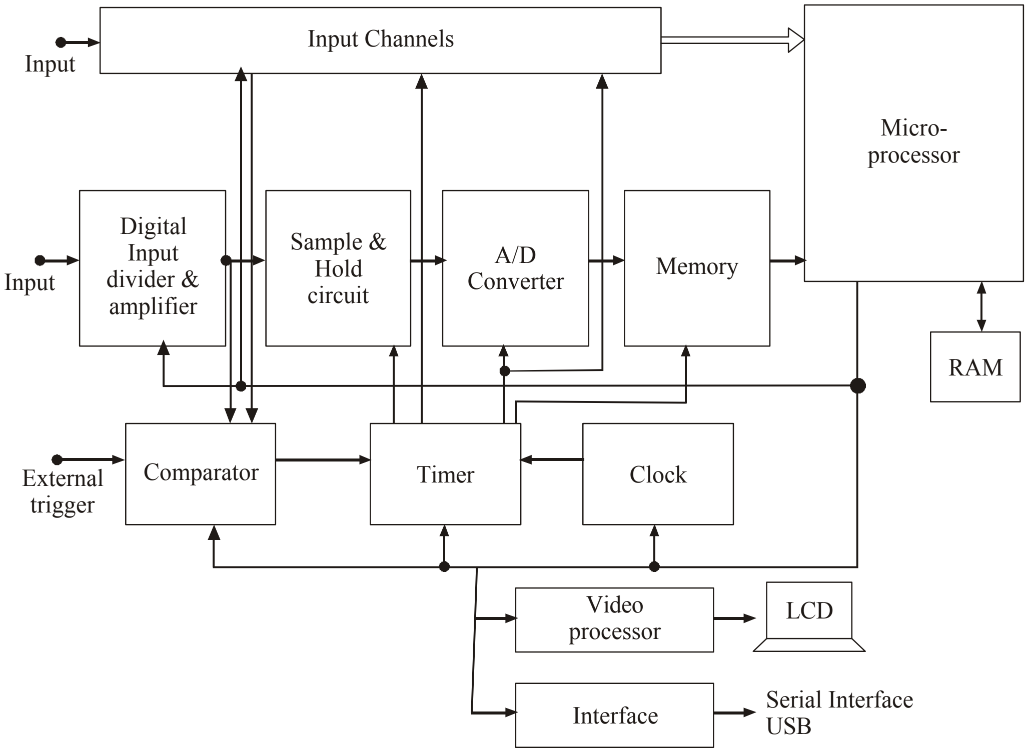


Fig. 10-27.

10-31. Digital Signal Processor (DSP)

A/D and D/A converters are mostly used in the digital signal processor. They are the integral part of DSPs. The DSP is a very advanced and most sophisticated microprocessor that uses a digitized data for special applications and some of the most complicated operations. DSP works on the data

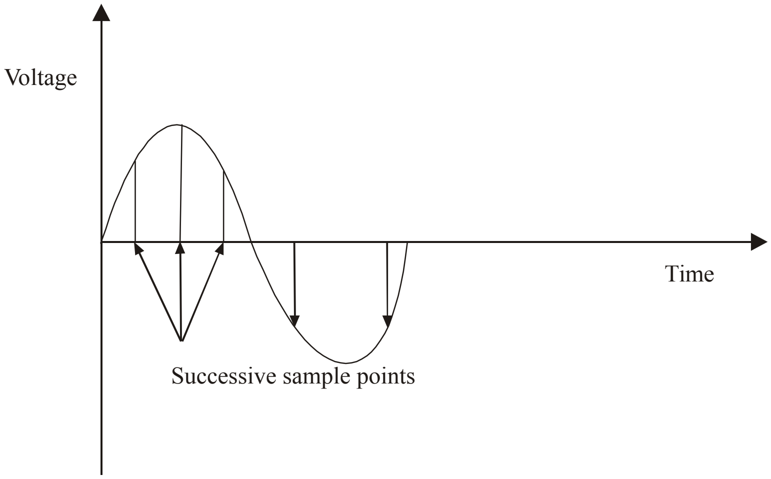


Fig. 10-28.

which is in digital form. This is done by sampling the voltage level at regular time intervals and converting the voltage level at that instant into a digital number proportional to the voltage. This process is performed by A/D converters. So this A/D converter gives the necessary digitized data to the DSP. In order to get the steady input voltage to the A/D converter, a sample-and-hold circuit is used before the A/D conversion. Once the conversion is complete, the sample-and-hold circuit is ready to update the voltage again, *i.e.*, ready for another conversion. In this way a succession of samples is made. It is represented in Fig. 10-28.

The digitized data from the A/D converter will be used in DSP for special purpose applications. The DSP performs complicated mathematical routines upon the representation of the signal. Finally to use the signal for the real uses, it then needs to be converted back to an analog form where it can be amplified. This will be passed into a speaker or headphones or for some other purposes. This conversion is performed by D/A converter.

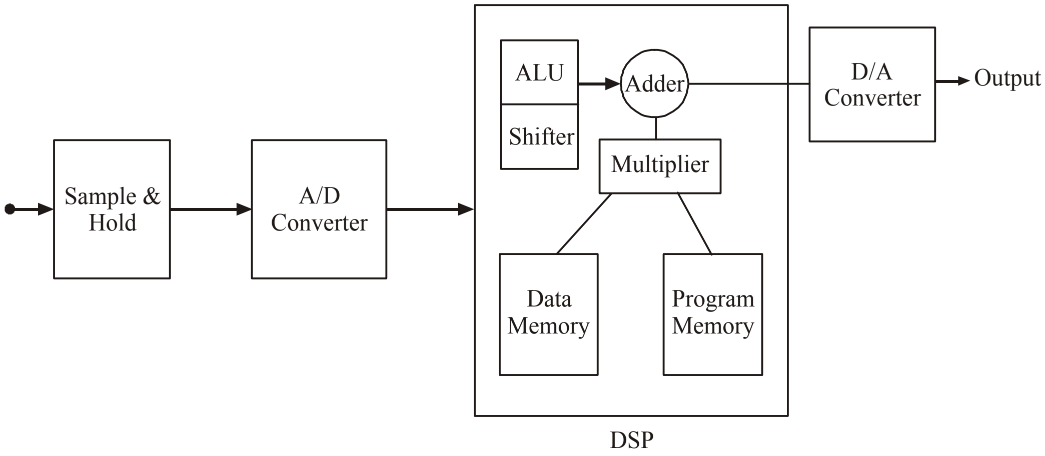


Fig. 10-29.

Fig. 10-29 shows a basic block diagram of a DSP. As shown, it has an arithmetic logic unit (ALU), data memory, program memory, circuits to shift, add and multiply data.

Although DSP is used in many applications, yet following are the ones which are important from subject point of view.

1. Digital audio applications — MPEG Audio & portable audio.
2. Digital cameras
3. Cellular telephones
4. Medical appliances
5. Storage products — disk drive servo control
6. Military applications — radar, sonar
7. Industrial control
8. Seismic exploration
9. Networking — Wireless
   — Modems
   — ADSL
   — VDSL

10-32. Multiplexing

To select one input from many analog input sources to be converted, multiplexing technique is used. The block diagram for this arrangement is shown in Fig. 10-30. Here four analog inputs are connected to a switch. This switch decides which input should be given for the conversion process. The switch selection is based on the circuit connected to the switch and its address bit selection.

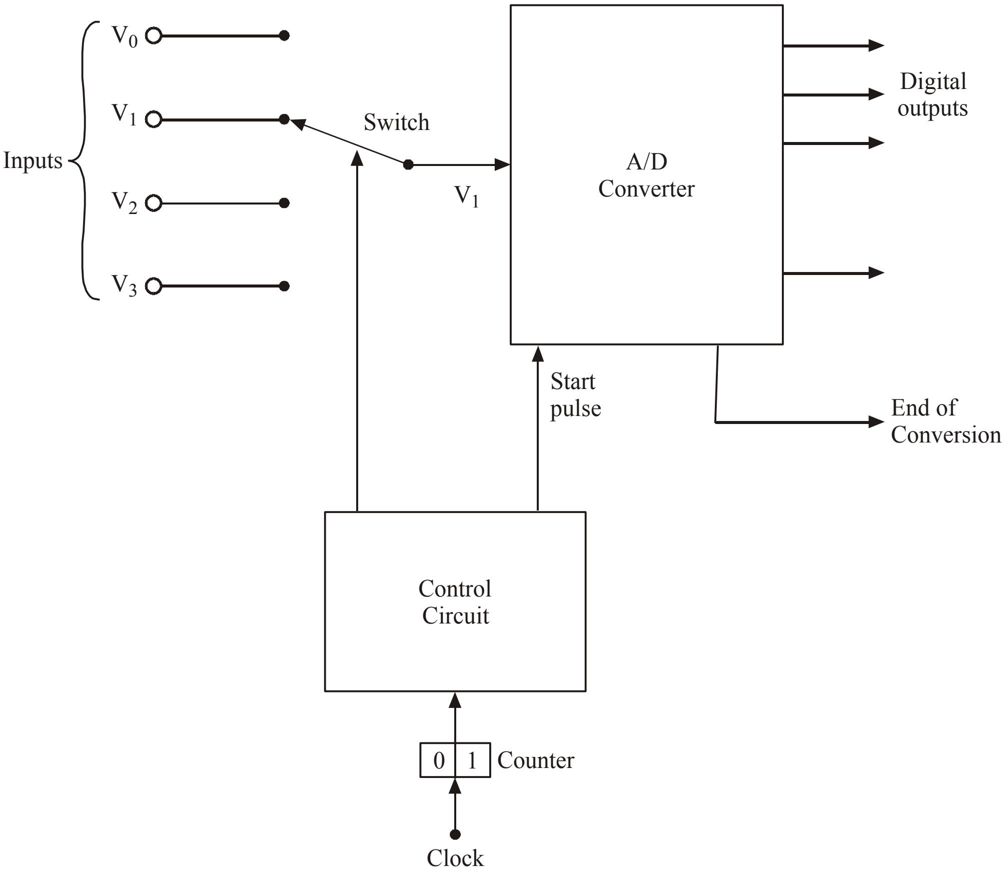


Fig. 10-30. Block diagram of Multiplexer & ADC

The control circuit is connected with a two bit counter. It decides the control to apply for the switch. For example, if the counter address is 00, then first analog input is selected for conversion. If the address is 01, then second input is selected. When a switch is selected, based on that an analog input is selected for analog-to-digital conversion. For this conversion a pulse will be given to the ADC. When the conversion is complete, signals will be sent out to denote that the conversion data is ready for use.

Now a days many ICs are having multiplexing circuitry on the same chip itself. For example, ADC 0808 chip can multiplex 8 different analog inputs. It houses a three bit selector for this input selection. Refer to the block diagram of ADC 0808 shown in Fig. 10-23 (a part of device specification sheet).

10-33. Data Acquisition System

In most of the real world applications, the analog data must be digitized and transferred into a computer's or microcontroller system's memory. The process by which the microcontroller acquires these digitized analog data is referred to as **data acquisition.** The microcontroller (or computer) can do several things with the data, depending on the application. In a storage application, such as digital audio recording, video recording, or a digital oscilloscope, the microcontroller will store the data and then transfer them to a D/A converter at a later time to reproduce the original signal. In a process control application, the microcontroller can examine the data or perform computations on them to determine what control outputs to generate.

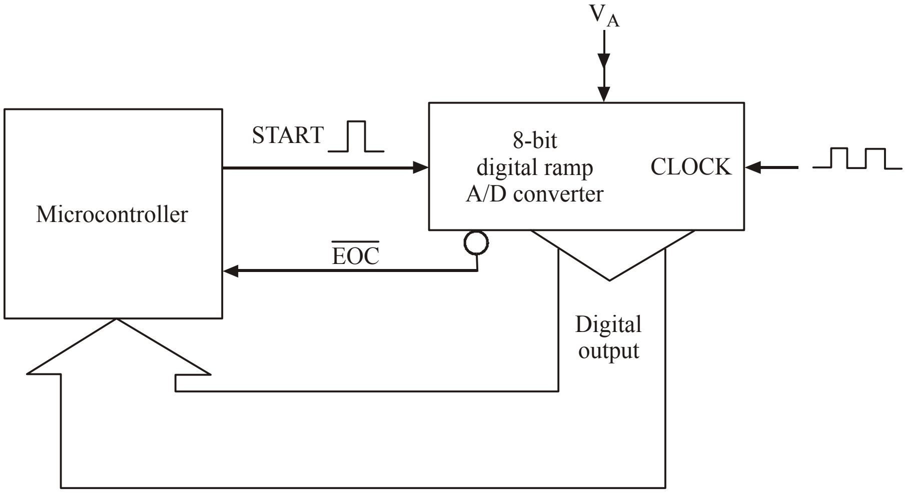


Fig. 10-31.

Fig. 10-31 shows the way a microcontroller is connected to a digital ramp A/D converter for data acquisition. As seen from this diagram, the microcontroller generates the "start" pulses that initiate each A/D conversion cycle. The "End-of conversion" ($\overline{EOC}$) signal from the A/D converter

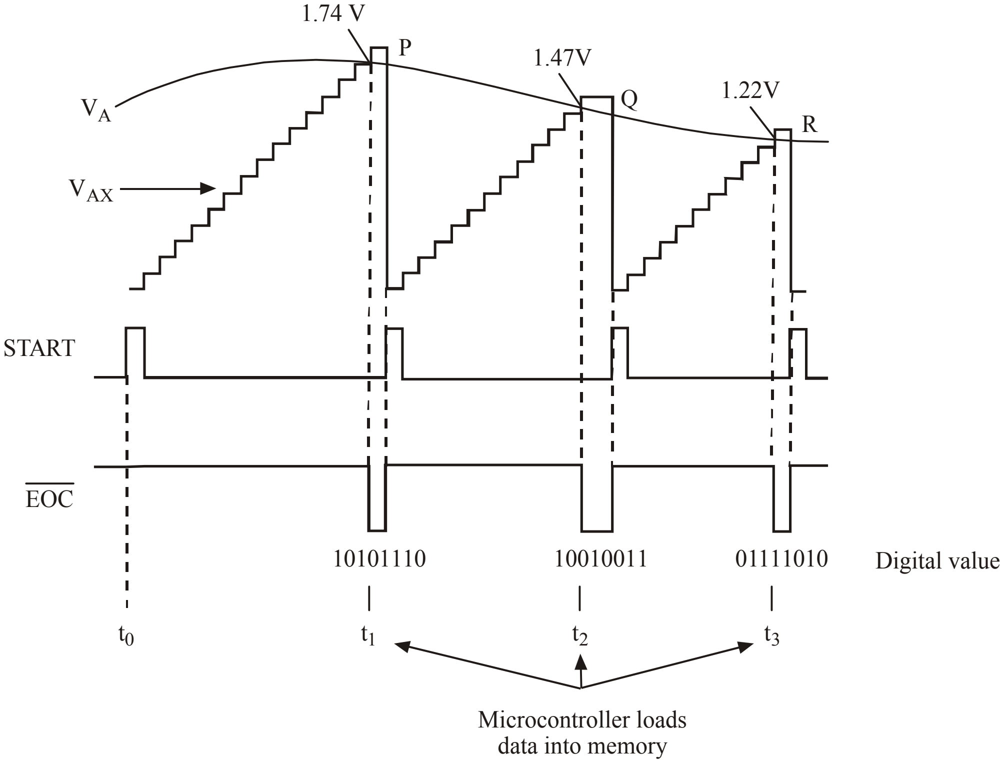


Fig. 10-32.

is fed to the microcontroller. The microcontroller monitors the $\overline{\text{EOC}}$ signal, to find out when the current A/D conversion is complete. When the conversion is complete, the microcontroller transfers the digital data from the A/D converter's output into its memory.

Fig. 10-32 shows the waveforms illustrating how the microcontroller acquires a digital version of the analog signal, V_A. The V_{AX} staircase waveform that is generated internal to the A/D converter is shown superimposed on the V_A waveform for illustration. As seen from the diagram, the process begins at t_0, when the microcontroller generates a **"start"** pulse to initiate an A/D conversion cycle. The conversion is completed at t_1, when the staircase waveform exceeds V_A. At this instant, the **end-of-conversion** $(\overline{\text{EOC}})$ goes **low.** The falling edge of $\overline{\text{EOC}}$ signals the microcontroller that the A/D converter has a digital output that now represents the value of V_A at point 'a', and the microcontroller will load these data into its memory.

The microcontroller generates a new "start pulse" shortly after t_1 to initiate a second conversion cycle. It may be carefully noted that the initiation of new "start" pulse resets the staircase to '0' and $\overline{\text{EOC}}$ back to logic "high" state. This happens because the new "start" pulse resets the counter in the A/D converter. The second conversion ends at "t_2" when the staircase waveform exceeds V_A. The microcontroller then loads the digital data corresponding to point 'b' into the memory. The entire process is repeated at "t_3", "t_4" and so on.

It will be interesting to know that the whole process whereby the microcontroller generates a "start" pulse monitors $\overline{\text{EOC}}$ and leads A/D converter data into memory is done under the control of a program that the microcontroller is executing. This data acquisition program will determine how many data points from the analog signal will be stored in the system memory.

10-34. Reconstructing an Analog Signal from a Digital Signal

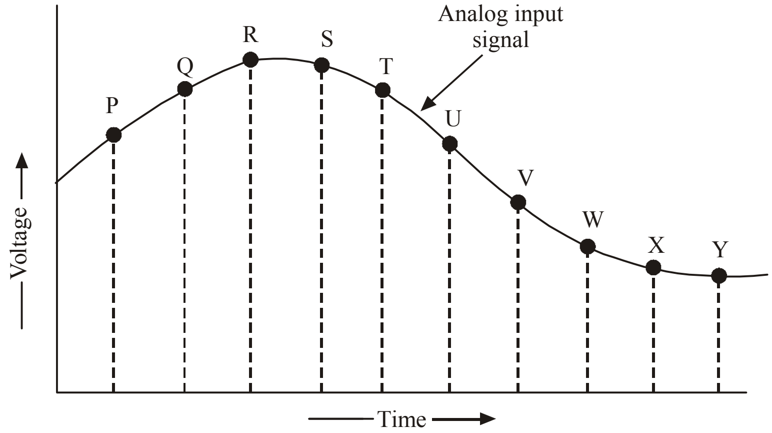
We have already discussed in the last article about the digitization of an analog signal. Notice from Fig. 10-25 that the A/D converter is operating at its maximum speed since a new "start" pulse is generated immediately after the microcontroller acquires the A/D converter output a data from the previous conversion.

It may be carefully noted from Fig. 10-25 that the conversion times are not constant. It is due to the fact that analog input value is changing. The microcontroller will store the digital data obtained from the various conversions in its memory. For instance, the digital data for the points P, Q and R (shown in Fig. 10-25) might look as shown in Table 10-4. These digital data are often used to later reconstruct an approximation to the original analog signal. To give you an example of signal reconstruction, recall a digital storage oscilloscope. In these oscilloscopes, the stored digital values are fed to a D/A converter to produce analog voltages that move the electron beam vertically as it is being swept horizontally by a time base signal. The result is as if an analog signal were reconstructed by drawing straight lines (vectors) from each digitized point to the next. Refer to Fig. 10-26 for illustration.

Table 10-4

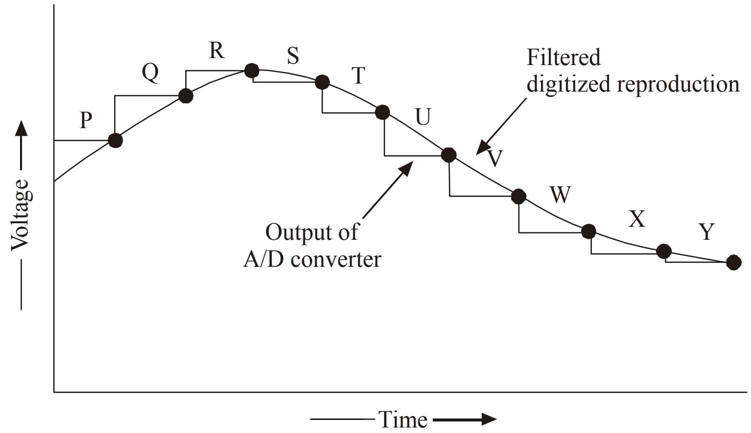
Point	Actual Voltage (V)	Digital Equivalent
P	1.74	10101110
Q	1.47	10010011
R	1.22	01111010

Fig. 10-33 shows an analog signal being digitized by an A/D converter. As seen from Fig. 10-33(*a*), the A/D converter continually performs conversions to digitize the analog signal at points P, Q, R, R, T, U, V, W, X and Y. If these data are used to reconstruct the signal, the result will look like as shown in Fig. 10-33(*b*).



(*a*) Digitizing an analog signal

Fig. 10-33



(*b*) Reconstructing the signal from the digital data.

Fig. 10-33.

A detailed examination of the original analog input signal shown in Fig. 10-33(*a*) and the reconstructed signal shown in Fig. 10-33(*b*) indicates that the two signals are fairly close to each other. This similarity between the two signals is possible as the analog signal does not make any rapid changes between the digitized points. It may be carefully noted that if the analog signal contained high frequency variations, the A/D converter would not be able to follow the variations. As a result of this, the reconstructed signal would also be less accurate. Hence it is important for the circuit designer to select an A/D converter whose conversion time is short enough to accommodate any fast variation of the analog input signal.

There are a variety of ADCs available commercially having different conversion speeds. The digital-ramp converters have a limitation of relatively longer conversion times.

SUMMARY

In this chapter you have learned :

1. D/A conversion is the process of taking a value represented in digital code and converting it to a voltage or current that is proportional to the digital value.
2. ADCs are used to convert the analog wave into numbers.
3. DACs are used to convert the digital numbers to the analog wave.
4. Resolution of a D/A converter is defined as the smallest change that can occur in the analog output as a result of a change in the digital input.
5. Manufacturers of D/A converters specify the resolution in number of bits.
6. The resolution of a D/A converter determines how many possible voltage values the microcontroller can send to the display/control valve.
7. There are many methods for producing the D/A operations.
8. The D/A converter is specified by its resolution, accuracy, maximum sampling frequency and monotonicity.
9. D/A converters are used in many applications such as control systems, automatic test equipments, signal reconstruction, A/D conversion and serial ADCs.
10. Digital-ramp A/D converter makes use of a binary counter as the register.
11. Successive approximation A/D converter has more complex circuitry than the digital-ramp A/D converter but a much shorter conversion time.

GLOSSARY

A/D Converter : It is used to convert analog or continuous voltage signal into a digital form.

D/A Converter : It is used to convert digital data from the system to an analog form.

Resolution of a D/A Converter : It is defined as the smallest change that can occur in the analog output as a result of a change in the digital input.

Pulse width modulator : A stable current or voltage is switched into a low pass analog filter with a duration determined by the digital input code.

Oversampling : It uses a lower resolution D/A converter.

Binary weighted method : A circuit contains one resistor or current sources for each bit of the D/A converter connected to a summing point.

Full-scale error : It is the maximum deviation of the D/A converter's output from its expected value.

Linearity error : It is the maximum deviation in step-size from the ideal step-size.

Maximum sampling frequency : This is a measurement of the maximum speed at which the D/A converter circuitry can operate and still produce the correct output.

Digital-ramp A/D converter : It is the most simplest version of A/D conversion that uses a binary counter as the register.

Successive approximation A/D converter : It is most widely used and has more complex circuitry than digital-ramp A/D converter.

DESCRIPTIVE QUESTIONS

1. Explain the D/A conversion process.
2. How can we describe the resolution of a D/A converter?
3. Explain different types of D/A conversions.
4. How can we specify the D/A converter?
5. What are the applications of D/A converters?
6. Explain about A/D conversion.
7. Explain the types of A/D convertors.

8. Discuss about the applications of A/D convertors.

9. Describe the construction of an ADC familiar to you.

(*Gauhati University,* 2007)

10. With the help of a neat diagram, explain parallel A/D converter.

(*PTU, May* 2007)

11. Comment on the parameters which serve to describe the quality of performance of a D/A convertor.

(*PTU, May* 2008)

12. Differentiate the resolution of the output from DAC that has a 12-bit input.

(*PTU, May* 2008)

13. Which is the fastest ADC add why?

(*PTU, Dec* 2008)

14. Describe the working of a successive approximation A/D converter with the help of a suitable diagram and compare its performance in terms of speed, accuracy and resolution with other ACDs.

(*PTU, Dec* 2008)

15. Explain the working of R-2R ladder type D/A converter.

(*PTU, May* 2009)

16. On what factors does the percentage resolution of D/A converter depend?

(*PTU, Dec.* 2009)

17. Define linearity, settling time, sensitivity and accuracy of A/D and D/A converters.

(*PTU, Dec.* 2009)

18. What is voltage to frequency converter? How is it used for designing A/D converter?

(*PTU, Dec.* 2009)

19. Describe the successive approximation A/D conversion principle with the neat diagram, explain this type of A/D converter.

(*GBTU/MTU,* 2007-08)

20. Explain “sampling theorem”.

(*Jamia Millia Islamia University,* 2007)

21. List various specifications of ADC and explain the dual slop A/D convert technique with help of block diagram.

(*Gujarat Technological University, Dec.* 2009)

22. Draw a neat diagram for an R-2R ladder type DAC and explain its operation.

(*WBUT., 2011-12*)

23. Describe the operation of successive approximation type ADC. How many clock pulse are required in worst case for each conversion cycle of an 8-bit SAR type ADC ?

(*WBUT., 2011-12*)

24. Draw a neat diagram for a weighted resistor type DAC and explain its operation.

(*WBUT., 2011*)

TUTORIAL PROBLEMS

1. A 12-bit D/A converter which uses a BCD input code has a full-scale output of 9.99V. Determine the step-size, percentage resolution and the value of V_{out} for an input code of 011010010101.
2. An 8-bit D/A converter has a full-scale error of 0.2% F.S. If the DAC has a full-scale output of 10 mA, what is the most that it can be in error for any digital input? If the D/A output reads 50 μA for a digital input 00000001, is this within the specified range of accuracy?
3. In an 8-bit weighted resistor D/A converter, the resistor value corresponds to the MSB is 64 kΩ. Determine the resistor value that corresponds to LSB?
4. An 8-bit ladder type D/A converter has a digital input 10100011. Find the analog voltage output for this D/A converter?
5. A 12-bit successive approximation type A/D converter has a resolution of 40 mV. Calculate the value of digital output if an analog input signal is 3.64V.

MULTIPLE CHOICE QUESTIONS

1. D/A conversion will use an input as

(*a*) Hexadecimal number (*b*) An octal number
(*c*) A binary number (*d*) An analog wave

2. The resolution of D/A conversion is equal to weight of

(*a*) Most-significant bit (MSB) (*b*) Least-significant bit (LSB)
(*c*) a & b (*d*) none of the above.

3. If an 8-bit D/A converter is used, how many possible steps of 0.39 mV will be there ?

(*a*) 1023 (*b*) 255
(*c*) 510 (*d*) 16

4. Which D/A converter will be having better resolution ?

(*a*) an 8-bit D/A (*b*) a 10-bit D/A
(*c*) a 12-bit D/A (*d*) a 4-bit D/A

5. A DAC 0808 has a maximum full-scale error of ± 0.19%. If the converter has full-scale current output as 2mA, then what will be the percentage of full-scale error?

(*a*) 3.8 μA (*b*) 1.9 μA
(*c*) 7.6 μA (*d*) 5.7 μA

6. A Digital-ramp A/D converter uses a

(*a*) binary number (*b*) decimal number
(*c*) BCD counter (*d*) All of the above

ANSWERS

1. (*c*) **2.** (*b*) **3.** (*b*) **4.** (*c*) **5.** (*a*) **6.** (*c*)

11

SEMICONDUCTOR MEMORIES

OUTLINE

Objectives

❍ know about what is a semiconductor memory.
❍ understand the read/write operation of memory devices.
❍ list the different types of semiconductor memory device.
❍ understand the organization of Read-only-Memory (ROM).
❍ describe the organisation of a typical memory cell in ROM, MROM, PROM, EPROM, EEPROM and flash read only memory.
❍ explain the organisation of a typical memory cell in static bipolar RAM, static MOS RAM, dynamic MOS RAM.
❍ describe what is cache memory and content addressable memory.

11-1. Introduction

Now a days semiconductor memories are used in almost all the electronic and digital appliances. A major advantage of digital systems over analog systems is the ability to store quantities of digital data and information. In the digital systems, this data can be stored for short or longer periods. The information stored in the memory mainly consists of instructions to be executed for some operations in binary form, intermediate and final results of the operations.

Earlier magnetic tapes were used to store the data. The magnetic tapes are capable of storing large amount of data compared to the semiconductor memories. But to read or write data usually takes longer time compared to semiconductor memories. The semiconductor memory devices are very much popular due to their small size, low cost, high reliability of stored data, high speed and easily expandable memory sizes.

11-2. Memory System of a Typical Computer System

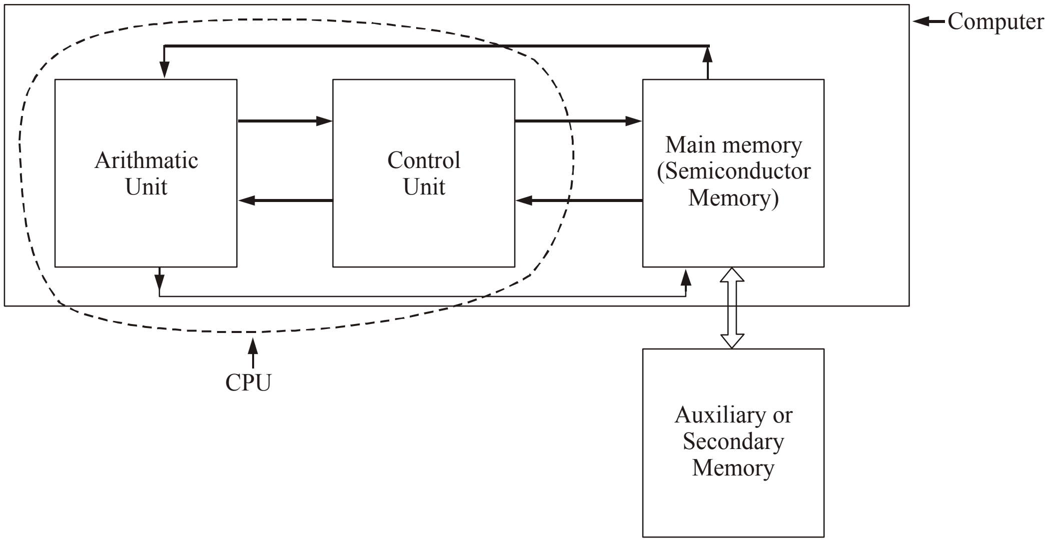

Computer
Arithmatic Unit
Control Unit
Main memory (Semiconductor Memory)
CPU
Auxiliary or Secondary Memory

Fig. 11-1. A digital computer system and its memory systems.

Fig. 11-1 shows the memory system of a typical computer system. As shown in the figure, there are two storage systems included namely (1) Main memory unit and (2) Auxiliary memory unit.

1. Main Memory Unit : In a computer system, semiconductor memory is used as a main memory, where fast operation is required. It is also called as primary memory unit. Normally the semiconductor memory is fixed on the circuit board itself. A computer's main memory which has programs to be executed is directly connected to the CPU (Central Processing Unit). A program and any data used by the program reside in the main memory while the computer is working on program and data.

2. Auxiliary Memory Unit : It is another form of storing the information in the digital computer system. It is also called secondary memory unit or mass storage system. As the name implies, the mass storage system has the capacity to store large amount of data without any electrical power. But its speed is slower than the main memory or semiconductor memory. Normally it stores the data and program which is not used by CPU at that point of time. The information from auxiliary memory will be transferred to the main semiconductor memory whenever the data or program is required by the CPU. The common examples of auxiliary memory devices are magnetic disk, magnetic tape, and compact disk.

In this chapter we will mainly discuss about the semiconductor memory and its characteristics.

11-3. Memory Organization

The basic element used in memory is called "flip-flop". In a flip-flop the data is stored in binary form, either as 0 or as 1. Each such binary digit of data is called a 'bit'. A collection of 4 bits is called a 'nibble', while that of 8 bits is called a 'byte'. Fig. 11-2 shows the combination of bits and bytes for a 32 bit memory system. As seen from this figure, a group of four bytes constitute a "data word" (or simply word). In some memory systems, the word could be a 4 bit, 8 bit or 16 bit long. But in modern computers, the memory system being used is 32 bit or even 64 bit long.

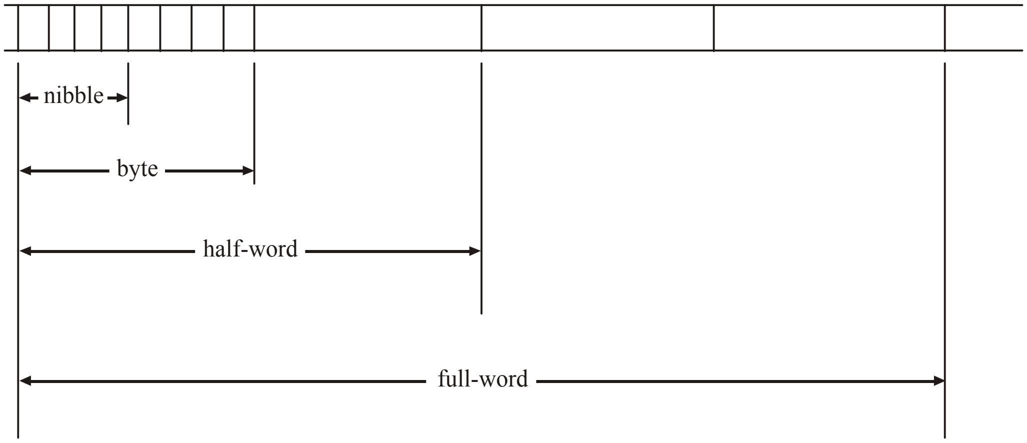


Fig. 11-2. Representation of bits and bytes.

Each location in the memory will be having one word of digital information. So this word will be having the information to perform a particular operation. The processor will use the particular address to read or write data into the memory.

The size of the memory is measured in terms of total number of storage locations (*i.e.* bytes) in it. Normally 1024 bytes form one kilo byte (kB), 1024 kB forms 1 Mega Byte (1 MB).

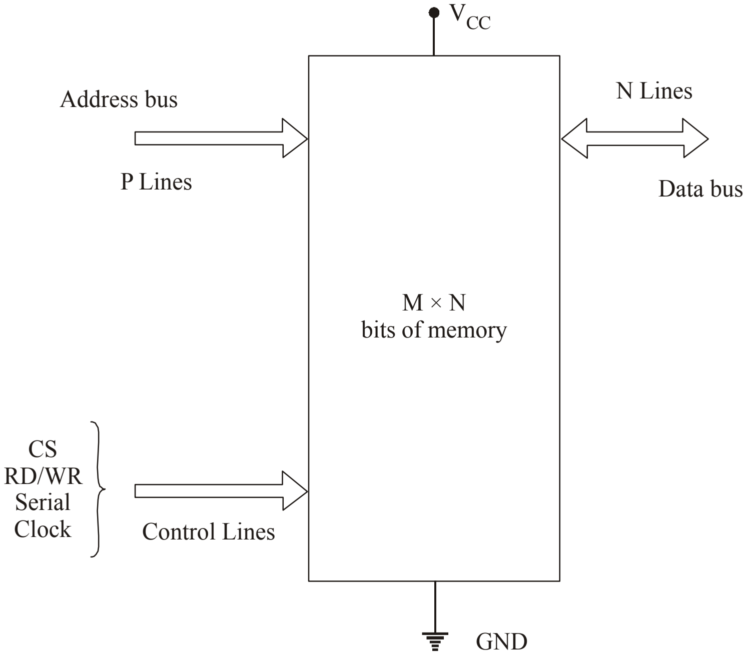


Fig. 11-3. Memory system block diagram.

Fig. 11-3 shows a block diagram of a memory device with M number of locations and N number of bits in each location. The size of this memory device is M × N bits. This means that M words of N bits each can be stored in the memory. As mentioned earlier, the common values of word sizes are 4, 8, 16, 32 and 64 bits. The commonly available size of the memory devices are 64, 256, 512, 1024, 2048 bytes and so on.

Each location (M) of the memory is defined by a unique address. To access any location p input lines are required where $2^p = M$. These set of lines are called address bus or address inputs.

It may be carefully noted that address bus is unidirectional and data bus is bidirectional. It means that same number of lines are used to write data into a memory location or read data from the memory location. Read/Write control lines decide whether the data to be written or to be read. Chip select and serial clock are also part of the control lines. In addition to the inputs, another two pins are required to provide power and connecting the device to the ground.

11-4. Write Operation of Semiconductor Memory

As a matter of fact, semiconductor memory acts as a main memory in modern computers. This main memory communicates with the CPU (Central Processing Unit) of the computer. The computer's main memory consists of many ICs connected together. There are three different groups of signal lines: (1) The group of address lines (called address bus), (2) The groups of data lines (called data bus), (3) The group of control lines (called control bus). These three group of signal lines (or buses) are connected to the CPU as shown in Fig. 11-4. Each bus will be having different number of lines based on the computer's requirements. These three buses are responsible for read and write operations between the CPU and the memory.

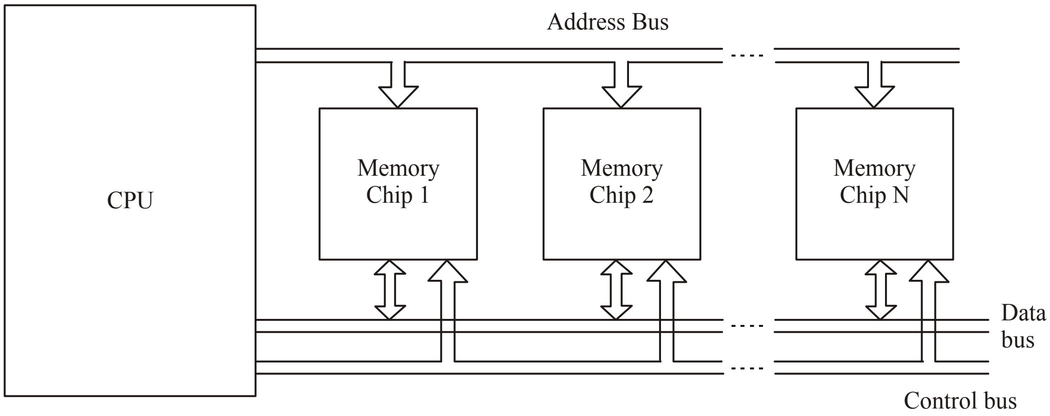


Fig. 11-4. CPU connected with memory chips.

To write data into a particular memory location, a suitable combination of 0s and 1s will be filled at a particular memory address. Fig. 11-5 shows the timing diagram for a typical write operation.

Here at time t_0, the CPU gives a new address where the data is to be stored. Then the CPU controls $\overline{RW}$ and $\overline{CS}$ lines to logic 'LOW' after a time interval of t_{AS} (also called as address setup time). This gives time to address decoder to respond to the new address. These $\overline{RW}$ and $\overline{CS}$ are kept logic 'LOW' for a time interval t_w (also called as write pulse time). During this write pulse time, at time t_1, the CPU supplies valid data to the data bus to be written into the memory. This data will be there at the input for atleast the time interval t_{Dw} (also called as data setup time) prior to, and for atleast a time interval t_{DH} (also called as data hold time) after the deactivation of $\overline{RW}$ and $\overline{CS}$ at time t_2. The address input remains stable after time t_2, for the time interval t_{wr}. If any of these setup time and hold time requirements are not met, the write operation will not take place smoothly. So the

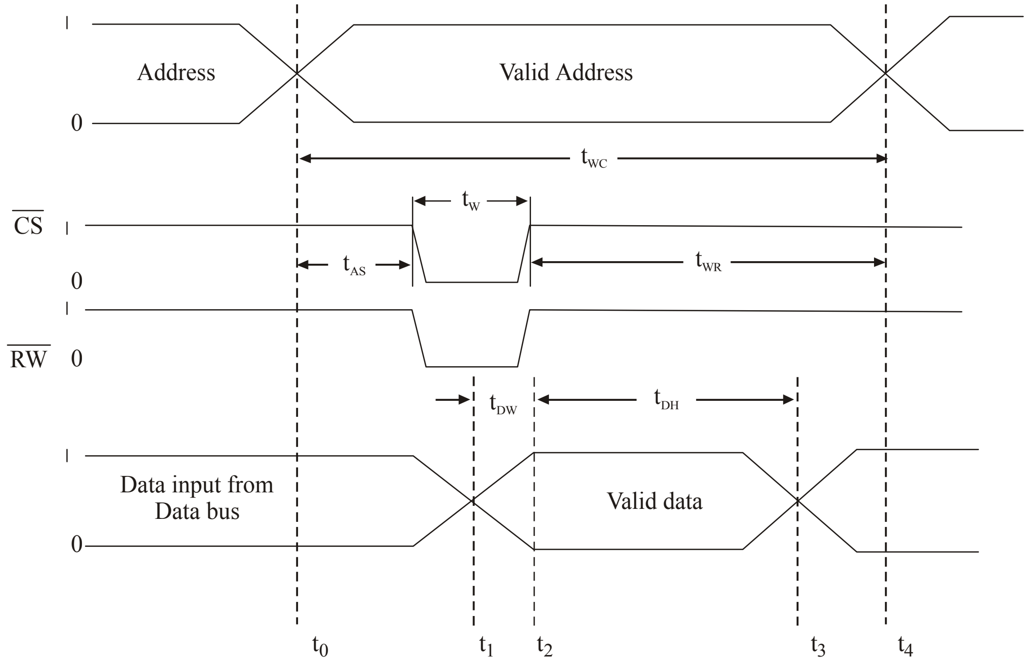


Fig. 11-5. Typical write cycle diagram.

complete write cycle time t_{wc} extends from t_0 to t_4, when the CPU changes the address lines to a new address for the next write cycle. Table 11-1 shows the different time durations for the write operations.

Table 11-1. Timing characteristics of write cycle operation

No.	Timing Parameter	Minimum Value (ns)
1.	Write Cycle time t_{WC}	35
2.	Address setup time t_{AS}	0
3.	Write recovery time t_{WR}	0
4.	Data valid till end of write t_{DW}	10
5.	Data hold time t_{DH}	0

Following are the important operations to be done during the write operation.

(*i*) Initiate a chip select signal by making chip select signal $\overline{CS} = 0$.

(*ii*) Apply the binary address of the memory location where the data is to be stored. It places the address on the address bus lines.

(*iii*) Apply the write control signal by making $\overline{RW} = 0$ for a certain period of time.

(*iv*) Apply the data to be written at the selected memory location to the data bus for a certain time.

Then the memory ICs decode the binary address to determine which location is being selected for the store operation. Then the data on the bus is transformed to the selected memory location.

Following are the timing parameters used in the write cycle operation, which are important from the subject point of view.

(*i*) Write Cycle Time (t_{wc}) : It is the minimum amount of time for which the address of location must be present.

(*ii*) Write Pulse Time (t_w) : It is the minimum amount of time for which write pulse should be high.

(*iii*) Write Release Time (t_{wr}) : It is the minimum amount of time for which address of desired location must be present after the end of write pulse time.

(*iv*) Data Setup Time (t_{Dw}) : This is the minimum amount of time for which data must be present before write pulse ends.

(*v*) Data Hold Time (t_{DH}) : This is the minimum amount of time for which data must be present after write pulse ends.

11-5. Read Operation of Semiconductor Memory

Whenever a computer is executing a program of instructions, the CPU normally continuously reads information from the locations in the memory that contains,

1. Program codes representing the operation to be performed.
2. The data to be operated upon.

So whenever the CPU wants to read data from a specific memory location, a set of operations is to be performed. Fig. 11-6 shows the timing diagram of a typical read operation.

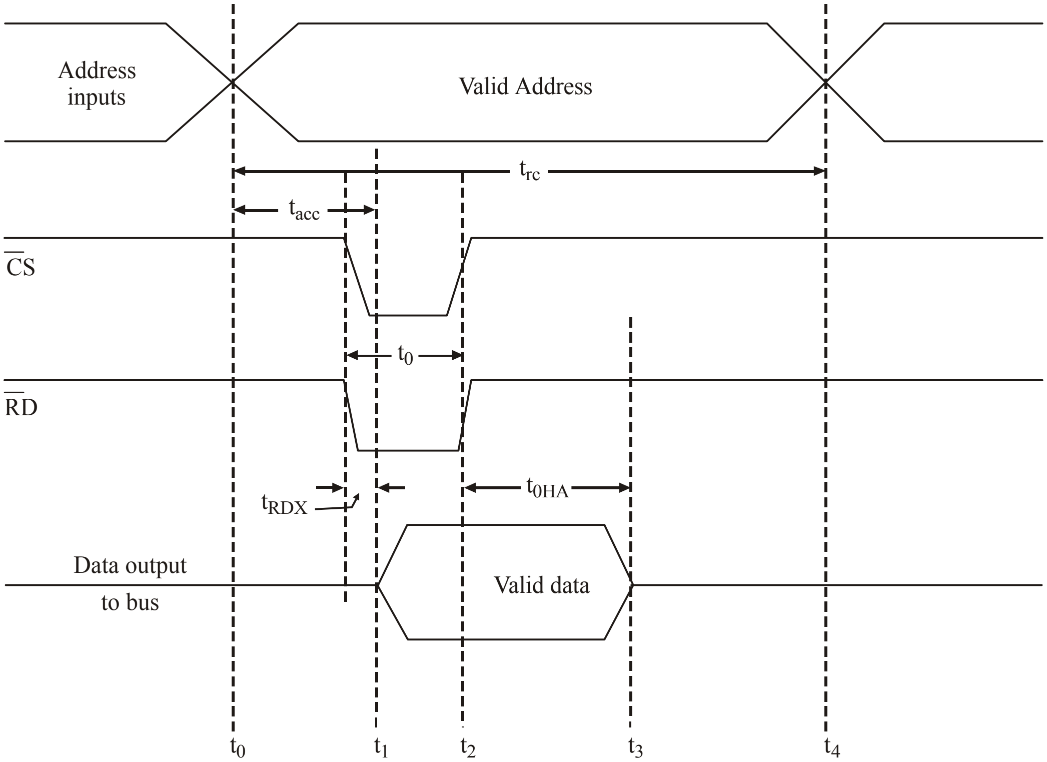


Fig. 11-6. Typical read cycle waveform.

The read cycle starts at time t_0. At time t_0, the CPU gives new address to the input from where the address of location to be read. After allowing some time to stabilize the address signals, the chip-select $(\overline{CS})$ is getting activated. So now the output data is placed at time t_1. Here the time between t_0 and t_1 is called the access time (t_{acc}). It is the time delay between the actual address input and the valid data output. Here t_{RDX}, is the time it takes for the data output from low to valid data output when the chipselect $(\overline{CS})$ is activated.

At time t_2, when the chipselect $(\overline{CS})$ HIGH, the output data returns to the original state after a time interval of t_{OHA}. So the original data output will be between t_1 and t_3. So the CPU can take data from the databus at any point during this time. The complete read cycle (t_{rc}) will go from t_1 to t_4, when the CPU changes the next different address input for the next read cycle.

Table 11-2 shows the read cycle operation timing details.

Table 11-2. Timing details of read cycle operation

No.	Timing Parameter	Value (ns)
1.	Ready cycle time t_{rc}	min 35
2.	Address access time t_{acc}	max 35
3.	Output hold from address change t_{OHA}	min 4
4.	Enable access time t_{RDX}	max 35

So, when the CPU wants to read data from a specific memory location, following operations must occur.

(*i*) Supply a binary address from which data to be retrieved to the address bus.

(*ii*) Initiate the chipselect signal by making $\overline{CS} = 0$.

(*iii*) Apply the read control signal by making $\overline{RD} = 0$ for a certain time.

Then the CPU activates the appropriate control signal lines for the memory read operation. The memory ICs place data from the selected memory location onto the databus, from which they are transferred to the CPU.

Following are timing parameters used in the read cycle operation, which are important from the subject point of view.

(*i*) Read Cycle Time (t_{rc}) : It is the minimum amount of time for which address must be present for reading data from the memory.

(*ii*) Reed pulse time (t_r) : It is the minimum amount of time for which read command signal must be active low.

(*iii*) Read to output active time (t_{RDX}) : It is the minimum amount of time delay between the beginning of read pulse and the output buffers coming to active state.

(*iv*) Data Hold time (t_{OHA}) : It is the minimum amount of time for which valid data is present at data output bus after the address ends.

11-6. Expansion of Memory Size

In many applications the required memory capacity or word size cannot be satisfied by a single memory chip. So for this expansion, several memory chips must be combined together to provide the necessary capacity or word size.

Memory sizes may be expanded by the following two methods, namely :

(1) Expansion of memory size by increasing the word size, and

(2) Expansion of memory size by increasing the word capacity.

Now we will discuss about these two types in detail in the following few pages.

(*i*) Expanding word size : Fig. 11-7 shows the expansion of memory size by increasing the word size. If the available word size of the memory is M, and if want to have the word size of M (that is m greater than M) then similar M word size of ICs can be connected together. The number of ICs connected to be an integer, and it should have the value m/M. The chips should be connected in the following way.

(*i*) Connect all the address lines of the ICs together. A_0 pin of each ICs should be connected together. Then we will have only one output corresponding to each pin of ICs *e.g.* A_0 is the common address line for all ICs.

(*ii*) Connect RD/WR pin of each memory chip together.

(*iii*) Connect all ChipSelect (CS) pins together.

Here the number of data input/output lines will be the product of total number of chips used and the word size of each chip.

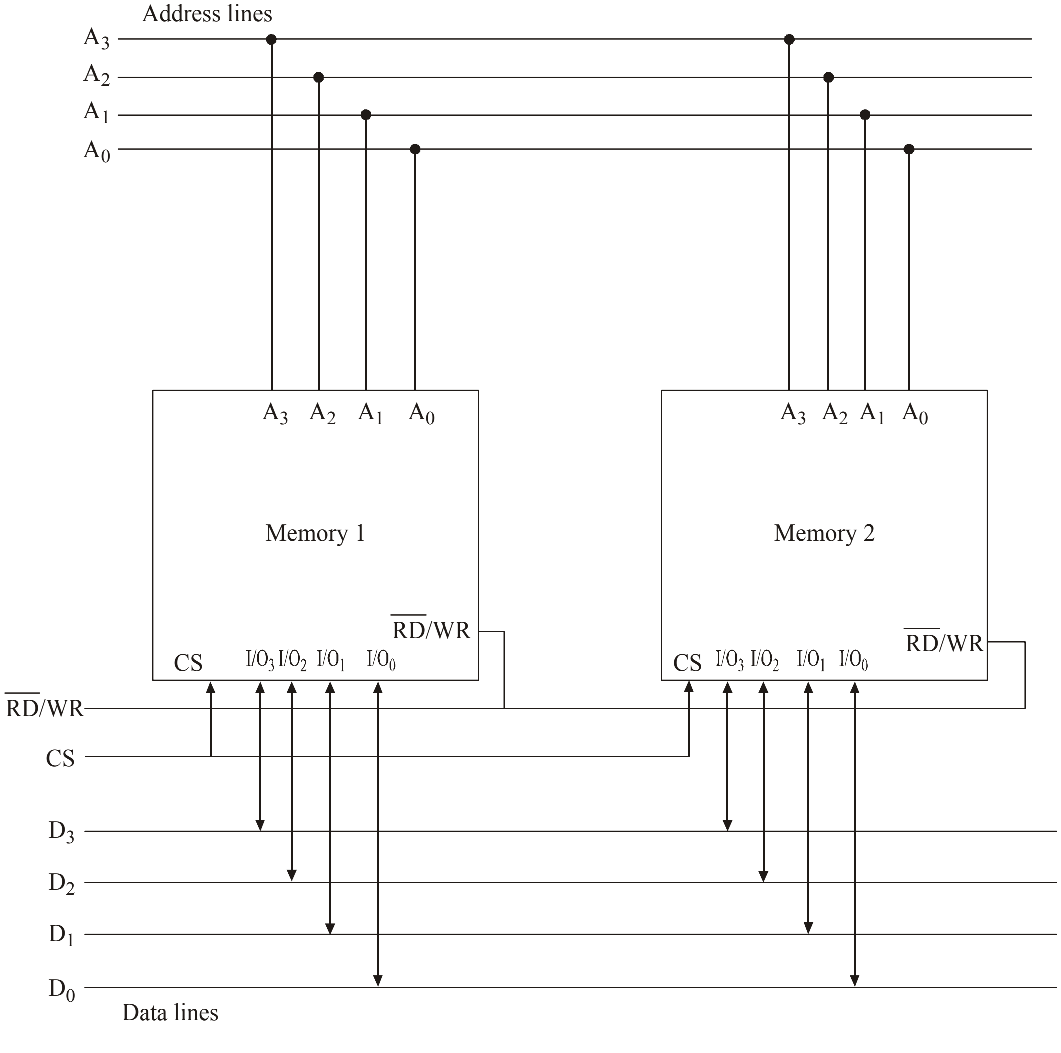


Fig. 11-7. Memory expansion by word size.

(*ii*) Expanding by word capacity :

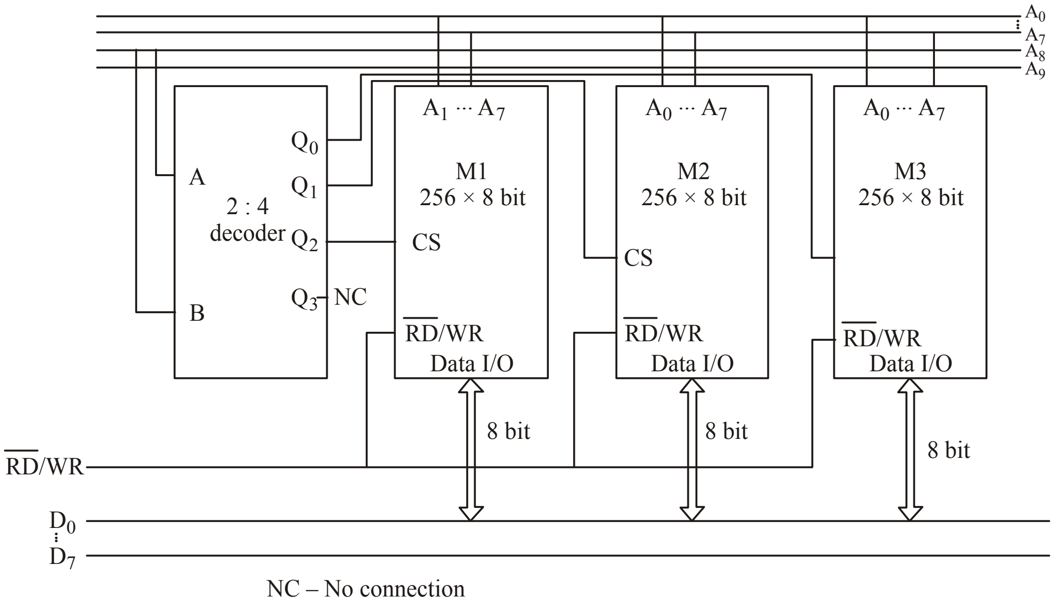


Fig. 11-8. 1024 & 8 bit formed by using three 256 × 8 bit memory.

Fig. 11-8 shows the expansion of memory size by using a word capacity. Here also the memory ICs will be combined together to get a required capacity.

Here three memory ICs of 256 × 8 bit are connected together with 2 × 4 decoder. If we want to have a memory of capacity '*n*' words then the current memory ICs of N words should be combined together to get a proper size. The ICs should be connected in the following way to get the required capacity.

(*i*) Connect all the address lines of each IC together.

(*ii*) Connect all the $\overline{RD}/WR$ pins together.

(*iii*) Connect a decoder circuit and connect each of its output to $\overline{CS}$ pin of each memory IC.

11-7. Classification of Semiconductor Memory

Memory devices can be classified into many types based on the principle of operation, its physical characteristics, access type and many more features. But the following are important from the subject point of view.

1. Sequential Semiconductor Memory
2. Random Access Memory (RAM)
3. Read Only Memory (ROM)
4. Content Addressable Memory.

Now we will study all the above types of memory in the following few pages.

1. Sequential Semiconductor Memory : In this type of memory, memory locations are arranged one after the other. Magnetic tapes and video cassettes are examples for this type of memory. Here writing data into the memory and reading data from memory will be in series fashion. So time required to read or write a data from the memory location will be different for different locations. The sequential memory is of two types, namely :

(*i*) Shift register memory and

(*ii*) Charge Coupled Devices (CCD)

These two types are explained in articles 11-9 and 11-10.

2. Random Access Memory (RAM) : It is also known as Read/Write memory. In this memory, time required to read or write a data into the memory location will be same. It requires external power to maintain the memory content. The term random access refers to the ability to read/write any memory location directly. It can be further classified into,

(*i*) **Dynamic RAM (DRAM) :** DRAM is a RAM device that requires periodic refreshing to retain its content.

(*ii*) **Static RAM (SRAM) :** SRAM is a RAM device that retains its content as long as power is supplied by an external power source. SRAM does not require periodic refreshing and it is faster and more importantly it is expensive than DRAM.

3. Read Only Memory (ROM) : This memory can only be read but cannot be written or re-written by the CPU. Because of this operation, it is called as Read Only Memory (ROM). It has non-volatile content and does not require any external power source to keep the data.

Read only memories are further classified into many types based on the techniques used to store the data. These are as follows :

(*i*) **Mask Programmed ROM :** The memory content is programmed during manufacturing process. Once programmed the content cannot be changed. It cannot be re-programmed. This is explained in detail in Article 11-12.

(*ii*) **Field Programmable ROM (PROM) :** The memory content can be custom programmed one time. The memory content cannot be changed once programmed. This is explained in Article 11-13.

(*iii*) **Erasable Programmable ROM (EPROM) :** An EPROM device can be custom programmed, erased and re-programmed as often as required within its lifetime (hundred or even thousands of time). The memory content is fixed once programmed. Traditional EPROM devices are erased by exposure to ultra-violet (UV) light. An EPROM must be removed from its housing unit first. It is then re-programmed using a special hardware called EPROM programmer. This type of memory is explained in Article 11-14.

(*iv*) **Electrically Erasable Programmable ROM (EEPROM or E^{2}PROM) :** Modern EPROM devices are erased electrically. So it is called as EEPROM. One important difference between EPROM and an EEPROM device is that with EEPROM device, memory content of single byte can be selectively erased out and programmed. Therefore with an EEPROM device, we can alter the already existing data for our requirements. Another difference is that EEPROM can be re-programmed without a special programmer and can stay in the device while being programmed. This type of memory is explained in Article 11-15.

(*v*) **Flash Memory :** The flash memory is a variation of EEPROM, which allows for block level (for example 512 byte) programmability that is much faster than EEPROM. Nowadays flash memories are used in mobile phones, cameras and many other electronic appliances. This type of memory is explained in detail in Article 11-16.

4. Content Addressable Memory : It is a special purpose random access memory that performs association operation in addition to read and write operation. This memory is explained in detail in Article 11-23.

11-8. Volatile and Non-Volatile Memory

Semiconductor memory may also be classified as volatile and non-volatile memory. The memory is volatile in its contents or data stored on it exists as long as electric power is available. If the power supply is cut off, then the information stored in the memory will be lost. A read and write memory of such type is called a volatile memory.

The non-volatile memory is the memory in which data remains intact even if electric power is switched off. All types of ROM are non-volatile memory. All the secondary memory is also non-volatile. So if we want to store the information for longer time, then we have to store in the non-volatile memory.

11-9. Shift Register Memory

In shift register memory, the data write and data read are done in a sequential manner. The access time required to read a data from memory is not same as to write a data into the memory. It means that, if *n*th location is being accessed at a particular time, then (*n* + 2)th location is not accessible, unless all the intermediate locations have been accessed one by one. Since the average access time is quite long, this memory is very slow to respond.

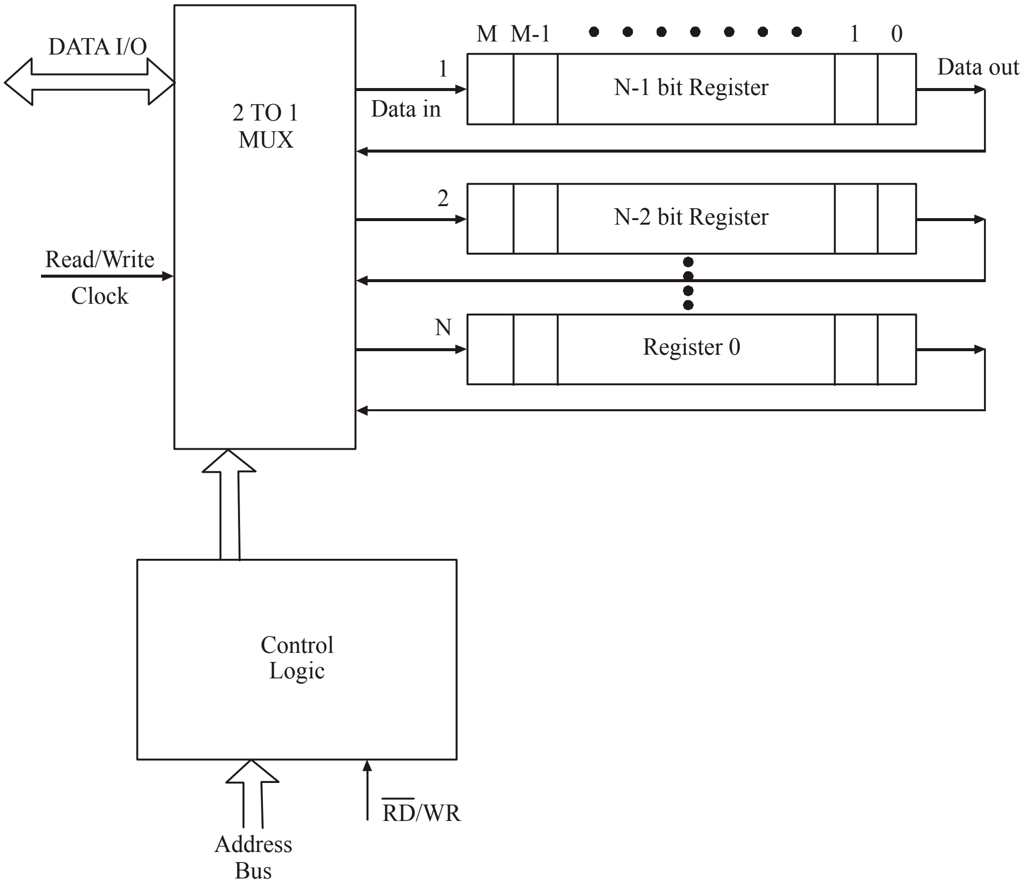


Fig. 11-9. M × N bit shift register memory.

A shift register memory of size M × N is shown in Fig. 11-9. As seen from the figure, the shift register memory has ‘N’ number of register and each register can accommodate ‘M’ number of bits. With every read/write clock, the data is entered in and read out from shift register. When any data bit enters the shift register, each bit in register moves either left or right towards the position at which bit entered. If the bits in register move towards right, then this configuration is called FIFO (first in first out) and memory is called FIFO shift register memory. If the bits in register move towards left as any new bit entered, then the configuration is called LIFO (Last in first out) and memory is called LIFO shift register memory.

Here the control logic selects the desired location and decides whether to read or write. When $\overline{\text{RD}}/\text{WR} = 0,$ the control logic operates in read mode and the data is read from the shift register. When $\overline{\text{RD}}/\text{WR} = 1,$ then the data is entered into the shift register.

11-10. Charged Coupled Device (CCD) Memory

Charged coupled device memory is an array of Metal Oxide Semiconductor (MOS) capacitors. That means, its base is constructed with a material of a good conductor and is topped with a layer of metal oxide. Normally silicon is used as a base material and silicon dioxide is used as a coating. The top layer is made of silicon-polysilicon. Fig. 11-10 shows an array of CCD channels. Transfer of charge from one channel to the other is in sequential manner. The data is stored in the form of capacitors. Small charge denotes logic 0 and large charge denotes logic 1.

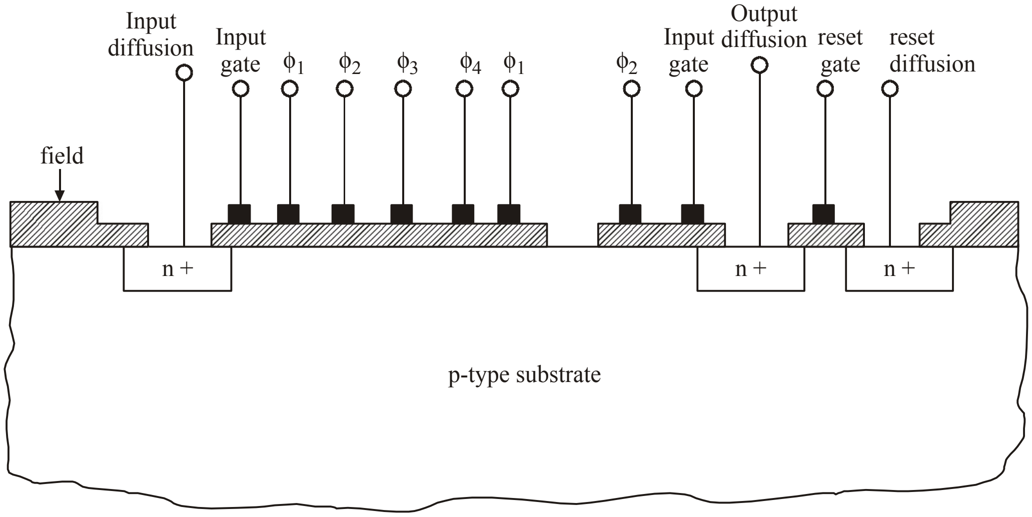


Fig. 11-10. CCD Memory arrangement.

The charge mores from one channel to other channel in one direction only. During the charge transfer along the channel, some charge is always lost. Therefore it is necessary to refresh the charge periodically. For this, a refresh circuit is used to retain the leakage charge. The whole device is a long array of MOS devices in which all the ϕ_1 electrodes are connected together. All the remaining electrodes are connected in similar arrangement.

The four clock waveforms used to drive the circuit is shown in Fig. 11-11. During the time interval t_1, only the electrode ϕ_1 is at positive voltage. A positive voltage allows the current through any P-N material and the depletion regions are formed under ϕ_1 electrode. The charge in depletion region under ϕ_1 is obtained by either injecting from outside source or from the previous electrode.

During the time interval t_2, the depletion region under ϕ_1 electrode remains same while the new depletion region generated under electrode ϕ_2, because voltage at ϕ_2 becomes positive. At time interval t_3, all three electrodes ϕ_1 , ϕ_2 and ϕ_3 become positive. So the depletion regions are generated from electrodes ϕ_1 to ϕ_3. Because of this, the charge is able to spread through the extended regions. At time interval t_4, the voltage at ϕ_1 becomes 0 and this will eliminate the depletion regions under the ϕ_1 electrode. At time interval t_5, the voltage at ϕ_2 becomes 0 and the depletion region under ϕ_2 will be eliminated. Now the voltage at electrodes ϕ_1, ϕ_3 and ϕ_4 become positive.

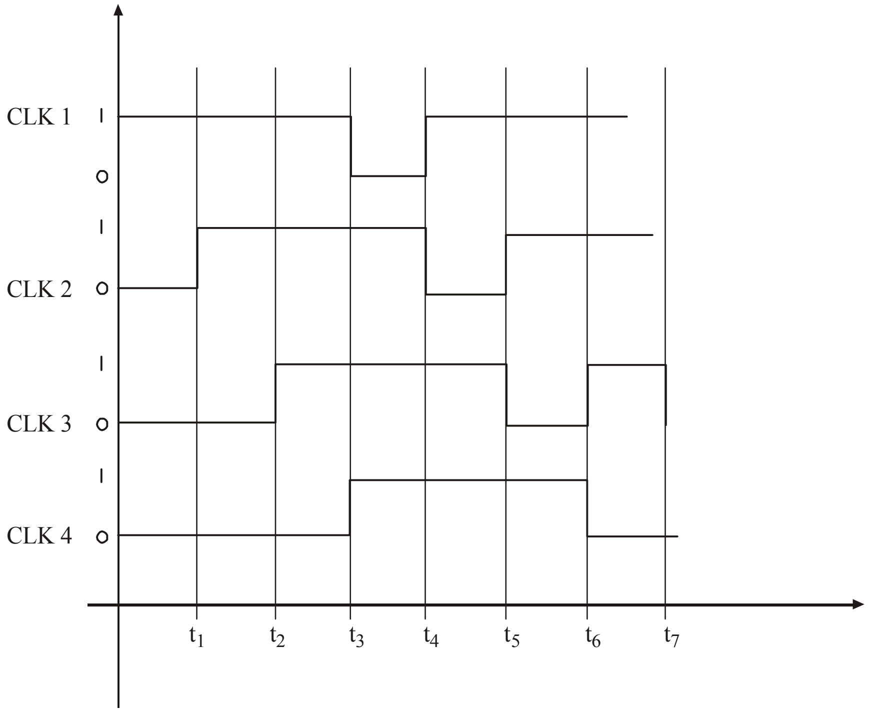


Fig. 11-11. Clock waveform for CCD device

Now, we can understand that after four intervals, the charge shifted from one channel to the preceding channel. So the injection and detection of charge must be done in synchronism with the clock waveform.

11-11. ROM Organization

The ROM (*i.e.*, Read Only Memory) is a semiconductor memory in which the data written into the memory location is permanent. It is non-volatile. During normal operation no new data can be read from ROM, but data can be read from ROM.

Fig. 11-12 shows the ROM organization containing 16 bit. From the figure, we know that it is an array of selectively open and closed diodes.

To select any memory location in this 16 bits, a 4 bit address ($A_3 A_2 A_1 A_0$) is required. The lower half locations $A_1 A_0$ are decoded by Decoder 1 from any one line of the four rows. The upper half locations $A_3 A_2$ are decoded by Decoder 2 which selects any one line from the four columns. The diode matrix is formed by connecting one diode with a switch between each row and column. For example, the diode D_{22} is connected between row 2 and column 3. The output will be enabled by applying logic 1 at the chipselect input. For example, when row 2 is connected with column 2, the address 1010 is obtained by closing the diode switch D_{22}. This implies that logic 1 is stored at the address 1010. If the diode switch D_{22} is open, then logic 0 is stored at the address 1010.

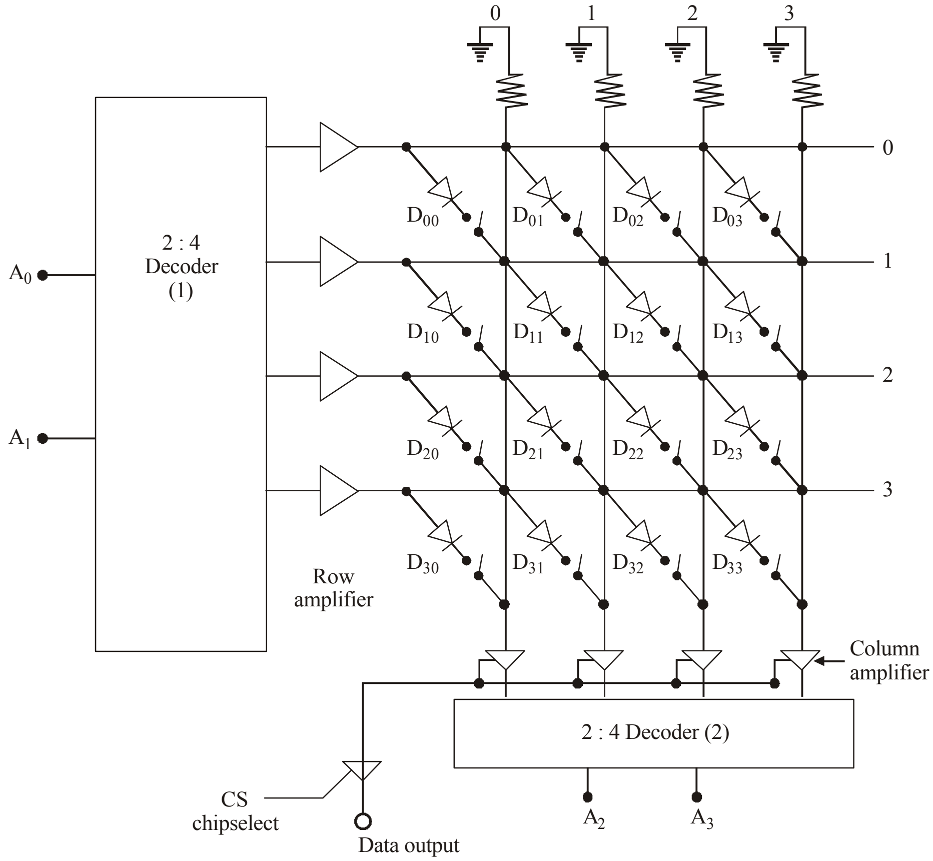


Fig. 11-12. 16 bit ROM organization.

Fig. 11-13 shows the timing diagram for the ROM read operation.

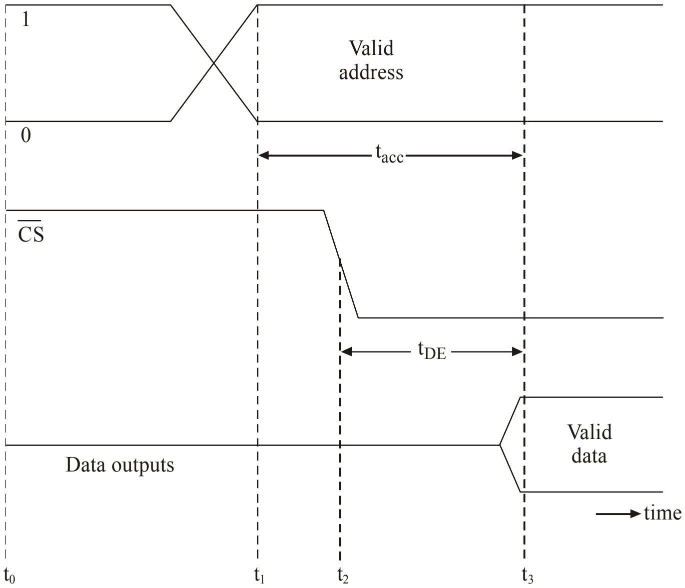


Fig. 11-13. Typical ROM Read timing diagram.

According to this figure, at time t_0 input address, chipselect and data outputs are at some levels. At t_1, a new valid address is given. At this point, the internal ROM circuitry begins to decode the new address and to send the data to the output buffer. At t_2, chipselect input $(\overline{CS})$ is activated to enable the output buffers. At time t_3, we get the valid output data.

There are two important parameters in the ROM read operation. They are,

(*i*) Access time (t_{acc}) : The delay between the ROM's inputs and the appearance of data outputs during a read operation.

(*ii*) Output Enable Time (t_{OE}) : Time delay between the chipselect input and the valid output.

Depending upon the programming process, the ROM memory may be categorized as,

(*i*) Mask programmable read-only memory (MROM),
(*ii*) Programmable read-only memory (PROM),
(*iii*) Erasable programmable read-only memory (EPROM), and
(*iv*) Flash read-only memory.

Now we will discuss about these types in detail.

11-12. Mask Programmable ROM (MROM)

The mask-programmed ROM has the data stored into it permanently by the manufacturer according to the customer's requirements. Electrical interconnections on the chip is done by a photographic negative, also called as 'mask'. A different set of mask is required to store different type of information. These masks are very expensive and it will be very useful, if we want to manufacture large quantity of ICs.

Fig. 11-14 shows 16 × 4 Mask read-only memory, also called as 16 × 4 binary-to-gray converter. This will explain how MROMs can be used as a data converter or a look-up table. The figure shows the binary and gray code data. The MROM decoder is addressed by the binary data and the gray code data are stored in the memory. Fig. 11-15 shows the permanent programming of Logic 1 and Logic 0 using bipolar transistors in the memory of MROM IC. This can also be done by unipolar MOSFET. Every row in MROM contains four intersect points.

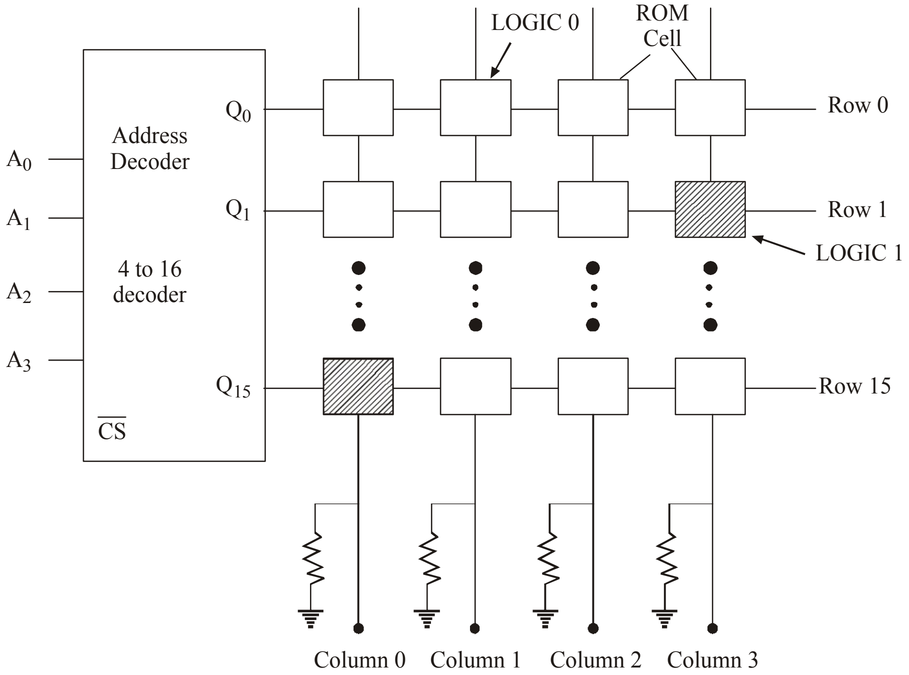


Fig. 11-14. 16 × 4 bit Mask Read-only Memory.

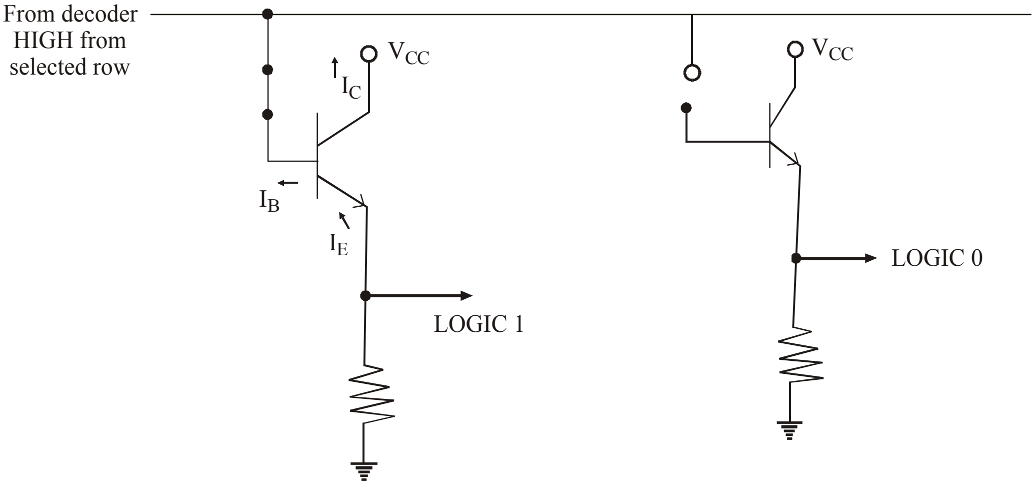


Fig. 11-15. (*a*) Generation of LOGIC 1 (*b*) Generation of LOGIC 0

Fig. 11-15 shows the circuit of LOGIC 1 and LOGIC 0 generations. When the chipselect is selected, the 1-of-16 decoder produces an active high output for the specified row address. This high is applied to each of the base circuit of transistor after a selected row. When a high is applied to its base from the decoder, the transistor conducts through the output register shown in the figure and thus produces the Logic 1 level.

In the LOGIC 0 generation, the connection from the decoder output line to the base of the transistor has not been made. This will be done by masking during manufacturing process. Since the base is not connected, the transistor will cut off from the selected row. Since no current flows through the resistor, LOGIC 0 output is generated. If the decoder is not enabled, all the output lines remain low. This causes all transistors in the matrix to cut off. Table 11-3 shows binary code and gray code data of MCOM behaviour.

Table 11-3. Binary Code and Gray Code representation

Binary Code	Gray Code	Binary Code	Gray Code
0000	0000	1001	1101
0001	0001	1010	1111
0010	0011	1011	1110
0011	0010	1100	1010
0100	0110	1101	1011
0101	0111	1110	1001
0110	0101	1111	1000
0111	1000		

11-13. Programmable Read-Only Memory (PROM)

A mask programmable ROM is very expensive and can be used only in high volume applications. So for lower volume applications, manufacturers have developed ROMs with fusible links. These fusible links can be modified by the user based on the requirements. But once programmed, this PROM is like a MROM and that cannot be erased or re-programmed again. So if the program in the PROM is faulty or if we want to change the program, then we must throw away the PROMs. So for this reason, these PROMs are referred as “one time programmable” (OTP) ROMs.

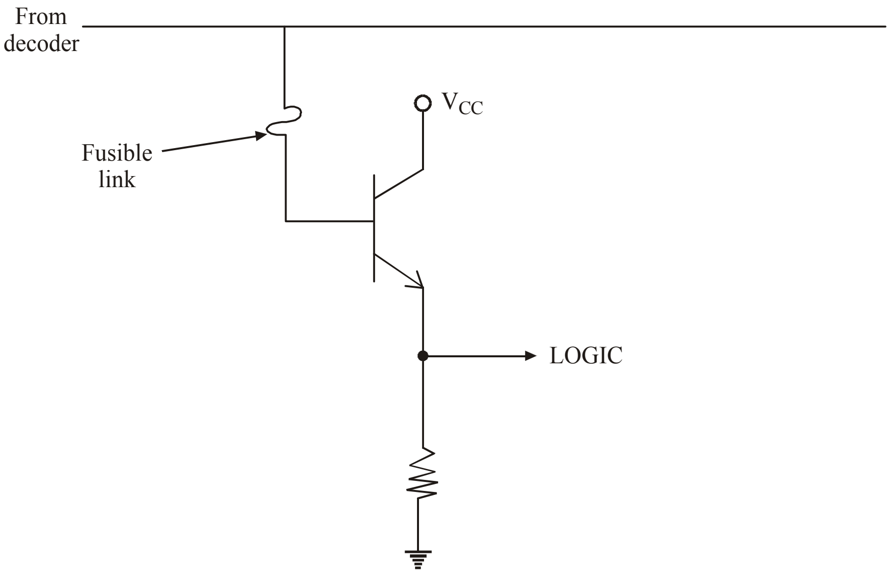


Fig. 11-16. PROM with a fusible link.

Fig. 11-16 shows the PROM transistor with a fusible link. This operation can be compared with figure of Logic 1 and Logic 0 generation circuits. When the fusible link is intact and the decoder output is high, then the transistor conducts and it develops Logic 1 output. When the fusible link is blown, the transistor remains cut off even though the decoder selects a row and thus develops Logic 0.

The user can program these PROM chips by a PROM programmer or PROM burner. This will blow the selected fuse links based on the requirement of the user. Programming a PROM like this is called as burning or burning-in aprogram.

11-14. Erasable Programmable ROM (EPROM)

An EPROM can be programmed by the user, and that can be erased and re-programmed whenever required. So once programmed it will become as non-volatile and can store the data indefinitely. EPROM will be programmed by giving voltage in the range of 10-to-25V for about 50 ms per address location to the appropriate chip inputs. Normally this program will be done separately designed circuit.

The storage cells in an EPROM are MOS transistors with a silicon gate without any electrical connection. It does not have any floating gates. Normally, each transistor is OFF and each cell is storing a logic 1. A transistor can be turned ON by applying high voltage pulse, which generates high-energy electrons into the floating gate region. These electrons will remain there once the input pulse is stopped, since there is no discharge path. This keeps the transistor ON for permanently even though the power is not there, and now the cell is storing logic 0. During this programming process, the EPROM's address and data pins select which memory cells should be programmed to logic 0 and which memory cells should be left at logic 1.

This EPROM cell has a window on the top of the IC for Erasing. So by applying UV light through this window, it generates a photocurrent from floating gate that erases the stored charge. Normally 15 to 20 minutes will take to erase the complete IC and we cannot erase any selected cell.

These devices are more reliable and less expensive. Nowadays many small and medium volume products are using this type of EPROM. Its main disadvantages are, namely (*i*) should be removed from the circuit for programming, (*ii*) individual memory cell cannot be erased or programmed and (*iii*) it takes longer time to erase the IC.

11-15. Electrically Erasable Programmable ROM (EEPROM)

The drawbacks of EPROM devices are overcome by the introduction of EEPROM. The electrically erasable PROM (E^2PROM) is a non-volatile, static storage device that is electrically erasable. Now with this EEPROM devices, data can be erased by bit, bytes or by whole. The chip can be removed without removing from the circuit and it doesn't require any PROM programmer.

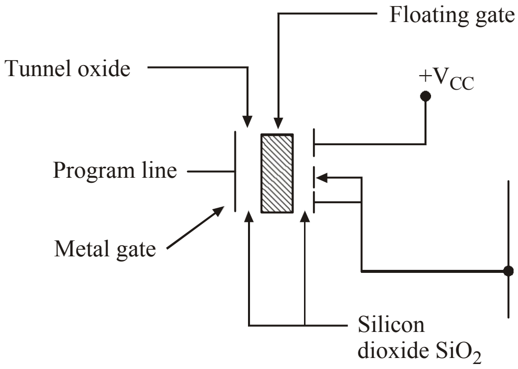


Fig. 11-17. Basic EEPROM memory unit.

EEPROMs are used in many applications nowadays. It is used in PCs, cameras, mobile phones, radios, music players and so many electronic appliances.

Fig. 11-17 shows the basic EEPROM memory system. This device consists of *n*-channel FET with an additional floating gate similar to the EPROM. The floating gate is isolated from the rest of device by silicon dioxide (SiO_2). An extremely thin tunnel oxide layer is used between the floating gate and the drain end of the channel. The charge is transferred to or from the floating gate by a process called Fowler-Nordheim tunneling. This quantum mechanical phenomenon occurs only between the floating gate and the drain, which will be referred to as the tunneling region.

11-16. Flash Read-Only Memory

Basically erasable PROMs (EPROM) provide high density typically (16K-4 MB) and relatively fast access times. But erasing takes longer time. EEPROMs work for lower densities (256-4KB) and slower read times. Because the data are read serially. But Flash memories are available in range of MBs and several Giga Byte capacities. Nowadays 32GB flash memory is available. These high densities are achieved because they use one transistor per cell. Its structure is same as EPROM technology. The oxide thickness between the substrate and the gate has been reduced in flash memory to a point that allows electrical erasure of data.

Normally flash memory offers power consumption of around 0.05 watt hour. But hard disk uses 1 watt hour. Flash memories access time to read the data is 60 ns. Its write time per byte is 10 ns. Its block erase time is 1.6 sec per 64 KB of block.

Normally two types of flash memories are available. They are NAND and NOR flash memory. They differ primarily by how they are getting programmed and erased.

1. NAND flash memory : It uses Fowler-Nordheim (F-N) technology for programming and erasing. This is used in EEPROM also. The low current used in F-N tunneling cases power supply requirements. In addition, the device receives less stress on the gate oxide, since the tunneling is from the channel. This produces an extended life of up to 1,000,000 program/erase in comparison to 100,000 of NOR type flash.

2. NOR flash memory : Here Hot-electron-injection (HEI) is used in this type of flash memory for programming. This process causes electrons to be injected from drain to the floating gate. HEI requires a 12V power source and uses much more current than NAND flash. It requires 10 μs/byte for programming.

Fig. 11-18 shows the basic block diagram of the flash memory.

Here for example, the block diagram of one flash memory (MN29N16)-National semiconductor) is shown in Fig. 11-18. By this diagram, we will know that each page has a memory of 256 bytes and 8 bytes for error correction. So totally 264 bytes/page is available. In a block, 16 pages are available and a device has 512 blocks. So total memory capacity,

$$264 \times 16 \times 512 = 2{,}162{,}688 \text{ bytes} = 17{,}301{,}504 \text{ bits.}$$

A block of memory (16 pages) stores 4K bytes of data. A block of 16 pages is the smallest unit within this device and can be erased. Both reading and writing can be done for individual pages of 264 bytes.

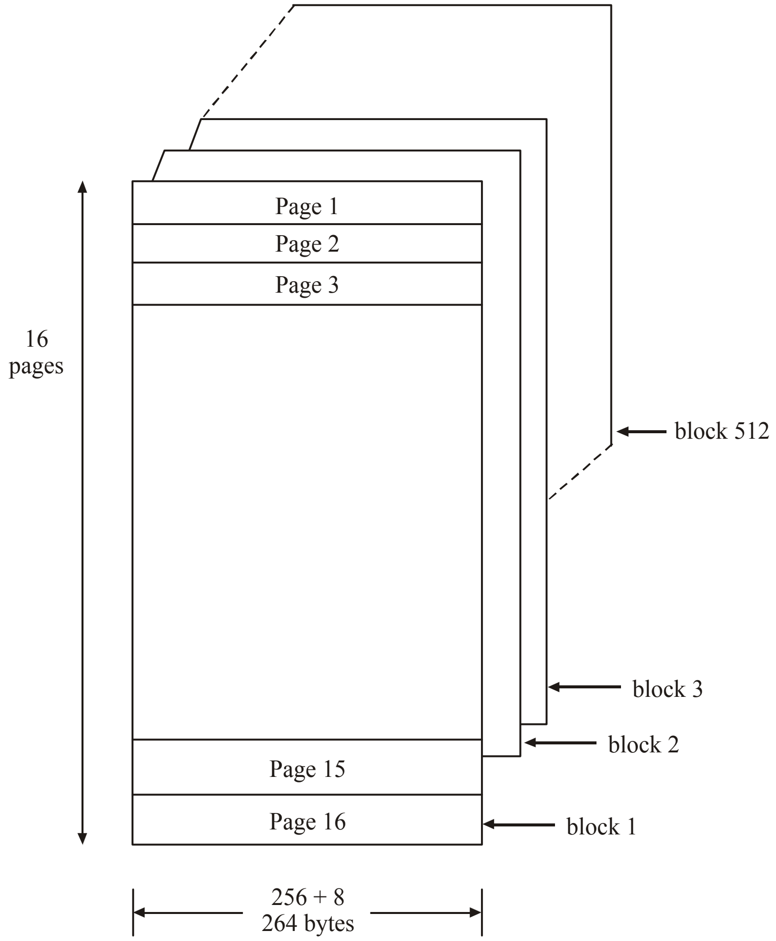


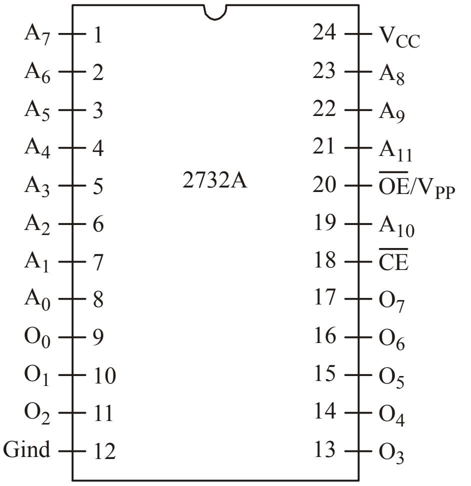
Fig. 11-18. Block diagram flash memory (MN 29N16-National semiconductor)

11-17. ROM ICs

We have already discussed about various types of ROMs and their behaviours. Now we will discuss about one of the commercially available ICs in the market.

27XXX is a popular ROM family of ICs. The IC's pin configuration and its pin descriptions are shown in Fig. 11-19. The 2732 EPROM has 12 address pin (A_0 to A_{11}) which can access 4096 bytes of words in the memory. The desired stored location is read by supplying the corresponding address. It uses a +5V supply and can be erased using ultra-violet light. After erasing, all the memory cells return to logic 1.

Erasing and programming an EPROM is handled by special equipment called PROM burners (programmers). For programming, the EPROM is brought in programming mode and new data is entered by changing memory cells to O's. A voltage of about 21V is applied at V_{PP}. The word to be programmed into the EPROM is addressed using 12 address lines. The $\overline{CE}$ input is like the chip select $(\overline{CS})$ inputs on some other memory chips. The $\overline{CE}$ input is activated low for about 50 ms. The $\overline{OE}/V_{PP}$ pin serves as a dual purpose. Under normal use the EPROM is being read, with $\overline{OE}$ pin active low. The eight output pins are labeled O_0 to O_7 on the 2732 EPROM. In writing mode, pin $\overline{OE}$ is activated high.

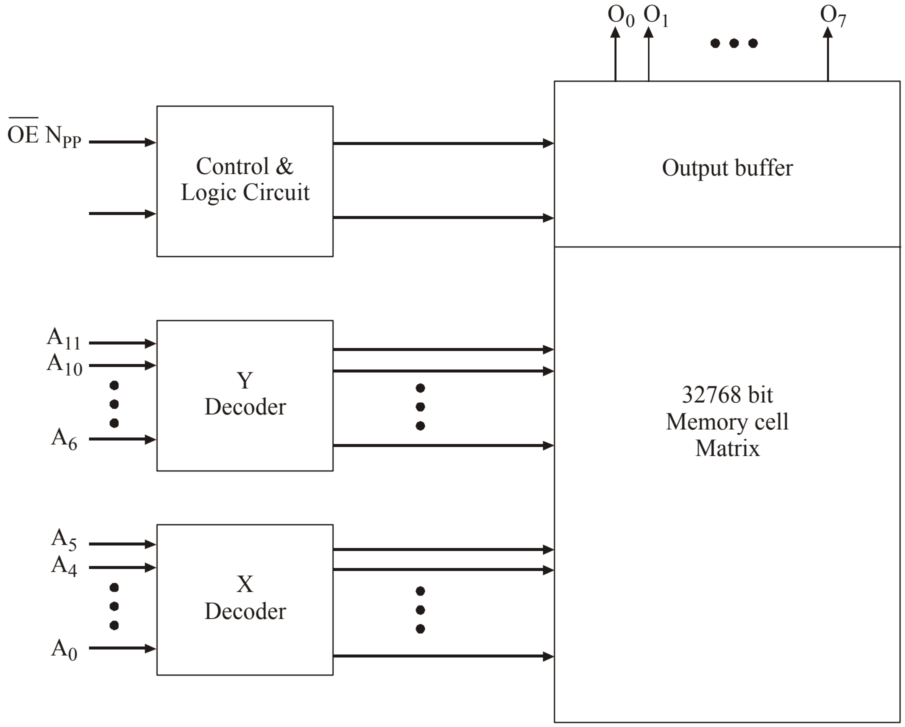


(*a*)

V_{CC}	+5 Volt DC
$A_0 - A_{11}$	Address Bus
V_{PP}	Programming Voltage
$O_0 - O_7$	Databus
$\overline{CE}$	Chip enable input

(*b*)

Fig. 11-19. (*a*) Pin configuration (*b*) Pin description



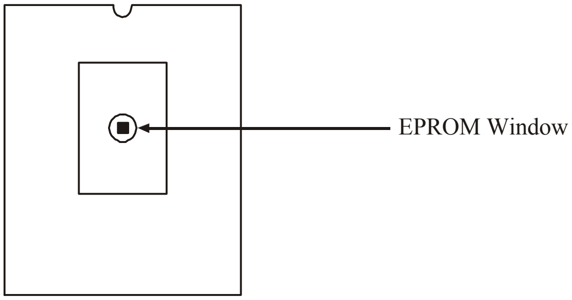


Fig. 11-20. Block diagram of EPROM.

This process is repeated for all memory locations. The IC chip has a window to erase the EPROM with UV light. After programming it is common to protect the EPROM window with an opaque sticker. This sticker protects the chip from unwanted UV rays and sunlights. Fig. 11-20 shows the block diagram of organization of 2732 EPROM and EPROM IC with a window.

Some of the commonly available commercial ICs are listed in Table 11-4. Please refer the manufacturer's datasheet for more information regarding their functionalities.

Table 11-4. Commonly available ROM ICs

IC No.	Memory Size	Type of ROM	Access time (ns)	Technology
74188A	32 × 8	MROM	450	Bipolar
2308	1024 × 8	MROM	460	MOS
825123	32 × 8	PROM	100	MOS
3608	1024 × 8	PROM	80	Bipolar
3605	1024 × 4	PROM	70	Bipolar
825147	512 × 8	PROM	70	MOS
475256	32K × 8	MROM	400	MOS
2704	512 × 8	EPROM	450	MOS
2708	1024 × 8	EPROM	450	MOS
2716	2048 × 8	EPROM	450	MOS
2732	4096 × 8	EPROM	450	MOS
2764	8192 × 8	EPROM	450	MOS
27256	32K × 8	EPROM	450	MOS
2816	2K × 8	E^2PROM	250	MOS
28256	32K × 8	E^2PROM	250	MOS
27F64	8K × 8	Flash ROM	100	MOS
27C201C	256 K × 8	Flash ROM	85	MOS

Example 11-1. *The memory capacity of a EPROM is 16 K × 32. Calculate the following: (a) Number of bits in each word (b) Number of accessible address lines (c) Number of locations it has and (d) Number of memory cells.*

Solution. Given : Memory capacity = 16K × 32.

(*a*) *Number of bits in each word*

We know that number of bits in each word is equal to the number of data lines. This in turn is equal to 32.

∴ Number of bits in each word = 32. **Ans.**

(*b*) *Number of accessible address lines*

Let n = number of accessible address lines

We know that the memory size,

$$16K = 2^n \Rightarrow 1K = 2^{10}$$

$$\therefore \quad 16K = 2^4 \times 2^{10} = 2^{14}; \quad \therefore 2^{14} = 2^n \quad \text{or} \quad n = 14 \textbf{ Ans.}$$

(*c*) *Number of memory locations*

We know that Number of memory locations,

$$= \; 2^{14} = 16,384. \textbf{ Ans.}$$

Note that this is the same as the memory size (16K).

(*d*) *Number of memory cells*

We know that number of memory cells

= number of memory locations × number of bits per word

= 16,384 × 32

= 524,288. **Ans.**

11-18. Static Bipolar RAM

Static bipolar RAM is a volatile memory. The basic storage cell is a flip-flop which simply consists of two cross-coupled transistors as shown in Fig. 11-21. The transistors Q_1 and Q_2 are provided with additional emitters which are used for addressing location. The emitters E_R and E_C are the output of row select and column select address decoders respectively. A bias voltage of 0.5 volt is applied to emitter E_D through resistance R_b.

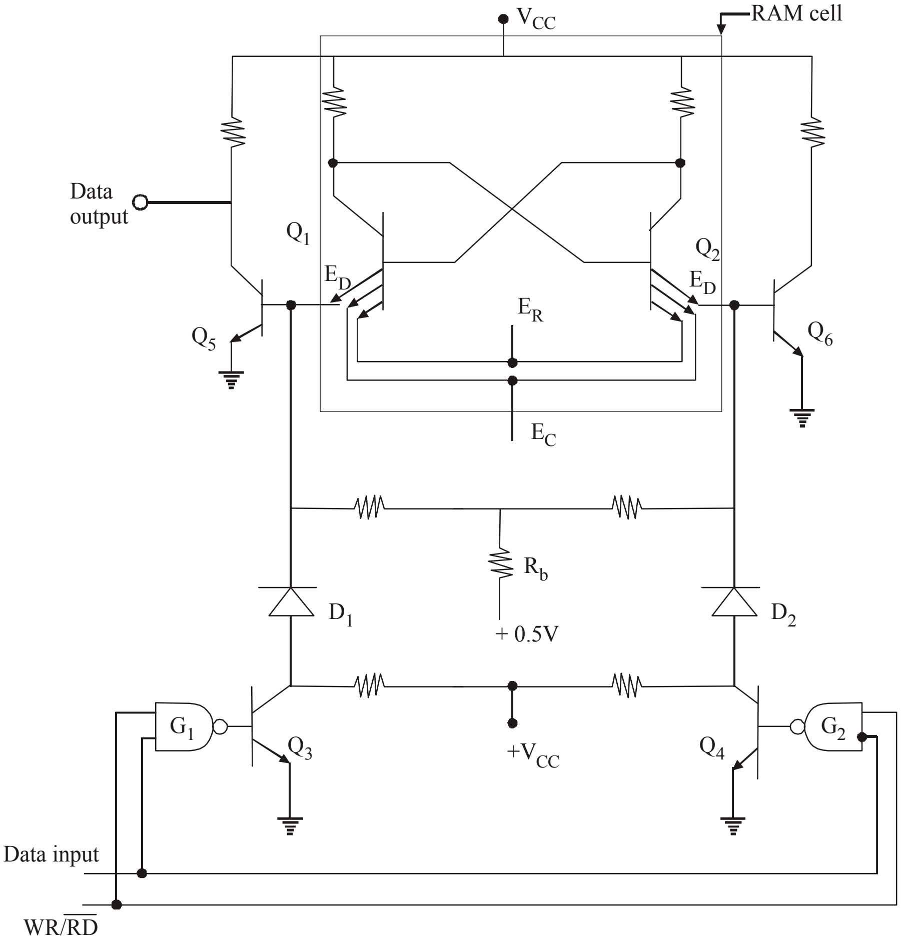


Fig. 11-21. Static Bipolar RAM.

The number of flip-flops required in a RAM is equal to the number of cells in it. Since the number of cells is usually very large, so it is required to use minimum devices per cell. This in turn is required to occupy minimum circuit per cell. This helps to reduce the chip area per cell in order to reduce the cost.

For read operation of RAM cell, $\overline{RD}/WR$ should be at logic 0. For all the address cells, E_R and E_C should be high with $\overline{RD}/WR = 0$. The AND gate G_1 & G_2 will be high, which turns ON the transistor Q_3 and Q_4. Therefore the diode D_1 and D_2 are non-conducting or reverse biased. Let the state of flip-flop is such that Q_1 is ON and Q_2 is OFF. As E_R and E_C are at logic 1, the current will flow in base of transistor Q_5, and the data output will be same logic level as present at the collector of Q_1.

For write operation of RAM cell, E_R, E_D and control pin $\overline{RD}/WR$ should be at logic '1'. If the data input is 1, the output of G_1 gate will be 1 and output of G_2 gate will be zero. Therefore transistor Q_3 is ON and Q_4 is OFF. In this case diode D_2 is forward biased which raise the voltage at $\overline{ED}$. Hence irrespective of original state (*i.e.*, Q_1 ON and Q_2 OFF) Q_2 cannot conduct. Hence logic level at collector of Q_2 will be logic level of data input.

11-19. Static MOS RAM

A static MOS RAM uses two cross-coupled MOS transistor as a basic memory cell. This memory is slower than the bipolar RAM, but easier to drive. Fig. 11-22 shows a static MOS RAM cell with read and write circuit. Here A_R is the row select control, A_C, the column select control, W, the write select and R and read select control respectively.

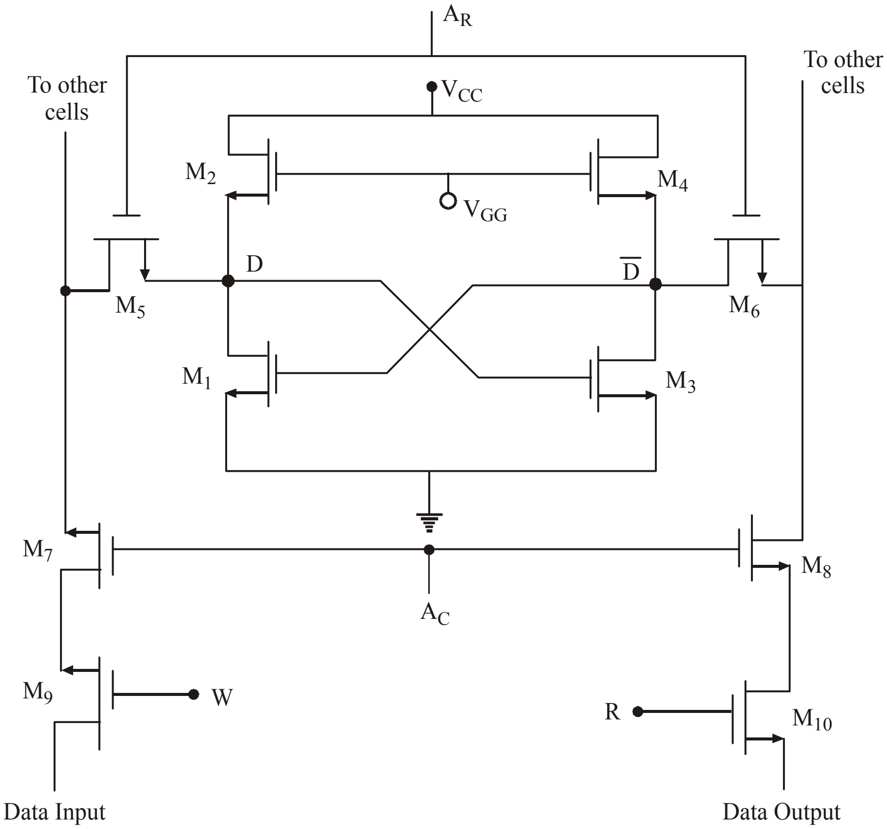


Fig. 11-22. Static MOS RAM cell.

Here the cell is addressed by setting A_R and A_C equal to logic 1. To write into the cell, we have to set W = 1 and MOS M_9 should be turned ON. If the data input is 1, the logic at node D will correspond to 1 turning M_3 ON and the logic level at D will be 0. Thus the logic level at D will become the logic level of the data input. To read data from the cell, we have to set R = 1 and W = 0. This will connect the node $\overline{D}$ to data output line by turning MOS M_{10} ON. Hence the complement of data stored in cell is read at data output line.

11-20. Dynamic MOS RAM

Dynamic RAM cell is of type of a RAM that stores each bit of data in a separate capacitor with an IC. It uses a few MOS transistors than a Static RAM cell. This reduces the silicon chip area and results in low cost. The basic cell is MOS flip-flop with a capacitor. The state of a cell is stored on a capacitor. So if the capacitor leaks charge, the data will be lost. So to avoid this a refreshing circuit is used to refresh the capacitor.

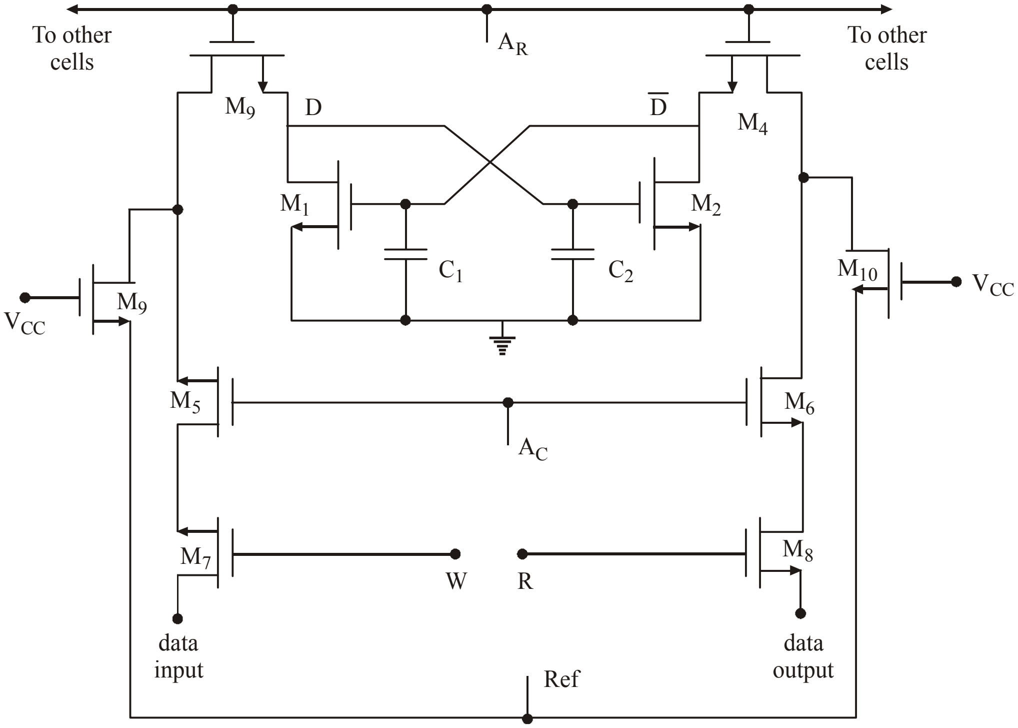


Fig. 11-23. Dynamic MOS RAM.

Fig. 11-23 shows the dynamic MOS RAM cell with read, write and refresh circuit. Here A_R denotes row select, A_C, column select, W, write select and R, Read select respectively. Normally the cell is selected by the row select line A_R and column select line, A_C to logic 1. Let the original state of cell is M_1 ON and M_2 OFF.

For write operation set W = 1 and R = 0. The MOS transistors M_1, M_5, M_3 are ON. If data input is logic 1, the level at junction D is logic 1, which turns on the MOS transistor M_2 by charging capacitor C_2. As M_2 is ON, the capacitor C_1 is discharged and M_1 becomes OFF. So level at junction D represents the input data.

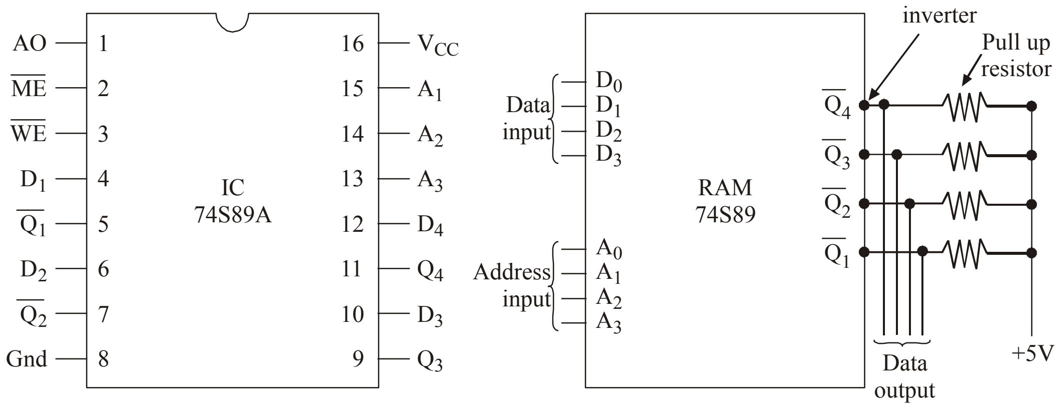
For read operation, we set W = 0 and R = 1. This turns ON the transistors M_8, M_6 and M_4. The logic level at junction $\overline{D}$ is connected to data output. The charge on capacitor C_1 and C_2 does not return for long period. During the refresh time, the logic 1 is applied to node Ref which turns ON the transistors M_{10} and M_9. Let us consider the original state of flip-flop M_1 ON and M_2 OFF. The current will flow in C_1 allowing it to refresh any charge lost due to leakage. Since M_1 is ON, C_2 will not charge as rapidly as C_1. Similarly current will not flow through C_2, when the state of cell is M_2 ON and M_1 OFF.

The data has to be refreshed by every 2 ms time period. Due to this, speed of the dynamic RAMs are lesser than that of the static RAM. The power dissipation of these memories are less as compared to that of static RAMs. DRAMs are famous for their high capacity and low power requirements.

11-21. RAM ICs

We have already discussed about the various types and operations of the RAM in the previous articles. Let us now study about one of the commercially available RAM ICs.

The 74S89A is a 64 bit data storage RAM in IC form. There are 16 addresses of memory locations in this IC. So there are 4 address lines on this package and each memory location is of 4 bit wide. IC's Pin diagram and block diagram are shown in Fig. 11-24. It has data input lines and data output lines. The data output lines are open collector type, so pull up resistor is required at each output data pin.



(*a*) Pin diagram

(*b*) Block diagram

Fig. 11-24. Typical RAM IC.

The memory enable $(\overline{ME})$ input and a read write enable $(\overline{WE})$ are the two control inputs in this chip. $\overline{ME}$ input is used to turn ON or select the chip for reading or writing operation. $\overline{WE}$ input is same as $RD/\overline{WR}$ signal. If the $\overline{WE}$ is at low, then the write operation will take place. During the write period, whatever data is on the input will be stored in the addressed locations and the complement of the stored data is available on the output pins. When the $\overline{ME}$ input is low and $\overline{WE}$ input is high, the chip is selected for read node. During reading, the data will be available at the data output pins in the complement of actual input data. Hence inverters have been attached to the output data pins of 74S89 IC to make the output the same as that stored in the memory location.

By seeing the truth table shown in Table 11-5, we can see the different modes of operations. The access time for IC 74S89A is 50 n sec. There are many RAM ICs like 74S89A available in the market. Normally these ICs have the advantage of inexpensive, small, reliable and able to operate at high speeds.

Table 11-5. Truth table for RAM IC

Control input		s	Operation Output
$\overline{ME}$	$\overline{WE}$		
0	0	Write	Complement input data
0	1	Read	Complement of actual contents of selected location
1	0	Inhibit storage	Complement input data
1	1	Do nothing	All data output pins high

A list of different types of RAM ICs are listed in Table 11-6. Please refer to the datasheet of each IC for its details and operations.

Table 11-6. Different RAM ICs and their characteristics

IC No	Memory Size	Type of RAM	Access time ns	Technology
2114	1024 × 4	Static	200	MOS
74S89A	16 × 4	Static	50	Schottky bipolar
2147H	4096× 4	Static	465	MOS
74S189	16 × 4	Static	40	Schottky bipolar
74S289	16 × 4	Static	35	Schottky bipolar
4464	64 × 4	Dynamic	200	MOS
5101	256 × 4	Static	800	CMOS
21019	1M × 4	Dynamic	300	MOS
2142-2	1024 × 4	Static	200	MOS
2167	16 K × 4	Static	100	MOS
2101A	256 × 4	Static	350	MOS

Example 11-2. *A microprocessor uses RAM chip of 1024 × 1 capacitor.*

(a) How many chips are required to yield the capacity of 1 Kbytes? How will the address and data lines of these chips be connected?

(b) How many chips will be required to provide the capacity of 16 Kbytes? How are they connected?

Solution: Given : Available size of RAM = 1024 × 1 = 1 K bit.

We know that, 1 K byte = 1K × 8 bit

(a) *Number of chips required to obtain 1 K byte RAM*

We have already seen that the given RAM chip is of 1 K bits. Therefore to obtain a RAM of 1 K byte, we need 8 chips of 1 K bits. Here address lines of all the 8 chips are connected together individually (*i.e.*, Address line A_0 of all chips are connected together with other chips and so on). The overall data lines will be equal to product of data lines of individual chip and number of chips. Here we need 8 chips to obtain this capacity. Fig. 11-25 shows the connection scheme for obtaining 16 K byte RAM using 1K × 1 bit devices.

(*b*) *Number of chips required to provide a 16 K byte RAM*

We have already seen in part (*a*) that in order to obtain 1K × 8 bit RAM, we need to connect 8 devices of 1K × 1 bit capacity. Now in order to obtain 16 K byte RAM, we have to connect 16 RAMs with each RAM of 1K × 8 bit.

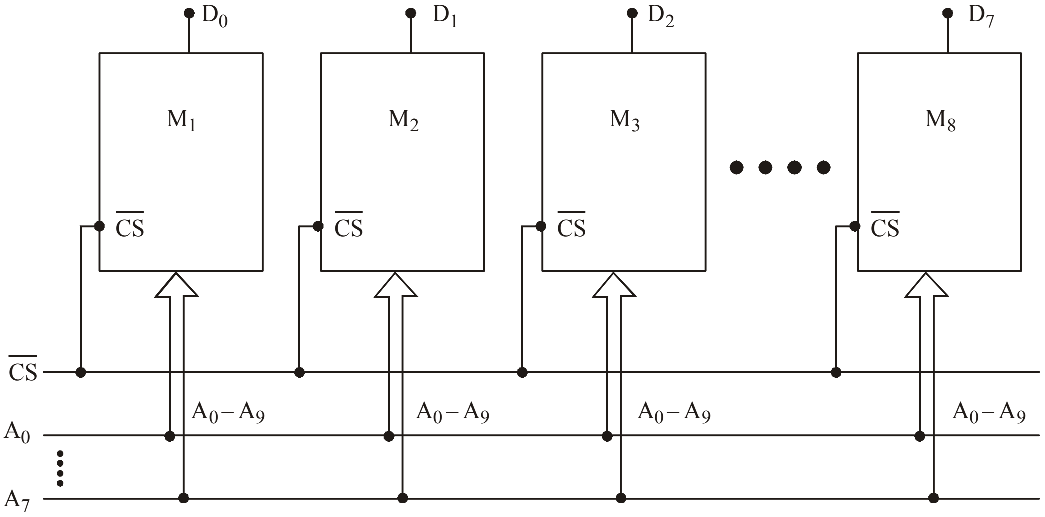


Fig. 11-25

Notice that each device has 1048 memory locations. In order to address each location, we need 10 address lines (i.e., A_0 to A_9). Also we have to select each device separately. So, we need a 4-to-16 decoder. This decoder has 4 inputs which are connected to A_{10}, A_{11}, A_{12} and A_{13} respectively. Fig. 11-26 shows the connection details of this operation.

The total number of devices required are 16 × 8 = 128. Notice the arrangement of RAM devices carefully. The M_1 to M_{16} blocks represent 1K × 8 bit RAM. Here each memory block represents 8 RAM devices whose address lines $A_0 – A_9$ connected to the corresponding lines of the microprocessor address bus. Moreover the $\overline{CS}$ (chip select) control of all the 8 devices is also connected together at one point and that point is connected to the corresponding pin of decoder output.

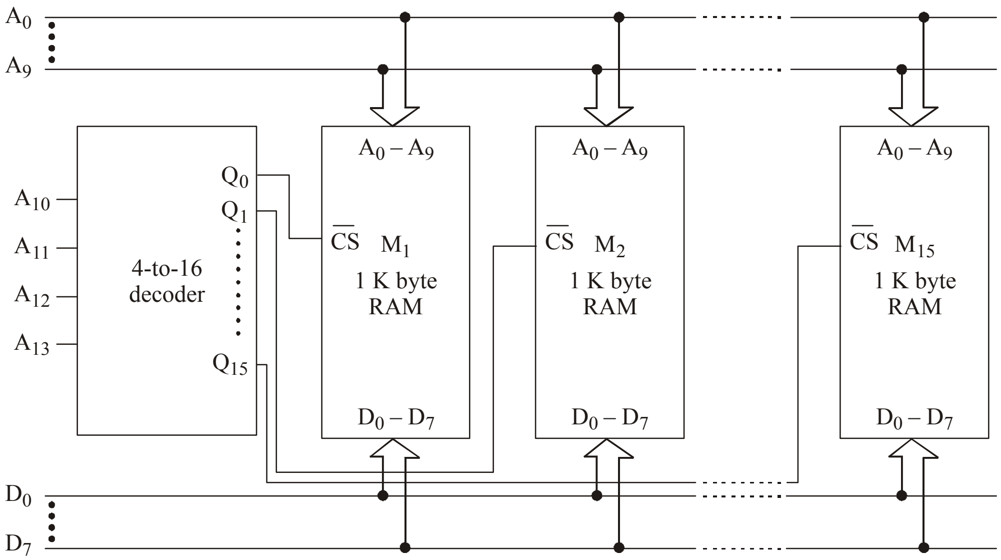


Fig. 11-26.

Example 11-3. *A 8085 microprocessor based system has 4 kB RAM. The address and chip enable pins of the RAM are connected as shown in Fig. 11-27. Calculate the memory map of RAM in HEX form.*

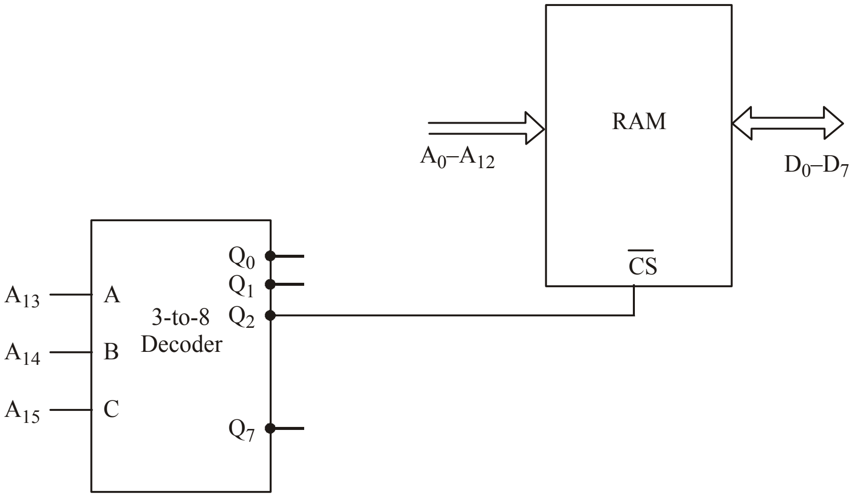


Fig. 11-27.

Solution: As seen from Fig. 11.27, the decoder inputs are connected to A_{13}, A_{14} and A_{15} pins of the microprocessor or address lines. The $\overline{Q_2}$ output of decoder is connected to $\overline{CS}$ of the RAM. So when $A_{15} = 0$, $A_{14} = 1$ and $A_{13} = 0$, the RAM is enabled. The A_{12}–A_0 lines can be any values in the range of 0, ...0 to 1......1. Therefore the address range can be worked out as follows;

Memory map of RAM in HEX form :

Starting address :

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{0\ \ 1\ \ 0\ \ 0}_{\downarrow\ 2} & \underbrace{0\ \ 0\ \ 0\ \ 0}_{\downarrow\ 0} & \underbrace{0\ \ 0\ \ 0\ \ 0}_{\downarrow\ 0} & \underbrace{0\ \ 0\ \ 0\ \ 0}_{\downarrow\ 0} \end{array}$$

Lower Value = 2000H.

Ending address :

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{0\ \ 1\ \ 0\ \ 1}_{\downarrow\ 5} & \underbrace{1\ \ 1\ \ 1\ \ 1}_{\downarrow\ F} & \underbrace{1\ \ 1\ \ 1\ \ 1}_{\downarrow\ F} & \underbrace{1\ \ 1\ \ 1\ \ 1}_{\downarrow\ F} \end{array}$$

Upper Value = 5FFFH

So the address range is from 2000H to 5FFFH. **Ans.**

Example 11-4. *Fig. 11-28 shows the memory circuit of a typical microprocessor base system.*

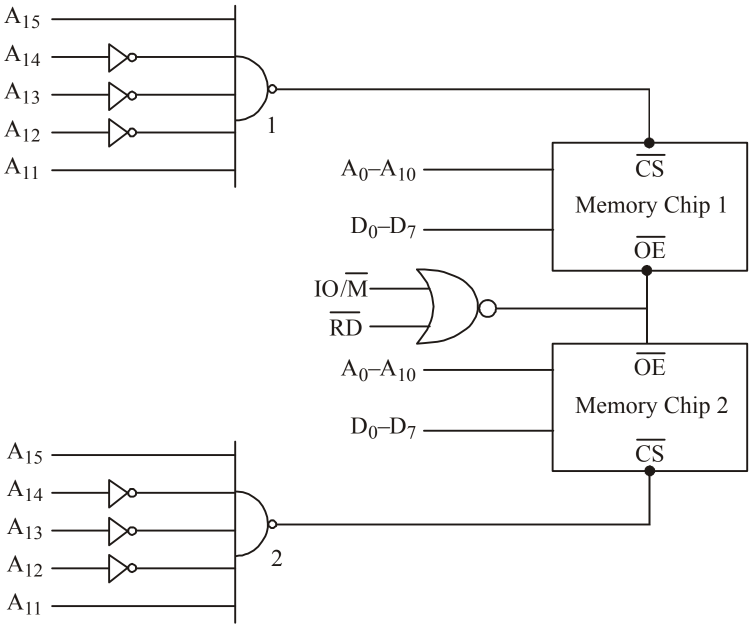


Fig. 11-28.

(*a*) *What is the total size of the memory in the circuit?*

(*b*) *What are the beginning and ending address of the memory in chip 1?*

(*c*) *What are the beginning and ending address of the memory in chip 2?*

(*d*) *Are the memory chips in the circuit, ROM or RAM?*

(*e*) *How will you replace the two NAND gates in the circuit with one 3-to-8 decoder without changing the memory size or memory address? Assume that the decoder has one active high enable E_1 and low enable E_2.* (GATE, 1992, E & C)

Solution : Given : The memory circuit of microprocessor shown in Figure 11.28.

(*a*) *Total size of the memory*

As seen from the memory circuit shown in Figure 11.28, we find that it has two memory chips. Each memory chip has A_0....A_{10} (*i.e.*, 11 memory address lines).

∴ Total size of the memory = 2 × 2^{11} = 2^{12} = 4K. **Ans.**

(*b*) *Chip 1 : Beginning and ending address*

The beginning address will correspond to the values of A_0....A_{10} and A_{11}, A_{12}, A_{13}, A_{14} and A_{15}. The values of A_0....A_{10} has to be 0....0 whereas the values of A_{11} to A_{15} as seen from the circuit has to be,

$$A_{11} = 1, A_{12} = 0, A_{13} = 0, A_{14} = 0, A_{15} = 1$$

These values of A_{11} through A_{15} will generate a logic LOW level at the output of NAND gate 1. This in turn will enable the memory chip 1. So we can work out the beginning address in hexadecimal as below.

$$\begin{matrix} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{1\quad 0\quad 0\quad 0}_{\downarrow} & \underbrace{1\quad 0\quad 0\quad 0}_{\downarrow} & \underbrace{0\quad 0\quad 0\quad 0}_{\downarrow} & \underbrace{0\quad 0\quad 0\quad 0}_{\downarrow} \\ 8 & 8 & 0 & 0 \end{matrix}$$

∴ The chip 1 beginning address = 8800H **(Ans.)**

The ending address may be computed from the fact that there is no change in values of A_{11}, A_{12}, A_{13}, A_{14}, and A_{15}. But A_0 through A_{10} will all be 1s instead of Os. Thus the ending address will be,

$$\begin{matrix} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{1\quad 0\quad 0\quad 0}_{\downarrow} & \underbrace{1\quad 1\quad 1\quad 1}_{\downarrow} & \underbrace{1\quad 1\quad 1\quad 1}_{\downarrow} & \underbrace{1\quad 1\quad 1\quad 1}_{\downarrow} \\ 8 & F & F & F \end{matrix}$$

∴ Ending address of chip 1 = 8FFFH **(Ans.)**

(*b*) *Chip 2 : Beginning and ending address*

Following the same approach as discussed earlier for chip 1, the beginning address of chip 2 in hexadecimal may be obtained as shown below.

$$\begin{matrix} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{1\quad 0\quad 1\quad 1}_{\downarrow} & \underbrace{1\quad 0\quad 0\quad 0}_{\downarrow} & \underbrace{0\quad 0\quad 0\quad 0}_{\downarrow} & \underbrace{0\quad 0\quad 0\quad 0}_{\downarrow} \\ B & 8 & 0 & 0 \end{matrix}$$

∴ The beginning address of chip 2 = B800H. **(Ans.)**

The ending address of chip 2 can also be worked out by proceeding in the same way as discussed earlier for chip 1, *i.e.*,

$$\begin{matrix} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{1\quad 0\quad 1\quad 1}_{\downarrow} & \underbrace{1\quad 1\quad 1\quad 1}_{\downarrow} & \underbrace{1\quad 1\quad 1\quad 1}_{\downarrow} & \underbrace{1\quad 1\quad 1\quad 1}_{\downarrow} \\ B & F & F & F \end{matrix}$$

∴ Thus the ending address of chip 2 = BFFFH. **(Ans.)**

(*c*) As seen from the Fig. 11-28, there is only read signal involved in the circuit. So we can conclude that memory chips in the circuit will be of ROM type.

(*d*) To replace the two NAND gates in the circuit by one 3-to-8 decoder without changing the size could be done in the following manner.

Active HIGH enable E_1
Active LOW enable E_2.

It may be seen that A_{15} remains HIGH and A_{14} remains LOW in both chips. So according to this type, only A_{13}, A_{12} and A_{11} will change as shown in Fig. 11-29.

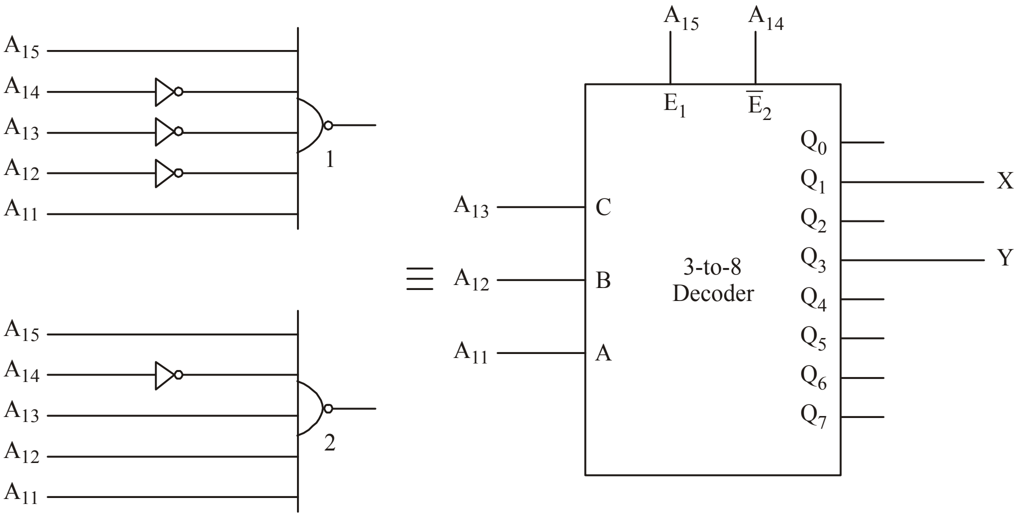


Fig. 11-29.

Example 11-5. *Consider the following decoder circuit shown in Fig. 11-30 for providing chip select signals to an EPROM, a RAM and an I/O chip with four addressable registers from a 3-to-8 decoder.*

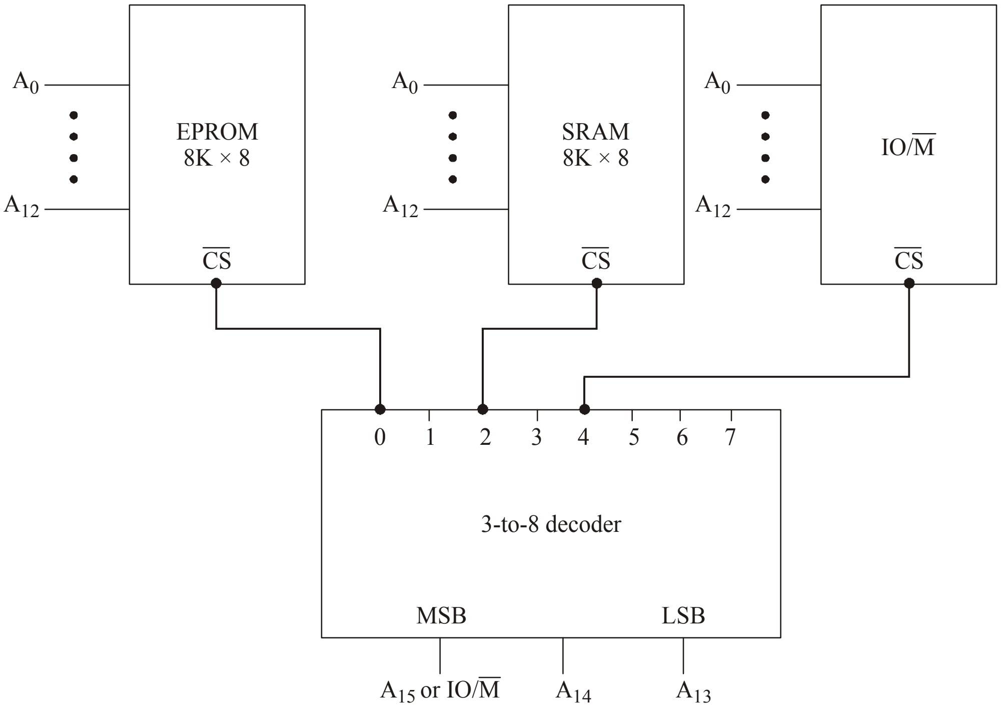


Fig. 11-30.

(*a*) *Specify all the memory address ranges to which the EPROM will respond.*

(*b*) *Specify all the memory address ranges to which the RAM will respond.*

(*c*) *Specify all the I/O address ranges to which the I/O chip will respond.*

Solution : (*a*) The output of the 3-to-8 decoder enables the 8K × 8 memory EPROM by active LOW $\overline{CS}$.

We know that 8K bytes = $2^{13} \times 8$ bits.

So the all starting address will be filled with zeros.

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} \end{array}$$

∴ So the starting address = 0000H.

Now in the ending address, the 13 address lines will be filled with 1s. Thus the ending address will be like this :

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{0\quad 0\quad 0\quad 1}_{\substack{\downarrow \\ 1}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow \\ F}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow \\ F}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow \\ F}} \end{array}$$

∴ The ending address = 1FFFH.

Hence the address range of 8K × 8 EPROM will be from 0000H to IFFFH. **(Ans.)**

(*b*) SRAM will be selected when A_{15}, A_{14} and A_{13} are 0, 1 and 0 respectively. So for this combination, the beginning address will be like,

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{0\quad 1\quad 0\quad 0}_{\substack{\downarrow \\ 4}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} \end{array}$$

∴ The starting address = 4000H.

The ending address will be like,

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{0\quad 1\quad 0\quad 0}_{\substack{\downarrow \\ 4}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow \\ F}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow \\ F}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow \\ F}} \end{array}$$

∴ The ending address = 4FFFH.

So the address will be in the range of 0000H to 4FFFH. **(Ans.)**

(*c*) The $IO/\overline{M}$ is selected when $IO/\overline{M}$ or A_{15} is HIGH and A_{14}, A_{13} are LOW. So for this combination, the beginning address will be like,

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{1\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 8}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} & \underbrace{0\quad 0\quad 0\quad 0}_{\substack{\downarrow \\ 0}} \end{array}$$

∴ The starting address = 8000H.

The ending address will be like,

$$\begin{array}{cccc} A_{15}\ A_{14}\ A_{13}\ A_{12} & A_{11}\ A_{10}\ A_9\ A_8 & A_7\ A_6\ A_5\ A_4 & A_3\ A_2\ A_1\ A_0 \\ \underbrace{1\quad 0\quad 0\quad 1}_{\substack{\downarrow\\ 9}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow\\ F}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow\\ F}} & \underbrace{1\quad 1\quad 1\quad 1}_{\substack{\downarrow\\ F}} \end{array}$$

∴ The ending address = 9FFFH.

The address ranges of $IO/\overline{M}$ chip will be from 0000H to 9FFFH. **(Ans.)**

Example 11-6. *A CPU has a parallel address bus, a parallel data bus, a* $\overline{RD}$ *and a* $\overline{WR}$ *signal. Two ROMs of size 4K words each and two RAMs of size 16K and 8K words, respectively are to be connected to the CPU. The memories are to be connected that they fill the address space of the CPU as per the memory map shown in Fig. 11-31. Assuming that chip select signals are active LOW.*

(*a*) *What is the number of lines in the address bus of the CPU?*

(*b*) *Determine the values of address X, Y, Z and W.*

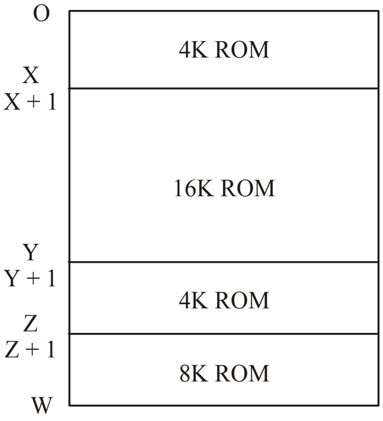


Fig. 11-31

Solution: (*a*) For the two 4K ROM, we can have 12 address lines, the 8K and 16K will be having two more address lines. Totally 14 lines and two more address lines for decoding operations will be there. Hence 16 address lines are available.

(*b*) Value of address at X,

$$\begin{aligned} X &= \text{4K ROM ending location} \\ &= 4096 - 1 \text{ (for ending of the block)} \\ &= 4095 \text{ (in decimal form)} \\ &= \text{0FFF H } \textbf{(Ans.)} \end{aligned}$$

Value of address at Y,

$$\begin{aligned} Y &= X + 16\text{ K} \\ &= 4 + 16\text{K} = 20\text{K} \\ &= 20480 - 1 = 20479 \\ &= \text{4FFFH. } \textbf{(Ans.)} \end{aligned}$$

Value of address at Z,

$$\begin{aligned} Z &= X + Y + 4K = 4 + 16 + 4 = 24K \\ &= 24576 - 1 = 24575 \\ &= 5FFFH. \textbf{ (Ans.)} \end{aligned}$$

Value of address at W,

$$\begin{aligned} W &= X + Y + Z + 8K \\ &= 4K + 16K + 4K + 8K = 32K \\ &= 32768 - 1 = 32767 \\ &= 7FFFH. \textbf{ (Ans.)} \end{aligned}$$

∴ The memory diagram will be shown in Fig. 11-32.

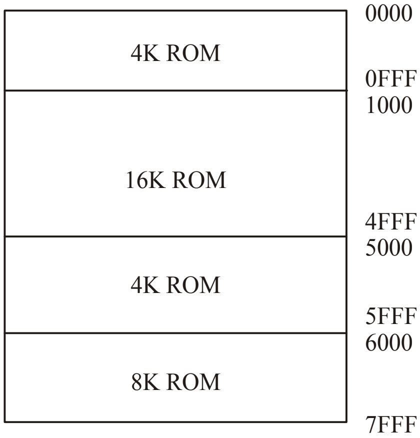


Fig. 11-32.

11-22. Cache Memory

As a matter of fact, most of the computer processing tasks require very large memory in order to reduce the processing time. So to increase the processing speed, a memory is used between CPU and main memory. This memory is known as Cache (pronounced as Cash) memory. In this Cache memory, frequently accessed data are stored for rapid access. Normally if we want to access the data from the main memory, it will take longer time to fetch the data. To avoid this time delay, Cache memory is being used in most of the computer applications. The main difference between main memory and Cache memory is that the Cache memory is manufactured with extremely fast static RAM and normally it cannot be accessible to the programmers.

Fig. 11-33 shows the position of the Cache memory between CPU and the main memory. In the absence of Cache memory, data is transferred serially and processor takes its time to compute any data. When the Cache memory is used, some information in main memory is transferred to Cache memory. When the CPU has processed the information in main memory, Cache memory contents are transferred at high speed to main memory. During the system operation, recently used data and instructions are stored in Cache memory. Fig. 11-34 shows how Cache memory uses the main memories data.

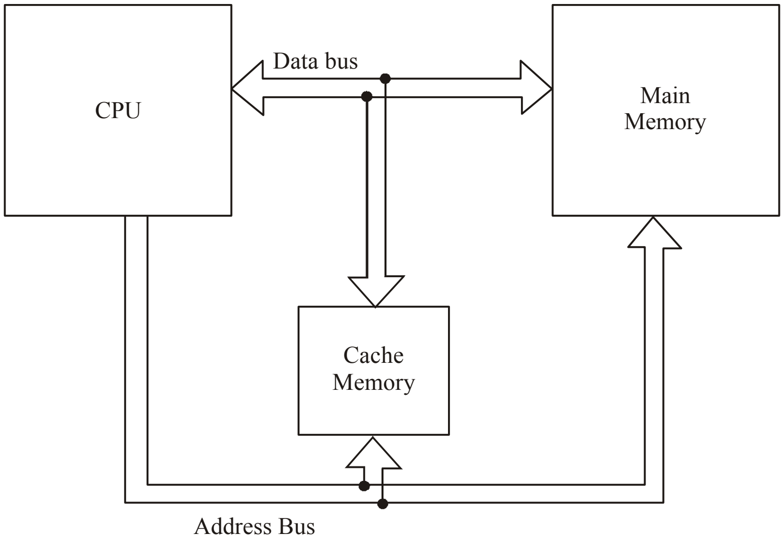
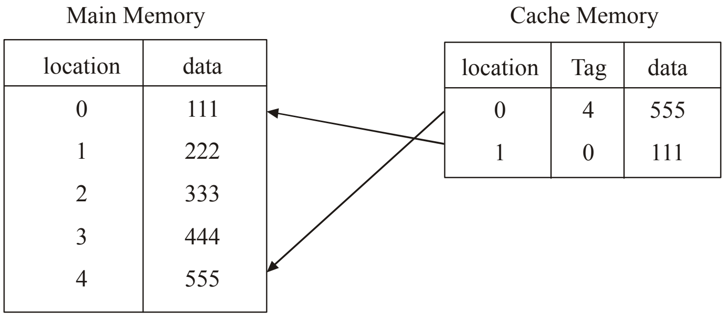


Fig. 11-33. Location of Cache memory.



Main Memory

location	data
0	111
1	222
2	333
3	444
4	555

Cache Memory

location	Tag	data
0	4	555
1	0	111

Fig. 11-34. Cache memory using main memories data.

The CPU directly manages the Cache memory size. Normally the sizes will vary from 16 K Bytes to 256 K Bytes. Basically Cache memory is used to minimize the access time to access the data in the main memory and to increase the speed of CPU.

11-23. Content Addressable Memory (CAM)

The Content Addressable Memory (CAM) is a special purpose random access memory that performs three basic operations such as read, write and associate. It is used in very high speed searching applications. Each location of this memory is addressed by one associate code referred as key. If this Code matches with the Content in a location, then only data can be read or write. The key used may either consist of entire data word or some specific bits of data word with other bits masked.

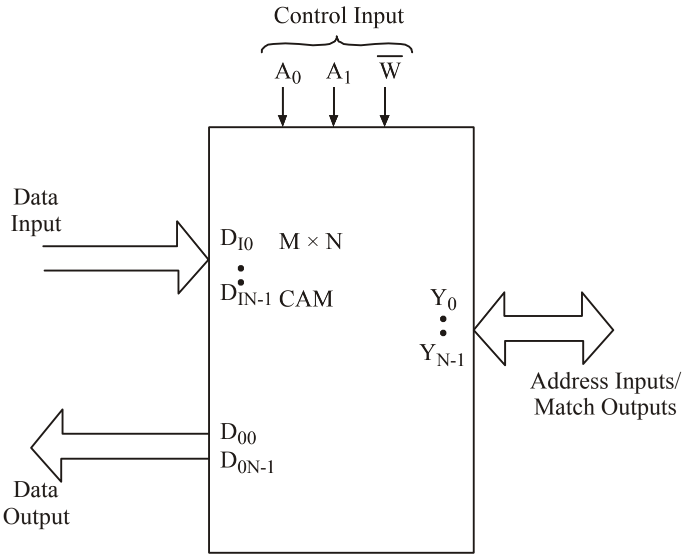


Fig. 11-35. Content Addressable Memory

Fig. 11-35 shows a block diagram of CAM. As seen from the figure, it has N data input lines and N data output lines and size of M words of N bits. The input data lines D_{I0} through D_{IN-1} are used to input data into the memory as well as key word if it is in associate operation. Data output lines are from D_{00} to D_{0N-1}. These are used to read data. Read and write operation of CAM is same as conventional memory. During write operation input data also appear on output data lines. Each memory location is made up of flip-flop and some logic gates as shown in Fig. 11-36.

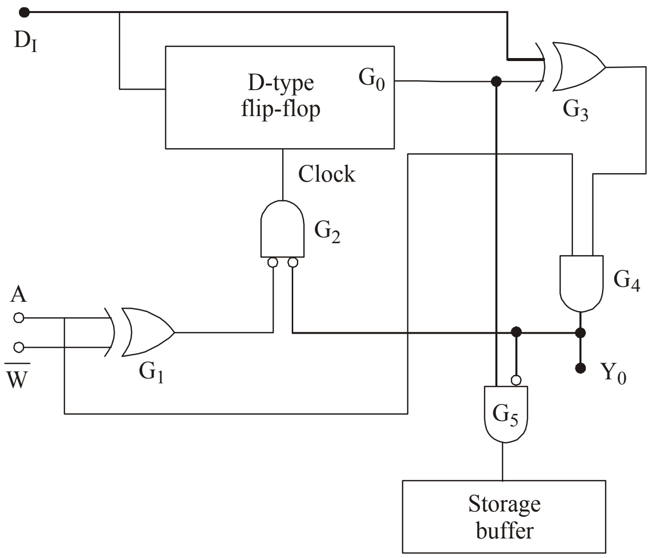


Fig. 11-36. Content Addressable Memory.

The Y lines between Y_0 and Y_{N-1} are bidirectional. These lines select a storage location in read or write operation. Here Y_0 represents the address in memory location 0 and Y_{N-1} is the last address location. For an association operation, these lines are used to match output for memory location. So if the keyword matches with the content of memory location, the line corresponding to that address will be HIGH. For example, if the content of memory location 1 matches with keyword, then Y_1 will be HIGH. The control inputs are required to select the required operation.

The word length and/or the word size can be expanded in similar manner as used for conventional memory. A_0 and A_1 are the associate inputs. During write operation, when A = 0, $\overline{W} = 0,$ output of Gate G_1 is HIGH and output of Gate G_2 is LOW. It means that no clock pulse is applied at the clock input of D flip-flop and input data will not appear at Q_0. In other words, data will not write in the storage buffer. Therefore when A = 0, it represents the mismatch of the location key. When A = 1 and $\overline{W} = 0,$ the output of Gate G_1 is LOW and output of G_2 is HIGH. This enables the clock input. Now Q_0 gives the status of input data. Thus A = 1 represents the match key.

A CAM has the ability to search out in the stored data on the basis of its content. Today CAM is used in many applications. They are :

1. Data base engines
2. Artificial Neural Network
3. Data compression hardware
4. Intrusion prevention system
5. CPU Cache Controller.

SUMMARY

In this chapter you have learned that

1. Semiconductor memory is used in digital systems and electronic appliances. That might have the data to execute a certain operation.
2. Data is stored in binary form (either 0 or 1).
3. The size of the memory is measured by the number of storage locations available in the system. Normally it is referred by bytes.
4. To write data into the memory, the address to be accessed will be in address input, the data value to be stored into the data inputs, and the Control Signals $(\overline{RW} \text{ and } \overline{CS})$ are used to store the data. To read a data from the memory, the address is sent to the input, then Control Signals will manipulate and the data will appear in the data output lines.
5. Memory capacity can be extended by increasing the word capacity by increasing the word size.
6. Access time for any location in sequential access memories is different.
7. Time required to access any location within RAM (Random Access Memory) is same.
8. Static RAM is faster than DRAM.
9. Data stored in dynamic RAM needs to be refreshed after a specified time because of charge leakage from capacitor.
10. RAM is a volatile memory and ROM is a non-volatile memory.
11. Depending on the type of EPROM (*i.e.*, whether EPROM or EEPROM), the data inside the memory can be erased by ultra-violet rays or electrical pulses.
12. Flash memory is the fastest version of EEPROM.
13. Static RAM uses either bipolar transistors or MOS transistors whereas Dynamic RAM uses only MOS transistors.
14. A typical dynamic RAM cell can be implemented by using one MOS transistor and one capacitor.
15. Charge coupled device stores data as charge on MOS capacitors and transfer of charge through various channels takes place in sequential manner.
16. The data is transferred to PROM by using a PROM programmer.

GLOSSARY

Memory Cell : A part of semiconductor memory which can store a single bit (1 or 0).

Memory Word : A group of memory cells that contain the instruction or data to execute. This depends on the requirements. Normally it should be in the range of 4 bit, 8 bit, 16 bit, 32 bit or 64 bit.

nibble : A collection of 4 bits.

byte : A collection of 8 bits.

Memory Capacity : It is a method of telling how many bytes of data stored in the memory.

Address : A unique number, that stores the memory in a system.

Address bus : Parallel lines/wires used for accessing a memory location.

Data bus : Parallel lines/wires used to read or write data into a memory location.

Read Operation : A process of locating a binary word in the memory, fetching it and transferring into the peripheral device.

Write Operation : The process of storing data in the memory.

Access time : It is a method to find the speed of a memory device.

RAM : Random Access Memory or Read Write Memory. We can read and write data into the memory.

ROM : Read Only Memory. A data cannot be written into it normally. Although, there are some special mechanisms which are used to write data into the memory.

Static Memory : A memory in which data does not change with time.

Dynamic Memory : A memory in which data needs to be refreshed periodically.

Volatile Memory : A memory in which data is lost when power is switched off.

Non-Volatile Memory: A memory in which data is not lost even after the power is switched off.

EPROM : It is electrically programmable read only memory. It can be burnt by the user. Once burnt, can be reused by erasing the previous contents by using ultra-violet rays.

EEPROM : It is same as EPROM, but erasing will be done electrically.

Flash Memory : It is a non-volatile memory and it is better than EPROM and EEPROM.

Shift Register Memory : It is a sequential semiconductor memory in which data bit moves from its position only when new data is entered.

CCD Memory : It is a charge coupled device memory with large bit packing density.

CAM (Content Addressable Memory) : It is a read/write memory which can be accessed by its content.

Main Memory Unit : It is also called as prime memory. It stores the instructions and data and that will be executed by CPU.

Auxiliary Memory Unit : It is called as secondary memory or mass storage system. It stores huge amount of data, but works slower than main memory.

DESCRIPTIVE QUESTIONS

1. Explain about memory system in a digital computer.
2. What are the different buses available in digital systems? Explain them in detail.
3. How the memories are classified? Explain them in detail.

4. Distinguish between ROM, PROM and MROM.

5. What are the differences between EPROM, EEPROM and flash memory? Where are they used?

6. What is the main difference between ROM and RAM? Explain the types of these devices.

7. Write short notes on the following :

(*a*) Volatile memory (*b*) Memory capacity

(*c*) Read cycle operation (*d*) Write cycle operation

(*e*) byte (*f*) word

8. What are the different types of methods used to expand the memory size?

9. What are the sequential memories? Explain the working of shift register memory.

10. What is the charge coupled device? How does the CCD work?

11. What is ROM? Write the truth table of 2-to-4 decoder with output polarity control and built with discrete gate and with an 8 × 4 ROM.

(*A.M.I.E., E & T 1999*)

12. Draw a ROM array and explain its working.

13. Compare the SRAMs and DRAMs.

14. Describe any of the RAM ICs.

15. Draw a BJT RAM cell and explain how data can be stored and retrieved from it. On what factor the switching speed and a MOSFET depends?

(*A.M.I.E., E & T 2002*)

16. Draw the diode matrix forming a ROM unit? Show how it stores data.

(*Gauhati University, 2007*)

17. Explain how reading and writing of information take place in RAM.

(*Gauhati University, 2006*)

18. Discuss in brief semiconductor memory organization and its operation.

(*Nagpur University, 2008*)

19. Draw and explain the structure of a RAM cell.

(*Mahatma Gandhi University, Jan. 2007*)

20. Explain with figures the organization of ROM.

(*Mahatma Gandhi University, Jan. 2007*)

21. Differentiate between RAM and ROM.

(*Mahatma Gandhi University, Dec. 2007*)

22. Draw the ROM and EPROM architecture and explain their working principles. Describe the different types of EPROM.

(*Mahatma Gandhi University, Dec. 2007*)

23. Explain the principle of various types of EPROM.

(*Mahatma Gandhi University, May/Jun. 2006*)

24. Bring out the difference between static RAM and Dynamic RAM.

(*Mahatma Gandhi University, Nov. 2005*)

25. State and explain the differences among ROM, PROM, RAM, SRAM and DRAM.

(*PTU, Dec 2008*)

26. What is the difference between static and dynamic RAM?

(*PTU, May 2009*)

27. Define noise margin. What is its importance?

(*PTU, May 2009*)

28. What are the various types of ROMs? Discuss their relative advantages and disadvantages.

(*PTU, Dec. 2009*)

29. Draw the basic dynamic memory cell.

(*Anna University, Nov. /Dec. 2005*)

30. Which memory is called volatile? Why?

(*Anna University, Nov. /Dec. 2005*)

31. Describe the RAM organization.

(*Anna University, Nov. /Dec. 2005*)

32. A bipolar RAM chip is arranged as 16 words. How many bits are stored in the chip?

(*Anna University, Nov. /Dec. 2005*)

33. Write a note on:

(*i*) MOSFET RAM cell

(*ii*) Dynamic RAM cell

(*Anna University, Nov. /Dec. 2005*)

34. What is meant by static and dynamic memories?

(*Anna University, May /Jun. 2006*)

35. How is individual location in a EEPROM programmed or erased?

(*Anna University, May /Jun. 2006*)

36. How can one make 64 × 8 ROM using four 32 × 4 ROMs? Draw such a circuit and explain.

(*Anna University, May /Jun. 2006*)

37. Draw a dynamic RAM cell and explain its operation. Compare its simplicity with that of NMOS static RAM cell, by way of diagram and operation.

(*Anna University, May /Jun. 2006*)

38. Explain static memory.

(*Anna University, Nov. /Dec. 2006*)

39. What is RAM.

(*Anna University, Nov. /Dec. 2006*)

40. Mention the two types of erasable PROM.

(*Anna University, Nov. /Dec. 2006*)

41. Illustrate the concept of 16 × 8 bit ROM arrange with diagram.

(*Anna University, Nov. /Dec. 2006*)

42. Describe the typical ROM internal organization with necessary diagram.

(*Anna University, Nov. /Dec. 2006*)

43. Explain the basic structure of a 256 × 4 static RAM, with neat diagram.

(*Anna University, Nov. /Dec. 2006*)

44. Elaborate the single fused PROM cell with clear sketch

(*Anna University, Nov. /Dec. 2006*)

45. What is static memory?

(*Anna University, May /Jun. 2006*)

46. Whether ROM is classified as a non volatile memory device? Why?

(*Anna University, May /Jun. 2006*)

47. Write the advantages of EPROM over a PROM.

(*Anna University, May /Jun. 2006*)

48. Categories RAM and ROM and explain in detail.

(*Anna University, May /Jun. 2006*)

49. Explain the following terms:

(*i*) Dynamic Memory
(*ii*) Volatile storage
(*iii*) Field programmable
(*iv*) Mask programmable

(*Anna University, May /Jun. 2006*)

50. Explain the basic structure of 256 × 4 static RAM with meat sketch.

(*Anna University, May /Jun. 2006*)

51. Draw the logic diagram of a memory cell.

(*Anna University, Nov. /Dec. 2007*)

52. Write short notes on:

(*i*) Ram

(*ii*) Types of ROM's

(*Anna University, Nov. /Dec. 2007*)

53. Design a 16 bit ROM array and explain the operation.

(*Anna University, Apr. /May 2008*)

54. Write short notes on:
(*i*) RAM
(*ii*) ROM
(*iii*) EPROM (*Nagpur University, 2004, 2008*)

55. Write down the classification of memories.

(*GBTU/MTU, 2009-10*)

56. Briefly describe the ROM family.

(*GBTU/MTU, 2005-06*)

57. What do you mean by DRAM? Why they are called dynamic RAM?

(*GBTU/MTU, 2005-06*)

58. Write the short notes on the following :

(*i*) Sequential and Random Access Memories

(*ii*) Semi-conductor memories

(*GBTU/MTU, 2004-05*)

59. Write a short note on ROM.

(*GBTU/MTU, 2005*)

60. Write a short note on SRAM call. Also show the various configuration of the static RAM cell.

(*GBTU/MTU, 2005*)

61. Describe EEPROM in short.

(*Nagpur University, 2004*)

62. Write short notes on the following :

(*i*) Formation of memory banks

(*ii*) Static and dynamic memories

(*GBTU/MTU, 2006-07*)

63. What is computer memory ? Give classification of memory system. Discuss EEPROM.

(*RGTU., Dec. 2010*)

64. Explain the construction and organization of the following memorie :

(*i*) SRAM

(*ii*) DRAM

(*iii*) ROM

(*v*) PROM

(*RGTU., Dec. 2010*)

65. Draw the logic diagram of a 4 × 4 RAM and describe the operation.

(*RGTU., June 2011*)

66. Discuss the principle of EPROM and EEPROM.

(*RGTU., Dec. 2009*)

67. Explain the organisation and construction of RAM BUS ROM.

(*RGTU., June 2009*)

68. With the help of logic diagram and circuit diagram explain static RAM cell.

(*RGTU., June 2009*)

69. Explain the working of Bipolar PROM array with fusible links.

(*RGTU., June 2010*)

TUTORIAL PROBLEMS

1. Consider the following connection scheme to memory in Fig. 11-37

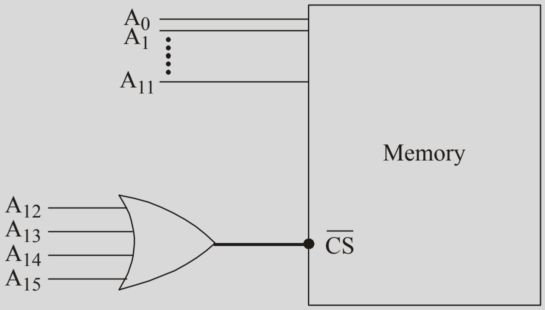


Fig. 11-37

Calculate the associate range of address for the memory.

2. Find the number of address lines for the following memory devices :

(*a*) 8192 × 4 (*b*) 4096 × 8 (*c*) 2048 × 8

3. Determine the number of bits in address bus of a processor which has a memory space of 256 locations. Which location is addressed if the address in hexadecimal is 6F?

4. A processor has 10 bits of address and 8 bits of data bus. If there are 16K × 8 bit memory chips available, how many chips are needed and how will they be connected?

5. A processor has 16 bits of address and 16 bits of data lines. Memory chips are available each of 32K × 4 bit. How many chips are needed and how to connect them in system to provide full capacity primary memory?

6. A static ROM has the following timing parameters : $t_W = 40$ ns; $t_{DH} = 20$ ns, $t_{ACC} = 100$ ns, $t_{RC} = 100$ ns, $t_{AC} = 20$ ns; $t_{CO} = 70$ ns, $t_{OD} = 30$ ns, $t_{WC} = 100$ ns, $t_{DS} = 10$ ns.

(*a*) How long data will remain after $\overline{\text{CS}}$ returns back to HIGH?

(*b*) What is the minimum time for which the data have to remain valid for a reliable write operation to occur?

(*c*) How long should $\overline{\text{CS}}$ and $\overline{\text{RW}}$ be kept HIGH after the new address stabilize during write cycle?

7. Among Read-only-memory and Read/Write memory, which is preferred for storing information? Which is not already stored for a memory system and why?

8. IC 2764 is 65536 bit EPROM organised as 8192 words of 8 bits each. Determine the number of address lines and data lines.

9. What is the capacity of a RAM which has 16 address inputs, 4 data inputs and 4 data outputs? Design a 512K × 8 bit memory using this RAM.

10. Design a 16 × 8 CAM using 8 × 2 CAM.

11. A memory system contains a cache memory, a main memory and a virtual memory. The access time of the Cache is 5 ns and it has an 80% hit rate. The access time of the main memory is 100 ns and it has a 99.5% hit rate. The access time of virtual memory is 10 ns. What is the average time of hierarchy?

(*IES 2005, E & T Engg.*)

12. A system has 48 bit virtual addresses, 36 bit physical addresses and 128 mB of main memory. If the system uses 4096 byte pages, how many virtual and physical pages can the address space support? How many page frames of main memory are there?

(*IES 2005, E & T Engg.*)

MULTIPLE CHOICE QUESTIONS

1. Match the following :

(*a*) Primary Memory	1. Non-volatile memory
(*b*) Secondary Memory	2. Volatile memory
(*c*) RAM	3. On board memory
(*d*) ROM	4. Mass storage memory
(*e*) Memory Cell	5. One word
	6. Storage device of 1 bit

2. A memory device is designed 16K × 4. How many memory locations are contained in the chip?
(*a*) 512 (*b*) 1024 (*c*) 16384 (*d*) 4096

3. How many bits are stored in each memory location in a 16K × 4 memory device?
(*a*) 16 (*b*) 4 (*c*) 8 (*d*) 24

4. How many total bits are stored in 16K × 4 memory device?
(*a*) 4096 (*b*) 1024 (*c*) 65536 (*d*) 10368

5. A Read/Write memory chip has a capacity of 64K bytes. Assuming separate data and address lines and availability of chip enable signal, what is number of pins required in the IC chip?
(*UPSC Civil Services 2002*)
(*a*) 28 (*b*) 26 (*c*) 24 (*d*) 22

6. Suppose 64 KB ROM ICs are available in abundance, 1 MB ROM can be obtained from,
(*a*) 16 ICs in a row (*b*) 16 ICs in a Column
(*c*) 8 ICs in a column and 2 ICs in a row (*d*) None of the above
(*IES-2006, Elect. Engg*)

7. Consider the following statements :
Data that are stored at a given address in a random access memory are lost,
(1) when power goes off.
(2) when the data are read from the address.
(3) when new data written at the address.
(4) because it is non-volatile memory.
(*a*) 1 and 2 (*b*) 1, 2 and 4 (*c*) 2 and 3 (*d*) 1 and 3
(*IES - 2006, Elect. Engg.*)

8. Four memory chips of 16 × 4 size have their address buses connected together. This system will be of size
(*a*) 64 × 4 (*b*) 16 × 6 (*c*) 32 × 8 (*d*) 256 × 1
(*UPSC Civil Services 2001*)

9. The number of memory chips of size 1K × 4 bits required to build a memory bank of size 16 × 8 bits is
(*a*) 64 (*b*) 32 (*c*) 16 (*d*) 8
(*A.M.I.E., 2003*)

10. Which one of the following is an example of non-volatile memory?
(*a*) Static RAM (*b*) Dynamic RAM
(*c*) ROM (*d*) Cache memory
(*UPSC Engg. Services, 1998*)

11. EPROM memory content can be erased by exposing it to
(*a*) UV light (*b*) Intense heat radiation
(*c*) IR rays (*d*) microwaves
(*A.M.I.E., 2002*)

12. If a RAM has 34 bits in its MAR (address register) and 16 bits in its MDR (data register), then its capacity will be
(*a*) 32 GB (*b*) 16 GB (*c*) 32 MB (*d*) 16 MB
(*UPSC Civil Services, 2000*)

13. Memory chips of four different sizes as below are available

1. 32K × 4   2. 32K × 16   3. 8K × 8   4. 16K × 4

All the memory chips as mentioned in the above list are Read/Write memory. What minimal combination of chips or chip alone can map full address space of 8085 microprocessor?

(*a*) 1 and 2   (*b*) 1 only   (*c*) 2 only   (*d*) 4 only

(*IES-2005, Elect. Engg.*)

14. Consider the following statements. In memories,

1. ROMs are used for temporary program and data storage.
2. Dynamic RAM is less expensive than SRAM.
3. MASK ROM is used in high volume microprocessor based system.

Which of the following answers given is/are correct?

(*a*) 1 only   (*b*) 1 and 2   (*c*) 2 and 3   (*d*) 1, 2 and 3

(*IES-2005, Elect. Engg.*)

15. A single ROM is used to design a combinational circuit described by a truth table. What is the number of address lines in the ROM?

(*a*) Number of input variables in the truth table.

(*b*) Number of output variables in the truth table.

(*c*) Number of input plus output variables in the truth table.

(*d*) Number of lines in the truth table?

(*IES 2006* (*E & T Engg.*))

16. Which one of the following statements is correct?

(*a*) RAM is a non-volatile memory whereas ROM is a volatile memory.

(*b*) RAM is a volatile memory whereas a ROM is a non-volatile memory.

(*c*) Both RAM and ROM are volatile memories but in ROM data is not lost when power is switched off.

(*d*) Both RAM and ROM are non-volatile memories but in RAM data is lost when power is switched off.

(*UPSC Civil Services 2000*)

17. Cycle time of Random access memory is the time required between successive

(*a*) access to memory   (*b*) clocks

(*c*) delete operation   (*d*) Read/write operation

(*A.M.I.E., 2000*)

18. The number of MOS transistors required to implement a typical dynamic RAM cell is

(*a*) 6   (*b*) 5   (*c*) 1   (*d*) 2

(*UPSC Civil Services 2002*)

19. The larger the RAM of a computer, the faster is its speed, since it eliminates

(*a*) need for ROM   (*b*) need for external memory

(*c*) frequent disk I/Os   (*d*) need for a data-wind path

(*UPSC Civil Services, 1998*)

20. Static RAM is preferred over dynamic RAM when the requirement is of

(*a*) slow speed of operation   (*b*) large storage capacity

(*c*) lower access time   (*d*) lower power consumption

(*A.M.I.E. 2001*)

21. Match List I (type of memory) with List II (property) and select the correct answer.

	List I		**List II**
A.	Static RAM (SRAM)	1.	Non-volatile memory
B.	Read-only memory (ROM)	2.	Memory refresh is required
C.	Dynamic RAM (DRAM)	3.	Low density memory as memory cell may have 4 or more transistors.

(*A.M.I.E., 2002*)

22. Match List I (Type of Memory) with List II (used as) and select the correct answer.

	List I		**List II**
A.	DRAM	1.	Cache Memory
B.	SRAM	2.	Main Memory
C.	Parallel access registers	3.	BIOS Memory
D.	ROM	4.	CPU Registers

(*UPSC Civil Services, 2003*)

ANSWERS

1.	*a*-3, *b*-4, *c*-2 *d*-1, *e*-6			**2.**	(*c*)	**3.**	(*b*)	**4.**	(*c*)	**5.**	(*c*)
6.	(*a*)	**7.**	(*c*)	**8.**	(*a*)	**9.**	(*b*)	**10.**	(*c*)	**11.**	(*a*)
12.	(*b*)	**13.**	(*c*)	**14.**	(*c*)	**15.**	(*c*)	**16.**	(*b*)	**17.**	(*a*)
18.	(*c*)	**19.**	(*d*)	**20.**	(*c*)	**21.**	A-3, B-1, C-2				
22.	A-3, B-2, C-4, D-1										

12

PROGRAMMABLE LOGIC DEVICES

OUTLINE

Objectives

After completing this chapter, you should be able to :

- define a Programmable Logic Device (PLD)
- know different types of programmable logic devices
- list the advantages of programmable logic devices
- sketch the logic symbol of PLD
- explain the operation of PROM as a PLD
- understand the programming of PROM as a PLD
- describe the architecture of Programmable Array Logic (PAL)
- explain the programming of PAL
- sketch block diagram of Programmable Logic Array (PLA) and understand its architecture
- list applications of PLAs
- design a combinational circuit using PLA device
- understand the architecture of Generic Array Logic (GAL)
- describe the architecture of Complex Programmable Logic Device (CPLD) and its applications.

❍ know the architecture of Field-Programmable Gate Arrays (FPGAs)

❍ compare the CPLDs and FPGAs

❍ understand the programming of FPGAs

12-1. Introduction

A programmable logic device (or PLD) is a device which is used in many digital electronic designs. Unlike a logic gate, which has a fixed function, these PLDs are very flexible and can implement both combinational and sequential logic functions, such as AND, OR, NAND, counter and shift registers on the same chip. PLD performs most of the complicated logic functions which are programmed into a single chip. Using these PLDs, an effective digital design circuit will be achieved. For the PLD programming we have to select inputs, outputs and the logical relationships of our required system. Then these details will be programmed into a single IC and that does the necessary operation. By using a single IC for many applications, the cost of developing circuit will be lower and reduction of many electronic components in the design. It reduces the power requirement also for a digital circuit. Now we shall study the different types of programmable logic devices, their architecture and implementation of desired logic functions.

12-2. History of PLD

Before the PLDs were invented, read-only memory (ROM) chips were used to create many combinational logic functions for the inputs. Consider a ROM with ‘m’ inputs as address lines and ‘n’ outputs as data lines. This is illustrated in Fig. 12-1.

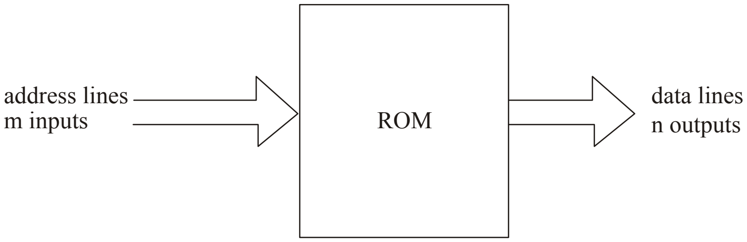


Fig. 12-1. Basic input output system of ROM.

So this ROM has 2^m words of n bits each. Now imagine that inputs are driven not by an m-bit address, but by m independent logic signals. Basically there are 2^m possible functions of these m signals, but the structure of the ROM allows just n of these functions to be produced at the output pins. So the ROM may be considered equivalent to n separate logic circuits, each of which generates a chosen function of the m inputs.

The advantage of using ROM as a PLD is that a required function of the m inputs can be made at any of the n outputs.

We can use standard IC also for the logic design. But we will be facing some of the problems while designing with the standard IC. Some of the problems are listed here.

1. Logic design requires hundreds or thousands of these ICs.
2. We need a bigger board space for fixing the ICs.
3. It takes longer time to insert, solder and for testing of these ICs.
4. It requires to keep large reserves of ICs.

These problems are overcome by using a PLD. Some of the advantages are listed below.

1. It requires less board space.
2. We need smaller enclosure.
3. It requires very few printed circuit boards.
4. It requires lesser power requirements.
5. It requires less time to assemble the ICs.
6. Since we have less ICs fixed on the board, troubleshooting is very simple.

Similarly, PROMs (programmable ROM), EPROMs (Erasable PROMs by ultra-violet) and EEPROMs (Electrically erasable) can be used as conventional PLDs. But these devices have some disadvantages too. Some of the disadvantages are listed below.

1. These devices are actually slower than the dedicated logic functions.
2. These devices require more power.
3. A very small fraction of the output is used, so inefficient use of outputs.
4. These devices cannot give proper outputs for asynchronous logic functions. So the ROM's output may glitch as the inputs switch.
5. Most of ROMs do not have input or output registers and hence these devices cannot be used as stand-alone for sequential logic circuits. An external TTL register is used for sequential designs.

From 1970 to 1975, many companies such as Texas Instruments, National Semiconductors, and GE have manufactured PLD devices on ROM technology. In 1975, Signetics introduced a PLD, that has a large acceptance from the industry side. In 1978, MMI introduced the Programmable Array Logic or PAL. It is composed by a programmable AND matrix to a fixed OR matrix. In the year 1985, Lattice Semiconductor introduce a Generic Array Logic (GAL). This is an extension of PAL with the additional features of being electrically erasable. PALs and GALs are available in small sizes, equivalent to a few hundred logic gates. For bigger logic circuits, Complex PLDs can be used. These CPLDs contain the equivalent of several PALs linked by programmable interconnections, in one integrated circuit. CPLDs can replace thousands or even hundreds of thousands of logic gates.

Another technology, which is based on gate array also called as field-programmable gate array (FPGA), is also used in creation of PLDs. FPGAs use a grid of logic gates, similar to that of an ordinary gate array, but the programming is done by the designer and not by the manufacturer.

12-3. Programmable Logic Device

Programmable logic device is an IC, has the feature of user configuration and capable of implementing various logic functions. It contains large number of logic gates, flip-flops and shift registers. It allows the circuit designer to customize it for any specific application.

Basically a programmable logic device (PLD) consists of two types of logic array. One is an array of AND gates and the other is an array of OR gates. These arrays are programmable to meet the desired logic functions. Fig. 12-2 shows an example of small PLD. The AND array produces the product term whereas the OR array produces sum of these products. Thus the output of PLD is called sum-of-product. The connection between the AND gates and OR gates are in the form of programmable fuses. For a specific function few of the fuses are broken selectively while the other remains intact. The blowing of the fuses is done by the manufacturer under the instruction of designer. The process of blowing fuse is called programming because it produces the desired circuit pattern interconnecting the gates.

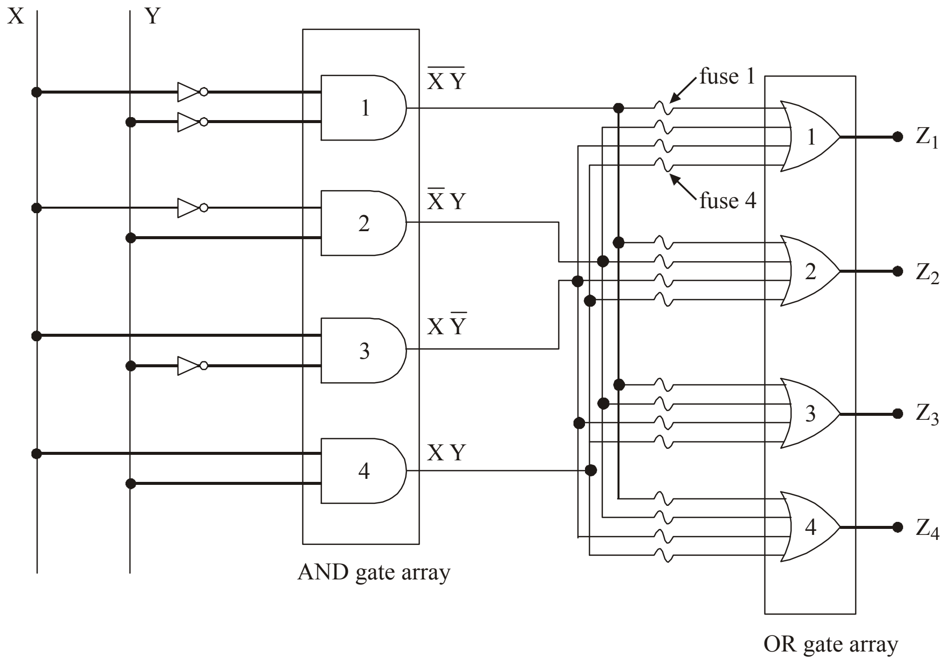


Fig. 12-2. Programmable logic device.

Each gate of AND array have two inputs and each gate of OR array have four inputs. Output of each AND gate is connected to one of 4-inputs of each OR gate through a fusible link. Let us consider initially all links. Each of the four outputs are,

$$\begin{aligned} Z_1 &= Z_2 = Z_3 = Z_4 \\ &= XY + X\overline{Y} + \overline{X}Y + \overline{X}\overline{Y} \\ &= X(Y + \overline{Y}) + \overline{X}(Y + \overline{Y}) \\ &= 1. \end{aligned}$$

Any of the four output can be programmed to be any function of X and Y.

For example, if we blow the fuse 1 of 2nd OR gate then output Z_2 will be,

$$\begin{aligned} Z_2 &= 0 = \overline{X}Y + X\overline{Y} + XY \\ &= X \oplus Y + XY. \end{aligned}$$

Based on our requirement the PLDs are programmed. But once the PLD is programmed, the device will permanently generate the selected output functions.

Programmable logic device may be classified into many types. They are namely,

1. Programmable read-only memory (PROM).
2. Programmable array logic (PAL).
3. Programmable logic array (PLA).
4. Generic Array Logic (GAL).
5. Complex programmable logic device (CPLD).
6. Field programmable gate array (FPGA).

12-4. Advantages of Programmable Logic Devices

Although programmable logic devices have many advantages, yet the following are important from the subject point of view :

1. Low space requirement *i.e.*, high density.
2. Low development cost.
3. Low power consumption.
4. Design security and flexibility.
5. Easy programming.
6. Compact circuitry.
7. High switching speed.

12-5. PLD Symbol

PLD manufacturers have adopted a simplified symbol to represent the internal circuitry. Fig. 12-3 shows the circuit symbol of a 2-input PLD. As seen from this figure, the two logic input gates A and B have two outputs each (*i.e.*, normal output and its complement). These outputs are connected to a 4-input AND gate by connecting the wires like 'x' or '•'. Here 'x' means an intact fuse. A dot (•) means a hardwire connection. This type of connection cannot be changed.

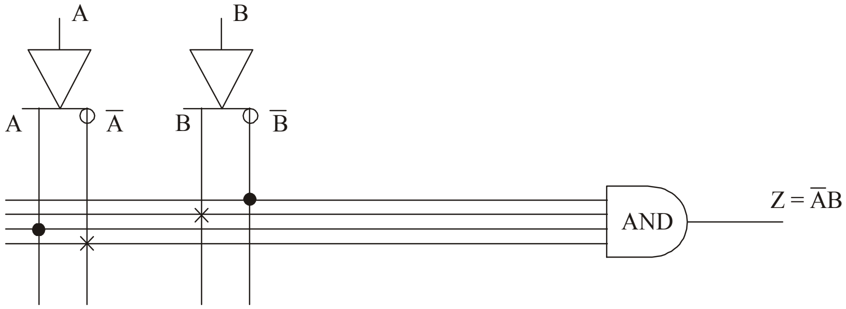


Fig. 12-3. Symbol for PLDs.

In Fig. 12-3, the inputs A and $\overline{B}$ are connected to generate the product AB (*i.e.*, $Z = A\overline{B}$). Here $\overline{A}$ and B are not connected to the AND gate (*i.e.*, intact fuse). So they do not have any effect on its output.

12-6. PROM as a PLD

The programmable ROM is a combinational circuit which can generate any possible logic function of the input variables. The PROM architecture for the PLD is illustrated in Fig. 12-4. As seen from this figure, we find that it has three inputs A, B and C connected through the buffer gates. The outputs of the buffer gates are suitably connected to an AND array. The outputs of the AND array are suitably connected to an OR array which produces the device outputs Z_1, Z_2 and Z_3. Here A '*' represents a hardwired connection. From Fig. 12-4, we find that the AND array is hardwired and the OR array is a programmable one.

A PROM can generate every possible AND product terms. So it will be a good selection for an application which requires every combination of an input variable. But for large number of input variables PROM PLD will not be helpful, since every additional input variable requires two fuses. Normally the PROM has a fixed AND array and the programmable OR array. This can be explained in the following simple block diagram shown in Fig. 12-4.

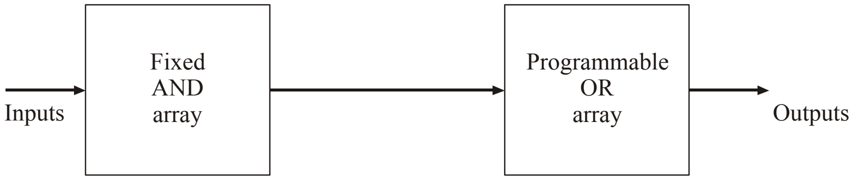


Fig. 12-4. Block diagram of PROM PLD.

The size of the PLD is described in terms of number of input variables, number of product terms and the number of outputs. Fig. 12-5 has 3 inputs that are decoded by the AND array. Each gate of AND array gives one of the possible product (*i.e.*, $\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{B}C$, $\bar{A}BC$, $\bar{A}B\bar{C}$, $A\bar{B}\bar{C}$, $A\bar{B}C$, $AB\bar{C}$ and ABC). The outputs of AND gate are connected to 3 input OR gate through a fuse (represented by X). The blowing of fuse is called programming of PROM. We will study how to program the fuses in the next article.

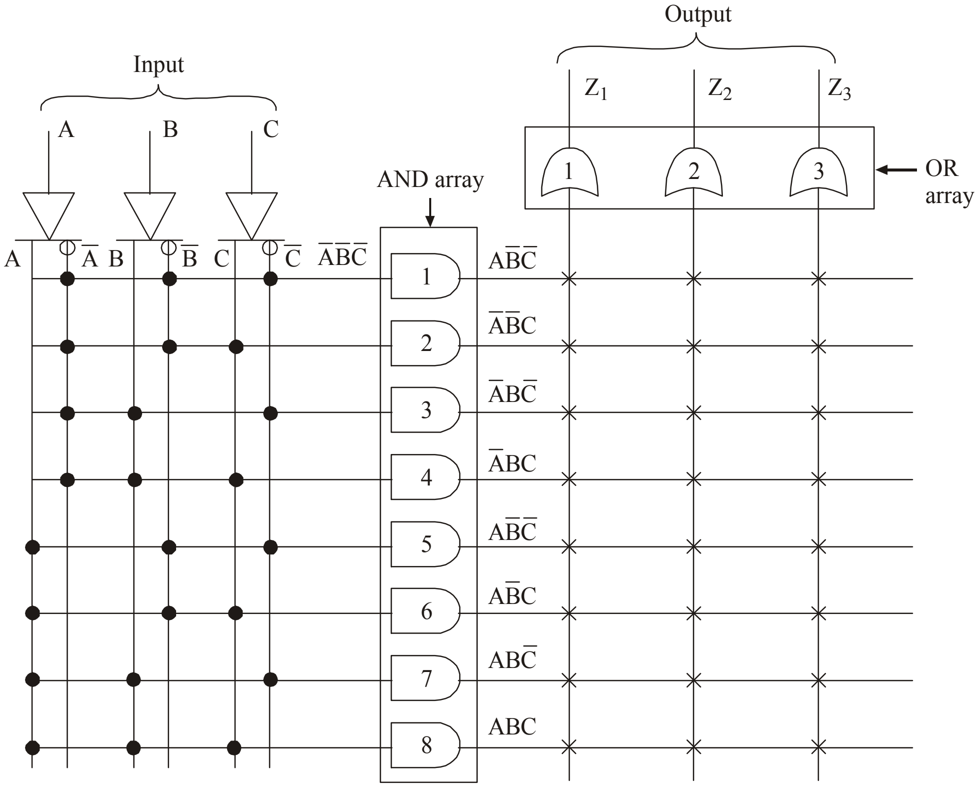


Fig. 12-5. PROM architecture for PLD.

Although PROM PLD has several advantages and disadvantages, yet the following ones are important from the subject point of view :

Advantages :

1. Low cost
2. No simplification of logic function is required.
3. Flexible design.

Disadvantages :

1. Large power requirement.
2. Much slower than the dedicated logic circuits.
3. It is not suitable for large number of inputs.

12-7. Programming the PROM

Fig. 12-6 has three inputs which are connected to eight AND arrays and its outputs are connected to three OR gate arrays. It generates three logic functions namely Z_1, Z_2 and Z_3.

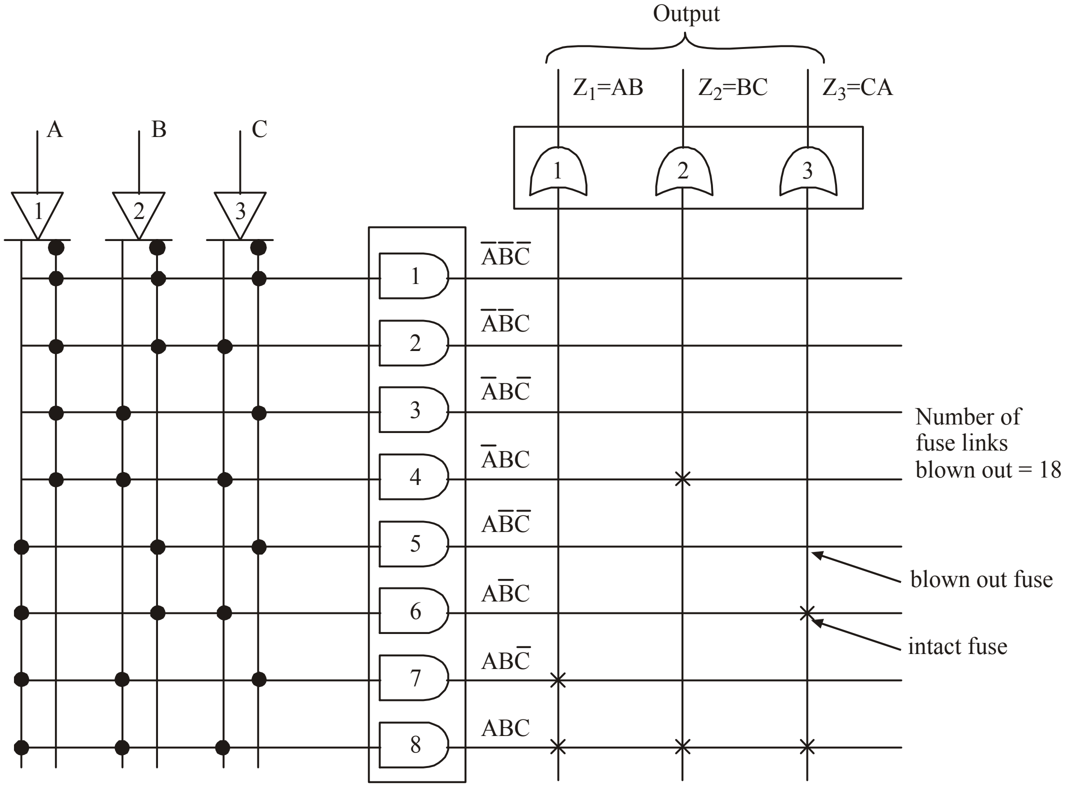

Output
Z_1=AB
Z_2=BC
Z_3=CA
A
B
C
1
2
3
1
2
3
1
2
3
4
5
6
7
8
$\overline{A}\overline{B}\overline{C}$
$\overline{A}\overline{B}C$
$\overline{A}B\overline{C}$
$\overline{A}BC$
$A\overline{B}\overline{C}$
$A\overline{B}C$
$AB\overline{C}$
ABC
Number of fuse links blown out = 18
blown out fuse
intact fuse

Fig. 12-6. Programming of PROM PLDs.

The desired output Z_1 = AB can be explained as follows :

$$
\begin{aligned}
Z_1 &= AB \\
&= AB\ (C + \overline{C}) \qquad \text{(Since } C + \overline{C} = 1) \\
&= ABC + AB\overline{C}
\end{aligned}
$$

So for the desired output Z_1, fuse links from AND gate 1, 2, 3, 4, 5, 6 to OR gate 1 should be blown out. This makes the output at Z_1.

Similarly, the desired output Z_2 can be expanded like this.

$$
\begin{aligned}
Z_2 &= BC \\
&= BC\ (A + \overline{A}) \qquad (\because C + \overline{C} = 1). \\
&= ABC + \overline{A}BC
\end{aligned}
$$

Here the fuse links from AND gates 1, 2, 3, 5, 6, 7 to OR gate 2 should be blown out.

Similarly, the desired output Z_3 can be expanded like this,

$$
\begin{aligned}
Z_3 &= \mathrm{CA} \\
&= \mathrm{CA}\,(\mathrm{B} + \overline{\mathrm{B}}) \qquad (\because \mathrm{B} + \overline{\mathrm{B}} = 1) \\
&= \mathrm{ABC} + \mathrm{A}\overline{\mathrm{B}}\mathrm{C}.
\end{aligned}
$$

For output Z_3, fuse links from AND gates 1, 2, 3, 4, 5 and 7 to OR gate 3 are needs to be blown out. Thus based on the output logic functions, its necessary fuses are blown out. AM 27513 is an example of PROM PLD IC. It has 9 address inputs and 4 data outputs. Thus the output of this IC will be any logic function of the 9 different inputs.

Example 12-1. *A PROM is used to implement the boolean function given below*

$f_1\,(A, B, C, D) = ABCD + \overline{A}\overline{B}\overline{C}\overline{D}$

$f_2(A, B, C, D) = A\overline{B}\overline{C}D + \overline{A}B\overline{C}\overline{D} + AB\overline{C}D$

(*a*) *What is minimum size of PROM required?*

(*b*) *Determine data in each location of PROM.*

Solution : Given : $f_1(\mathrm{A, B, C, D}) = \mathrm{ABCD} + \overline{\mathrm{A}}\overline{\mathrm{B}}\overline{\mathrm{C}}\overline{\mathrm{D}}$

$$f_2(\mathrm{A, B, C, D}) = \mathrm{A}\overline{\mathrm{B}}\overline{\mathrm{C}}\mathrm{D} + \overline{\mathrm{A}}\mathrm{B}\overline{\mathrm{C}}\overline{\mathrm{D}} + \mathrm{AB}\overline{\mathrm{C}}\mathrm{D}.$$

By seeing the given output functions, it is seen that there are four input variables A, B, C and D. Therefore PROM is required to have minimum of 2^n (2^4) = 24 locations.

Table 12-1

LOCATION	INPUTS				OUTPUT	DATA BIT	
	A	B	C	D		D_1	D_0
0	0	0	0	0	$\overline{\mathrm{A}}\overline{\mathrm{B}}\overline{\mathrm{C}}\overline{\mathrm{D}}$	0	1
1	0	0	0	1	$\overline{\mathrm{A}}\overline{\mathrm{B}}\overline{\mathrm{C}}\mathrm{D}$	0	0
2	0	0	1	0	$\overline{\mathrm{A}}\overline{\mathrm{B}}\mathrm{C}\overline{\mathrm{D}}$	0	0
3	0	0	1	1	$\overline{\mathrm{A}}\overline{\mathrm{B}}\mathrm{CD}$	0	0
4	0	1	0	0	$\overline{\mathrm{A}}\mathrm{B}\overline{\mathrm{C}}\overline{\mathrm{D}}$	1	0
5	0	1	0	1	$\overline{\mathrm{A}}\mathrm{BC}\overline{\mathrm{D}}$	0	0
6	0	1	1	0	$\overline{\mathrm{A}}\mathrm{BC}\overline{\mathrm{D}}$	0	0
7	0	1	1	1	$\overline{\mathrm{A}}\mathrm{BCD}$	0	0
8	1	0	0	0	$\mathrm{A}\overline{\mathrm{B}}\overline{\mathrm{C}}\overline{\mathrm{D}}$	0	0
9	1	0	0	1	$\mathrm{A}\overline{\mathrm{B}}\overline{\mathrm{C}}\mathrm{D}$	1	0
10	1	0	1	0	$\mathrm{A}\overline{\mathrm{B}}\mathrm{C}\overline{\mathrm{D}}$	0	0
11	1	0	1	1	$\mathrm{A}\overline{\mathrm{B}}\mathrm{CD}$	0	0
12	1	1	0	0	$\mathrm{AB}\overline{\mathrm{C}}\overline{\mathrm{D}}$	0	0
13	1	1	0	1	$\mathrm{AB}\overline{\mathrm{C}}\mathrm{D}$	1	0
14	1	1	1	0	$\mathrm{ABC}\overline{\mathrm{D}}$	0	0
15	1	1	1	1	ABCD	0	1

(*a*) The size of the ROM is **16 × 2** bits.

(*b*) f_1 (A, B, C, D) = $\mathrm{ABCD} + \overline{\mathrm{A}}\overline{\mathrm{B}}\overline{\mathrm{C}}\overline{\mathrm{D}}$

$= \Sigma_{\mathrm{M}}(15, 0)$ (by seeing the truth table of the inputs)

f_2 (A, B, C, D) = $\mathrm{A}\overline{\mathrm{B}}\overline{\mathrm{C}}\mathrm{D} + \overline{\mathrm{A}}\mathrm{B}\overline{\mathrm{C}}\overline{\mathrm{D}} + \mathrm{AB}\overline{\mathrm{C}}\mathrm{D}$

$= \Sigma_{\mathrm{M}}\ (9, 4, 13)$

Table 12-1 shows the data in each location of PROM. By seeing the truth table, databit D_0 represent function f_1 and databit D_1 represent function f_2. Whenever minterm is present in function f_3 1 is placed in bit D_0 and 0 on remaining locations. Similarly, whenever minterm is present for function f_2, 1 is placed in bit D_1 and 0 in other locations.

Example 12-2. *Use a 16 × 4 PROM PLD to form the following output with minterm as*

$$f_1 = \Sigma_{\mathrm{m}}(0, 3, 5, 11)$$

$$f_2 = \Sigma_{\mathrm{m}}(1, 4, 7, 9)$$

Draw the truth table showing location in PROM.

Solution. Given :

Output functions $f_1 = \Sigma_{\mathrm{m}}(0, 3, 5, 11)$

$f_2 = \Sigma_{\mathrm{m}}(1, 4, 7, 9)$

For a 16 × 4 PROM, there are 16 locations and each location has 4 bit each. Fig. 12-36 shows the logic diagram. The active low signal is given to enable pin to enable the chip. The input pins A_0, A_1, A_2 and A_3 are used to select the location from 0 to 15.

The location in the PROM is shown in Table 12-2.

Table 12-2. Truth table for PROM

Location	Data bits			
	D_3	D_2	D_1	D_0
0	0	0	0	1
1	0	0	1	0
2	0	0	0	0
3	0	0	0	1
4	0	0	1	0
5	0	0	0	1
6	0	0	0	0
7	0	0	1	0
8	0	0	0	0
9	0	0	1	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	1
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0

Here we are using only two data bits D_0 and D_1. The remaining bits D_2 and D_3 are zero. The D_0 bit represents f_1, and bit D_1 represents the function f_2. Each minterm in given function has its own address. Whenever a minterm is present in f_1. 1 is placed in D and 0 is placed in other location. Similarly whenever a minterm is present in f_2, place 1 in D_1 and zero in other locations.

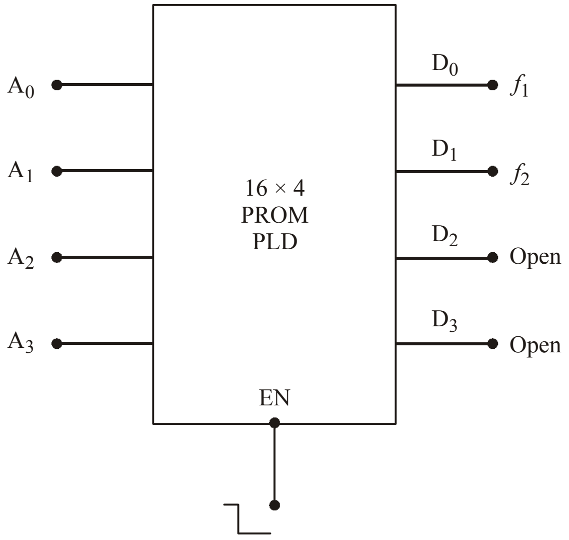


Fig. 12-7. Logic diagram.

Example 12-3. *Using PROM realize the following expressions*

$$f_1 = \Sigma\, m\,(0, 1, 3, 5, 7)$$
$$f_2 = \Sigma\, m\,(1, 2, 5, 6)$$
(*VTU., July/Aug. 2004*)

Solution. Given

$$f_1 = \Sigma\, m\,(0, 1, 3, 5, 7)$$
$$f_2 = \Sigma\, m\,(1, 2, 5, 6)$$

By seeing the given output functions, it is seen that there are four input variables A, B and C. Therefore PROM is required to have minimum of 2^n $(2^3) = 8$ locations

Table 12-3.

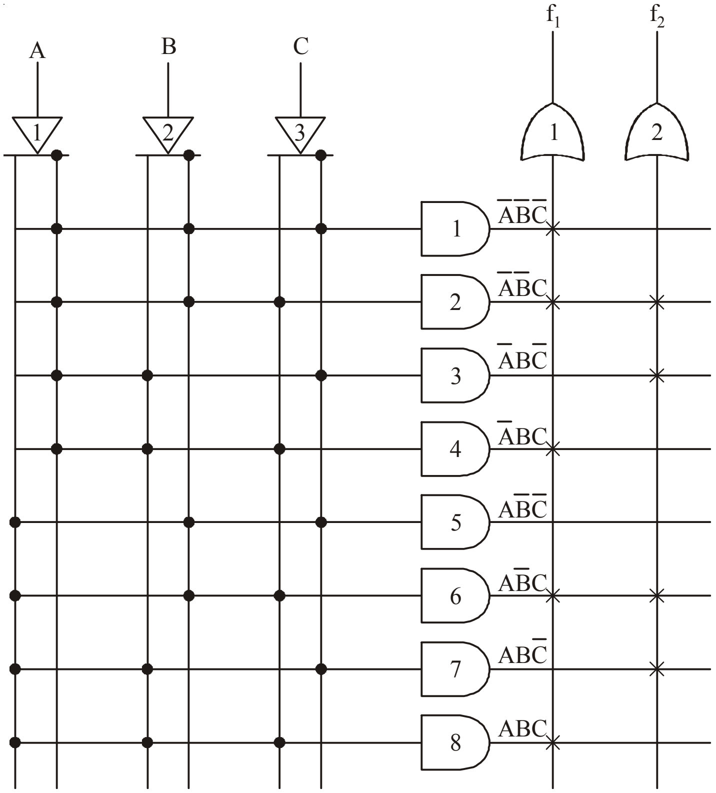
	INPUTS			OUTPUT	f_1	f_2
LOCATION	A	B	C			
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	1	0
1	0	0	1	$\overline{A}\,\overline{B}C$	1	1
2	0	1	0	$\overline{A}B\overline{C}$	0	1
3	0	1	1	$\overline{A}BC$	1	0
4	1	0	0	$A\overline{B}\,\overline{C}$	0	0
5	1	0	1	$A\overline{B}C$	1	1
6	1	1	0	$AB\overline{C}$	0	1
7	1	1	1	ABC	1	0

The size of the PROM is 16 × 2. Table 12-3 shows output in each location of PROM. From the truth table, the functions are,

$$f_1 = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$$

$$f_2 = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}C + AB\overline{C}$$

The functions can be realized as shown in Fig.12-8.



A
B
C
1
2
3
f_1
f_2
1
2
1
$\overline{A}\overline{B}\overline{C}$
2
$\overline{A}\overline{B}C$
3
$\overline{A}B\overline{C}$
4
$\overline{A}BC$
5
$A\overline{B}\overline{C}$
6
$A\overline{B}C$
7
$AB\overline{C}$
8
ABC

Fig. 12-8.

Example 12-4. *Implement the following Boolean expressions using a PROM.*

$$f_1(x_2, x_1, x_0) = \Sigma\, m\,(0, 1, 2, 5, 7)$$

$$f_2(x_2, x_1, x_0) = \Sigma\, m\,(1, 2, 4, 6)$$

(*VTU., Jan. /feb. 2006*)

Solution. Given: The Boolean expressions

$$f_1(x_2, x_1, x_0) = \Sigma\, m\,(0, 1, 2, 5, 7)$$

$$f_2(x_2, x_1, x_0) = \Sigma\, m\,(1, 2, 4, 6)$$

By seeing the given output function then an three variables x_2, x_1 and x_0. Therefore PROM is required to have minimum of 2^n $(2^3) = 8$ locations. Table 12-4 shows the data in each location of PROM. By seeing the truth table, database D_0 represent function f_1 and D_1 represent function f_2.

Table 12-4.

LOCATION	INPUTS			OUTPUT	DATA	BIT
	A	B	C		D_1	D_0
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	1	0
1	0	0	1	$\overline{A}\,\overline{B}\,C$	1	1
2	0	1	0	$\overline{A}\,B\,\overline{C}$	1	1
3	0	1	1	$\overline{A}\,B\,C$	0	0
4	1	0	0	$A\,\overline{B}\,\overline{C}$	0	1
5	1	0	1	$A\,\overline{B}\,C$	1	0
6	1	1	0	$A\,B\,\overline{C}$	0	1
7	1	1	1	ABC	1	0

For the desired output expression f_1, fuses links from AND gate 1, 2, 3, 6, 7 to OR gate 1 should be blown out. This make the output at f_1. Similarly, for the desired output f_2, fuses links from AND gate 2, 3, 5, 7 to OR gate 2 should be blown out as shown in Fig. 12-9.

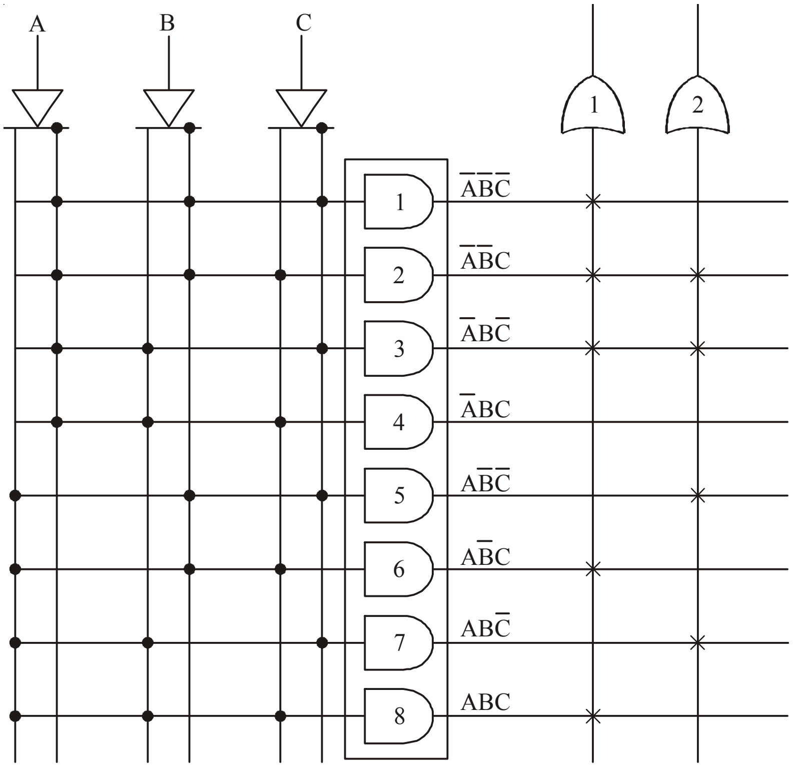

A
B
C
1
2
1
2
3
4
5
6
7
8
$\overline{A}\overline{B}\overline{C}$
$\overline{A}\overline{B}C$
$\overline{A}B\overline{C}$
$\overline{A}BC$
$A\overline{B}\overline{C}$
$A\overline{B}C$
$AB\overline{C}$
ABC

Fig. 12-9.

12-8. Programming Array Logic (PAL)

In the earlier article we have discussed about PROM based PLD device. It has a fixed AND array gates and a programmable OR array. In this article, we will discuss about programming array logic (or PAL). Like in the PROM PLDs, it has the same AND and OR gate arrays in it. But inputs to AND arrays are programmable and inputs to OR arrays are hardwired. Fig. 12-10 shows the simple block diagram of a PAL device.

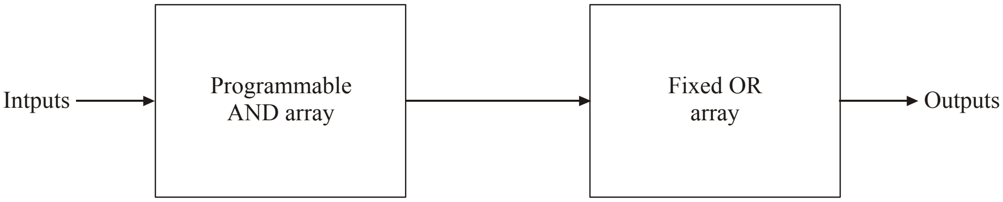


Fig. 12-10. Simple block diagram of PAL.

Fig. 12-11 shows the configuration of AND and OR arrays for PAL architecture. As seen from the figure, there are three inputs A, B and C connected to the buffer gates. The outputs of these buffer gates are suitably connected to 8 AND gates that are programmable one. Then the outputs of these AND gates are connected to the fixed OR gate arrays.

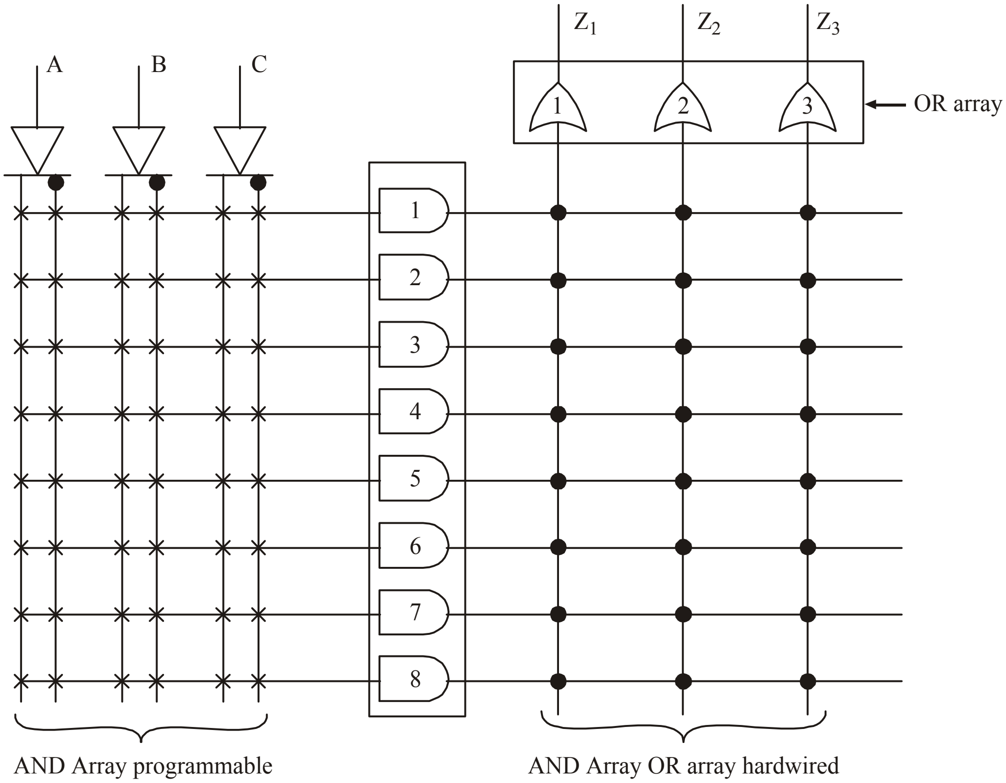


Fig. 12-11. PAL architecture

The number of fuse used in the PAL is the product of 2^M and N. Here M is the number of inputs and N is the product term. Fig. 12-12 contains 24 fuse links (*i.e.*, M = 3, N = 8). Each AND gate can be programmed to generate any desired product term and their complements. Fig. 12-10 shows the programming of PAL device.

If we compare the Fig. of PROM programming with Fig. 12-12 of programming the PAL, blowing of fuse links required for PAL are less. This shows that programming a PAL is much easier than programming a PROM PLDs. In the Fig. 12-12 the blown fuse is represented by '+' and intact fuse is represented by 'X'. When all the inputs to the AND gates are intact, it generates the output as 0.

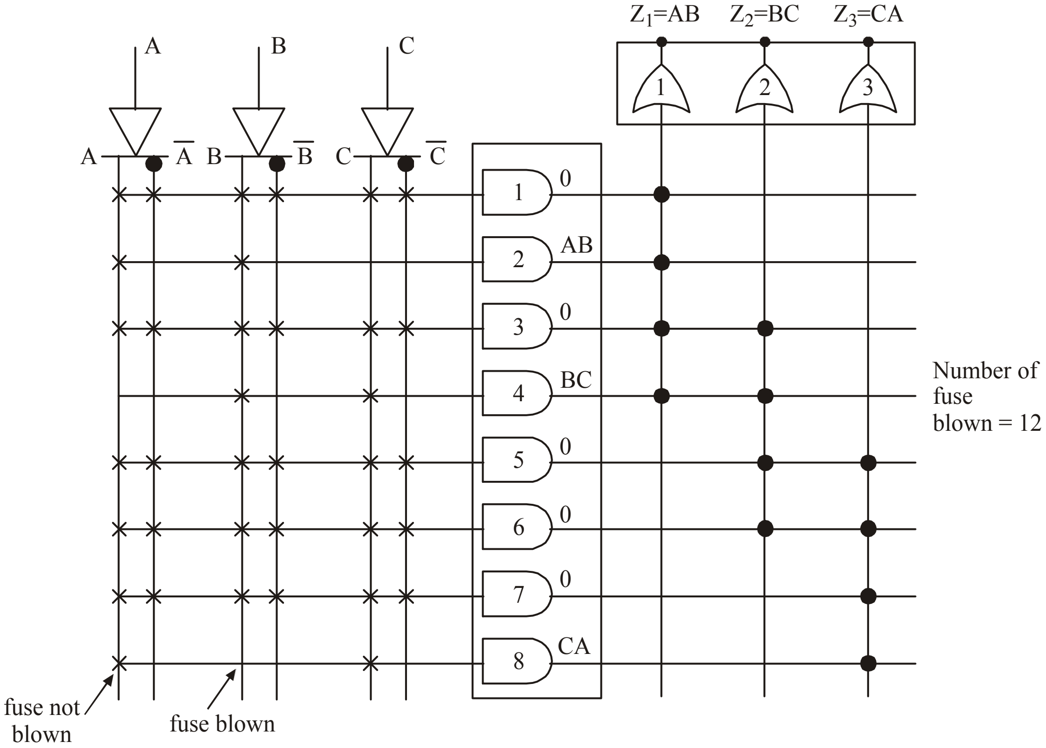


Fig. 12-12. Programming the PAL

PAL 10L8 is an example of PAL IC, manufactured using low power Schottry technology. This IC has 10 logic inputs and 8 output functions. To design a sequential circuit, PALs with flip-flops

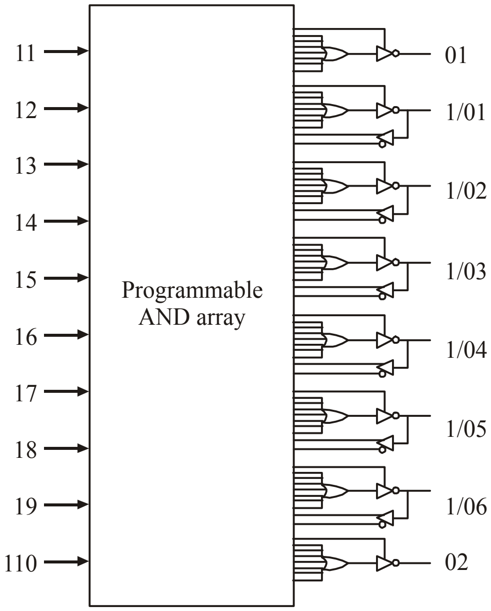


Fig. 12-13. Pin details of PAL 16L8.

have been developed. These are known as registered PALs. Some of the available PALs are 16R4, 16R6, 16R8 etc. Each PAL device is 'one time programmable'. So the program inside the PAL cannot be updated and reused, once the program is done. The PALs are programmed either by electrically using binary files or using some hardware description languages. MMI, Lattice Semiconductor, Zytrex and Altera are some of the PAL IC manufacturers available today. PALs are numbered in such a way that they denote the input and output details. For example, if we use PAL16L8 then it means that it has 16-ten inputs, 8-eight outputs and L-active-Low output. Block diagram and pin-details diagram of PAL16L8 is shown in Fig. 12-13. For further details regarding this chip, please refer www.ti.com.

Example 12-5 : *Draw the logic diagram of PAL to yield following outputs :*

$$\begin{aligned} f_0 &= A+\overline{B}C+0 \\ f_1 &= \overline{A}\overline{B}C+AB\overline{C}+0 \\ f_2 &= A\overline{C}D+\overline{A}\overline{B}+0 \\ f_3 &= A\overline{B}+C\overline{D}+0 \end{aligned}$$

Solution : Given : The input variables, A, B, C and D. We need 16 AND gates. For the four outputs $f_0, f_1, f_2,$ and f_3, we need 4 OR gates. Fig. 12-14 shows the logic diagram of PAL for the given

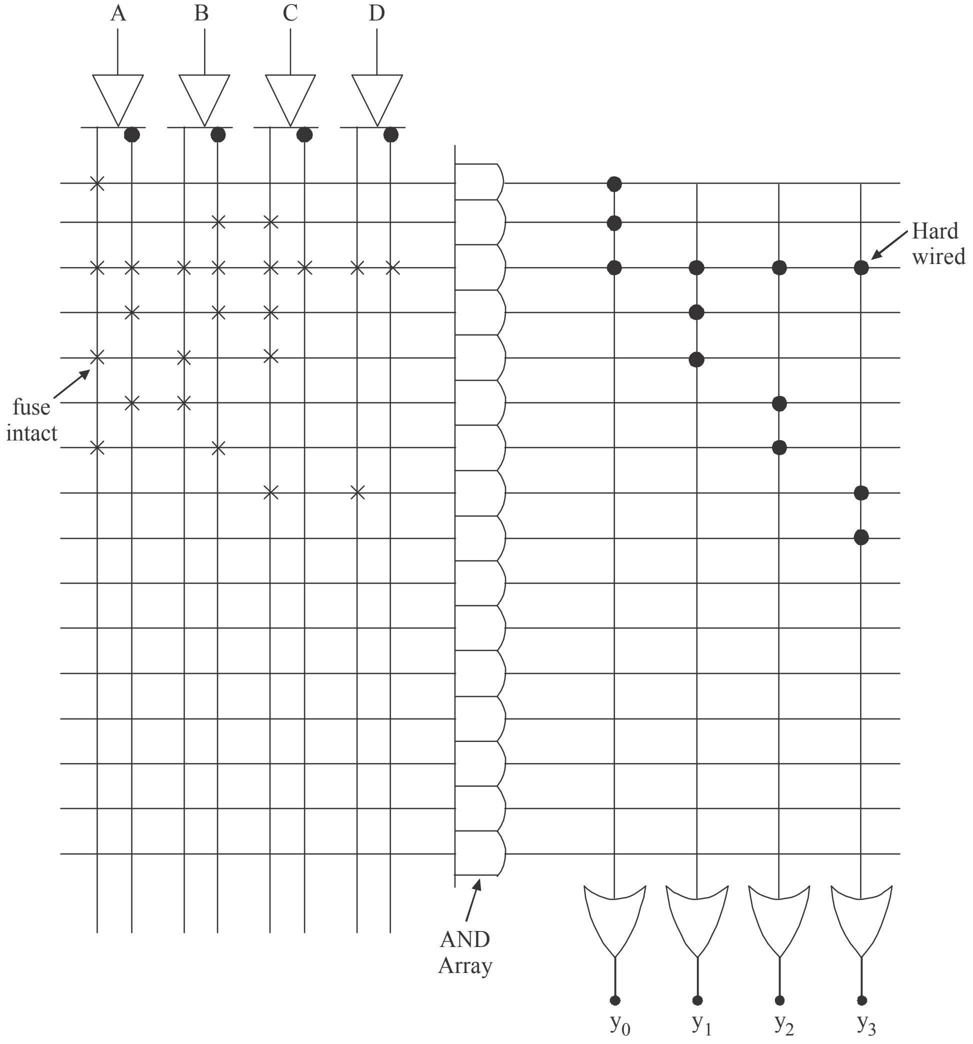


Fig. 12-14.

set of equations. By seeing the logic diagram, we knew that AND gate has 8 inputs and each OR gate has three inputs (Since output function is sum of maximum 3 products).

For the minterm A, all the fuses to AND gate have been blown out except a fuse at input A. For the minterm $\bar{B}C$, all the fuses have been blown out except two fuses — one at input B and the other at input C. For the 0 term, all the fuses to input of the AND gate are kept intact because $A.\bar{A}.B.\bar{B}.C.\bar{C}.D.\bar{D} = 0$. Similarly same procedure is adopted for the remaining terms.

12-9. Programmable Logic Array (PLA)

In the earlier article, we have discussed about PROM PLDs and PALs. Programmable Logic Array (or PLA) is having the characteristics of PROM and PAL. It has the programmable connections for both AND and OR arrays. So it is the most flexible type of PLDs compared to PROMs and PALs. Fig. 12-15 shows a simple block diagram of PLA.

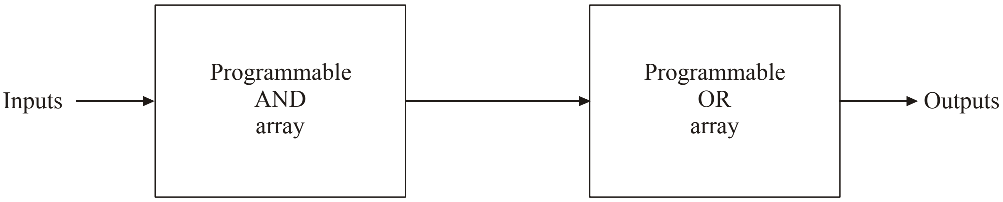


Fig. 12-15. Simple block diagram of PLA.

Normally in PLAs, inputs are fed into a number (K) of AND gates where $K < 2^n$, (n is the number of inputs). Every AND gate can be programmed to generate a product term of the input variables and does not generate all the minterms as in the ROM. The AND and OR gates are connected with the fuses. The specific boolean functions are implemented in sum of products form by opening appropriate fuses and leaving the desired connections.

Fig. 12-16 shows the implementation of PLA. It consists of 'n' inputs, 'm' outputs and 'K' product terms. The product term consists a group of 'K' AND gates each of 2n inputs.

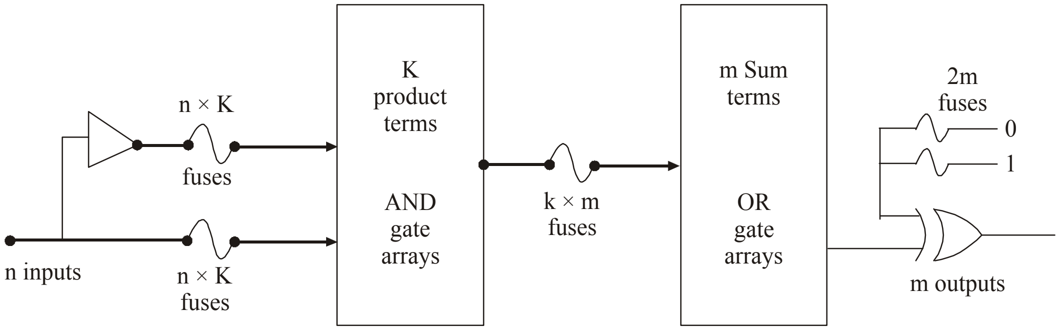


Fig. 12-16. Implementation of PLA.

Here the fuses are inserted between all n inputs and their complement values to each of the AND gates. Fuses are provided between the outputs of the AND gates and the inputs of the OR gates. Since the PLA has m outputs, the number of OR gates is m. The output of each OR gate goes to a XOR gate, where the other input has two sets of fuses, one connected to logic 0 and the other to logic 1. This allows the output function to be generated either in the true form or in the complement form. The output is inverted when the XOR input is connected to 1 (Since $X \oplus 1 = X$). The output does not change when XOR input is connected to 0 (Since $X \oplus 0 = X$).

So, the total number of programmable fuses = 2n + K + K × m + 2 m.

The size of the PLA is specified by the number of inputs (n), the number of product terms (K), and the number of outputs (m), (the number of sum terms is equal to the number of outputs).

Fig. 12-17 shows the architecture of the PLA. There are three inputs passing through the buffer gates. The outputs from the buffer gates are passed to the 8 AND gates array. Then the outputs of AND gates are fed to the OR gate array. From the OR gate, we will get the actual logic functions. Here both AND arrays and OR gate arrays are programmable. So it considers two sets of fuses (*i.e.*, one set for AND array and one set for OR array). The AND gate array provides the product terms and the OR gate logically sum these product terms and gives the required logic function.

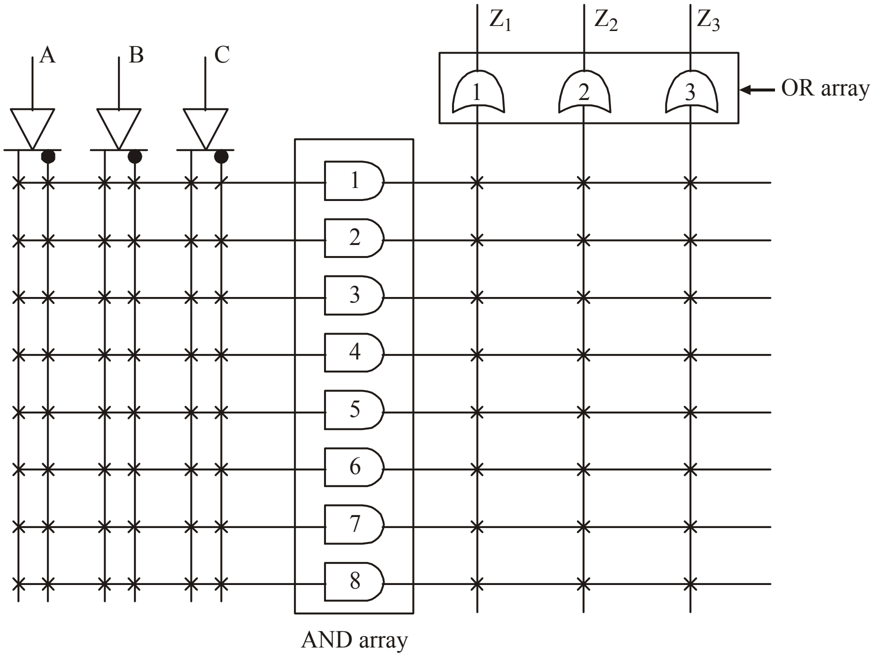


Fig. 12-17. Architecture of PLA.

Since it has two sets of fuses, it is difficult to program the PLA. PLAs are also often referred as FPLA (field programmable logic array).

There are several manufacturers of PLAs today. But Texas Instruments and National Instruments are some of the notable ones. Nowadays PLAs are not popular in commercial market. Instead of PLAs, most of the applications are developed by CPLD and FPGA technologies. We will discuss about these two technologies in the coming articles.

12-10. Application of PLAs

The programming logic array (PLA) can be used to implement combinational and sequential logic circuits. To design a combinational logic circuit PLA with XOR matrix is used, whereas for sequential circuit PLA having flip-flop in output circuit is used.

For some of the applications single PLA capacity is not sufficient. So we have to extend the capacity of the PLA. Fig. 12-18 shows one of possible connection schemes of extending input word length of a PLA.

To extend the input word length, following steps can be used :

1. Connect the inputs of all PLA devices individually in parallel.
2. Connect the outputs of all PLA devices individually in parallel.

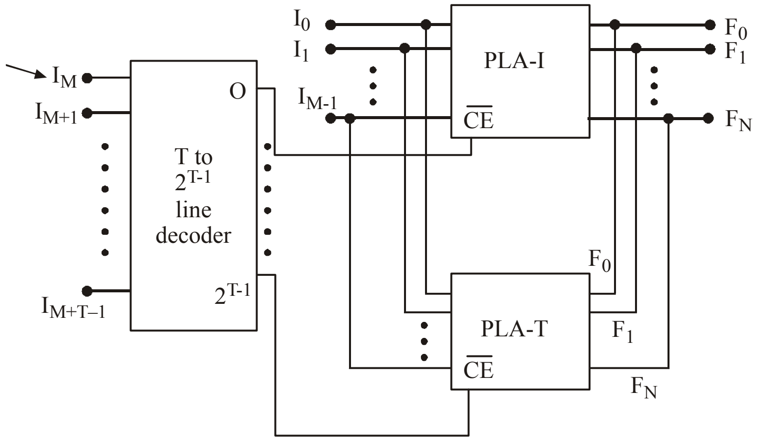


Fig. 12-18. Extending input word length of PLA.

The circuit shown in Fig. 12-18 has (M + T) inputs and N outputs. This connection also increase the number of product term. The number of product term is P × (2^{T-1}). The outputs of PLA should be connected together with passive pull high. Thus the input word can be extended.

Example 12-6. *Design a 3-input, 2 output combinational circuit using a PLA device. The following equations are the outputs with minterm :*

$$f_0 = \Sigma_m\ (0, 5, 9, 15)$$

$$f_1 = \Sigma_m\ (1, 3, 7, 11, 13)$$

Solution. Given : The given output functions are

$$f_0 = \Sigma_m\ (0, 5, 9, 15)$$

$$f_1 = \Sigma_m\ (1, 3, 7, 11, 13)$$

Here the functions f_0 and f_1 are given in minterm forms. The maximum number of minterm is 15. This is forming a 16 product team. Fig. 12-19 shows the architecture of the PLA for the given output. The number of input variables are 4 . ($\because 2^n = 16, n = 4$). Let A, B, C and D are the input variables. Table 12-5 shows the inputs to the AND gate with respect to the minterms.

From the table we can expand the functions such as,

$$f_0 = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}D + A\overline{B}\overline{C}D + ABCD \text{ and}$$

$$f_1 = \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}BCD + A\overline{B}CD + AB\overline{C}D$$

For minterm 0, the fuses from $\overline{A}, \overline{B}, \overline{C}$ and $\overline{D}$ to AND gate are kept intact while the remaining fuses are blown out. Similarly same procedure is applied for all the other minterms as well.

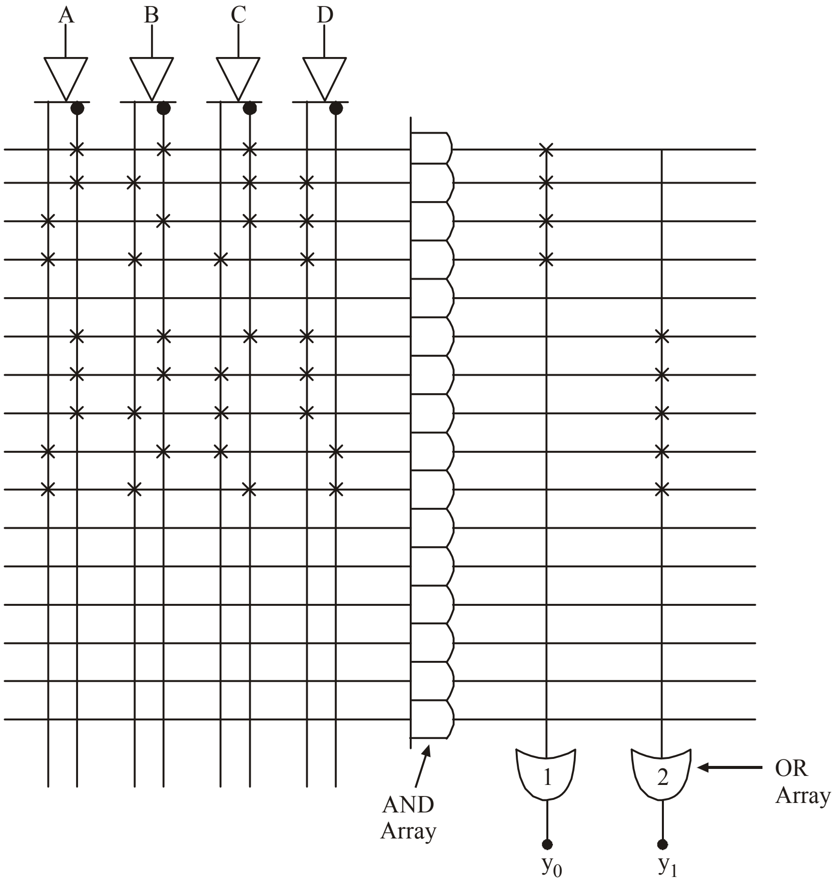


Fig. 12.19.

Table 12-5

Inputs				Minterm	Output
A	B	C	D		
0	0	0	0	0	$\overline{A}\overline{B}\overline{C}\overline{D}$
0	0	0	1	1	$\overline{A}\overline{B}\overline{C}D$
0	0	1	0	2	$\overline{A}\overline{B}C\overline{D}$
0	0	1	1	3	$\overline{A}\overline{B}CD$
0	1	0	0	4	$\overline{A}B\overline{C}\overline{D}$
0	1	0	1	5	$\overline{A}B\overline{C}D$
0	1	1	0	6	$\overline{A}BC\overline{D}$
0	1	1	1	7	$\overline{A}BCD$
1	0	0	0	8	$A\overline{B}\overline{C}\overline{D}$
1	0	0	1	9	$A\overline{B}\overline{C}D$
1	0	1	0	10	$A\overline{B}C\overline{D}$
1	0	1	1	11	$A\overline{B}CD$
1	1	0	0	12	$AB\overline{C}\overline{D}$
1	1	0	1	13	$AB\overline{C}D$
1	1	1	0	14	$ABC\overline{D}$
1	1	1	1	15	$ABCD$

Example 9-7. *Implement the following two Boolean functions with a PLA.*

$$f_0\,(\mathrm{A, B, C}) = \Sigma\,(0, 1, 2, 4)$$

$$f_1\,(\mathrm{A, B, C}) = \Sigma\,(0, 5, 6, 7)$$

(*Anna University, Nov. /Dec. 2007*)

Solution. Given: The given output functions are

$$f_0\,(\mathrm{A, B, C}) = \Sigma\,(0, 1, 2, 4)$$

$$f_1\,(\mathrm{A, B, C}) = \Sigma\,(0, 5, 6, 7)$$

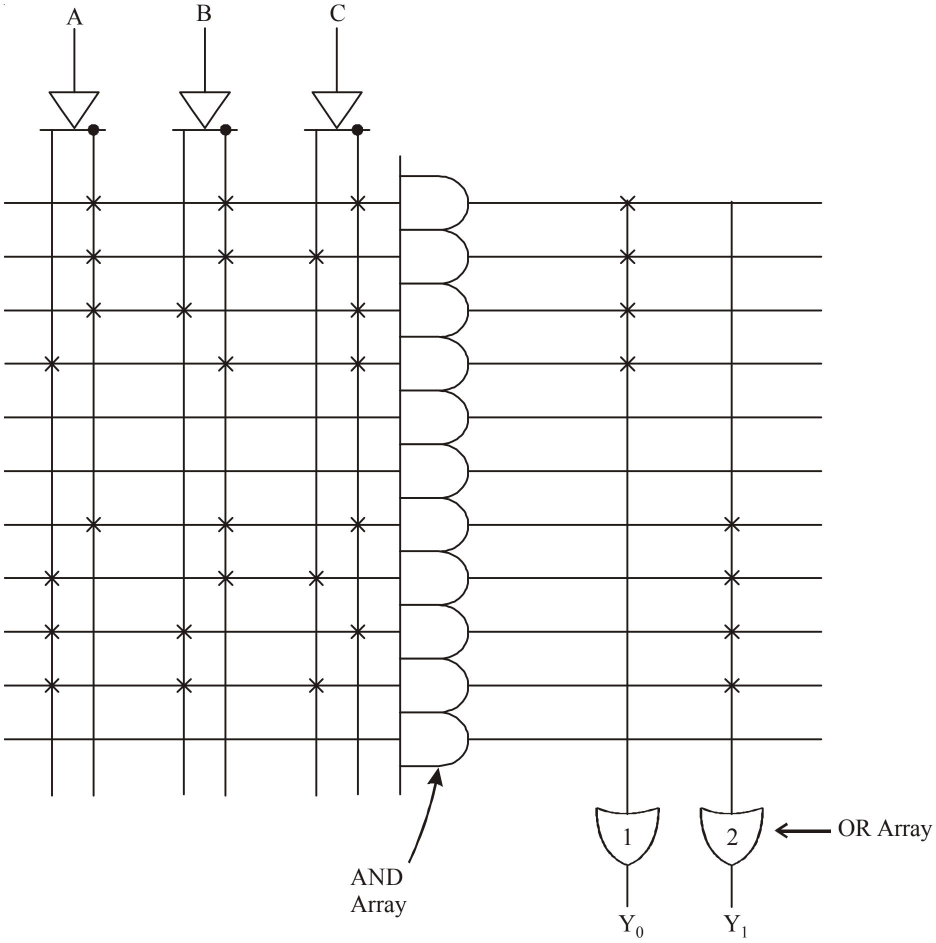


Fig 12-20.

Here the functions f_0 and f_1 are given in minterm forms. The maximum number of minterm is 7. This is forming a 8 product team. Fig 12-20. shows the architecture of the PLA for the given output. The number of variable are 3. ($\therefore$ $2^n = 8$, $n = 3$). Let, A, B and C are the input variables. Table 12-6 shows the input to the AND gate with respect to the minterms.

From the table we can expand the functions such as,

$$f_0 = \overline{\mathrm{A}}\,\overline{\mathrm{B}}\,\overline{\mathrm{C}} + \overline{\mathrm{A}}\,\overline{\mathrm{B}}\mathrm{C} + \overline{\mathrm{A}}\,\mathrm{B}\overline{\mathrm{C}} + \mathrm{A}\overline{\mathrm{B}}\,\overline{\mathrm{C}}$$

$$f_1 = \overline{\mathrm{A}}\,\overline{\mathrm{B}}\,\overline{\mathrm{C}} + \mathrm{A}\overline{\mathrm{B}}\mathrm{C} + \mathrm{AB}\overline{\mathrm{C}} + \mathrm{ABC}$$

For minterm 0, the fuses from $\overline{\mathrm{A}}\,\overline{\mathrm{B}}$ and $\overline{\mathrm{C}}$ to AND gate are kept intant while the remaining fuses are blown out. Similarly same procedure is applied for all the other minterm as well.

Table 12-6.

Input			Minterm	Output
A	B	C		
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	1	$\overline{A}\,\overline{B}\,C$
0	1	0	2	$\overline{A}\,B\,\overline{C}$
0	1	1	3	$\overline{A}\,B\,C$
1	0	0	4	$A\,\overline{B}\,\overline{C}$
1	0	1	5	$A\,\overline{B}\,C$
1	1	0	6	$A\,B\,\overline{C}$
1	1	1	7	ABC

Example 12-8. *Implement using PLA*

$$f_1 = \Sigma\,(0, 3, 4, 7)$$
$$f_2 = \Sigma\,(3, 5, 6, 7)$$

Solution. Given: The given output functions are

$$f_1 = \Sigma\,(0, 3, 4, 7)$$
$$f_2 = \Sigma\,(3, 5, 6, 7)$$

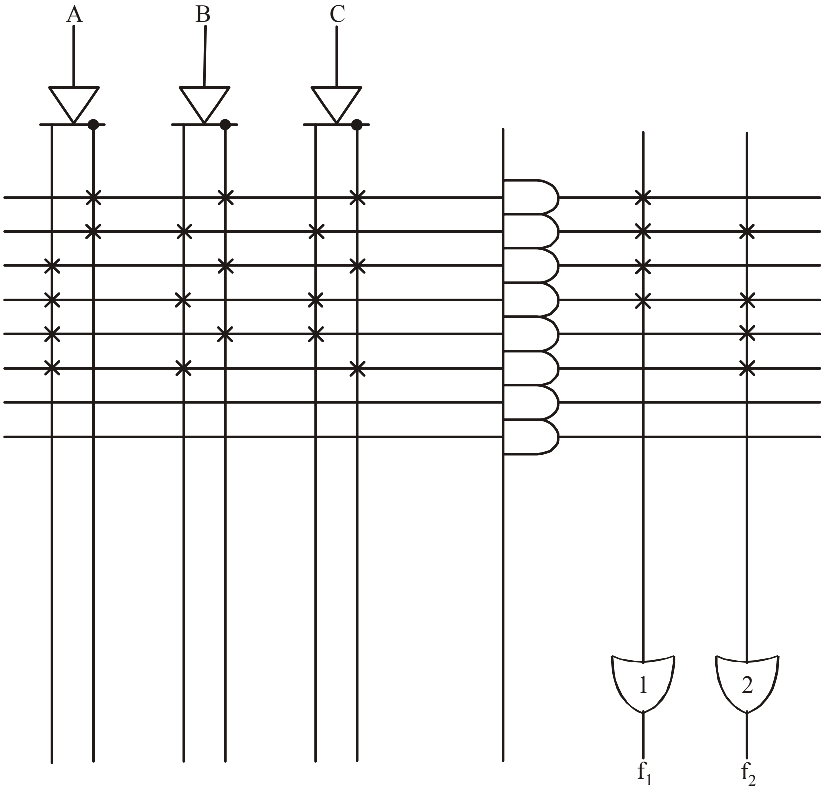


Fig. 12-21.

Here the function f_1 and f_2 are given minterms forms. The maximum number of minterm is 7. This is forming a 8 product terms. Fig. 12-21. shows the architecture of the PLA for the given output. The number of input variables are 3. ($\therefore 2^n = 8, n = 3$). Let A, B and C are the input variables. Fig.12-22 shows the inputs of the AND gate with respect to the minterms.

From the table we can expand the function such as

$$f_1 = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$$

$$f_2 = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

For minterm 0, the fuses from $\bar{A}$, $\bar{B}$ and $\bar{C}$ to AND are kept intant while the other remaining fuses are blown out. Similarly same procedure is applied for all the other minterm as well.

Inputs			Minterm	Output
A	B	C		
0	0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	1	$\bar{A}\bar{B}C$
0	1	0	2	$\bar{A}B\bar{C}$
0	1	1	3	$\bar{A}BC$
1	0	0	4	$A\bar{B}\bar{C}$
1	0	1	5	$A\bar{B}C$
1	1	0	6	$AB\bar{C}$
1	1	1	7	ABC

Fig. 12-22.

Example 12-9. *Implement using PLA*

$$f_1 = \Sigma\,(0, 3, 4, 7)$$

$$f_2 = \Sigma\,(3, 5, 6, 7)$$

(*Mahatma Gandhi University, Nov. 2005*)

Solution. Given: The given output functions are

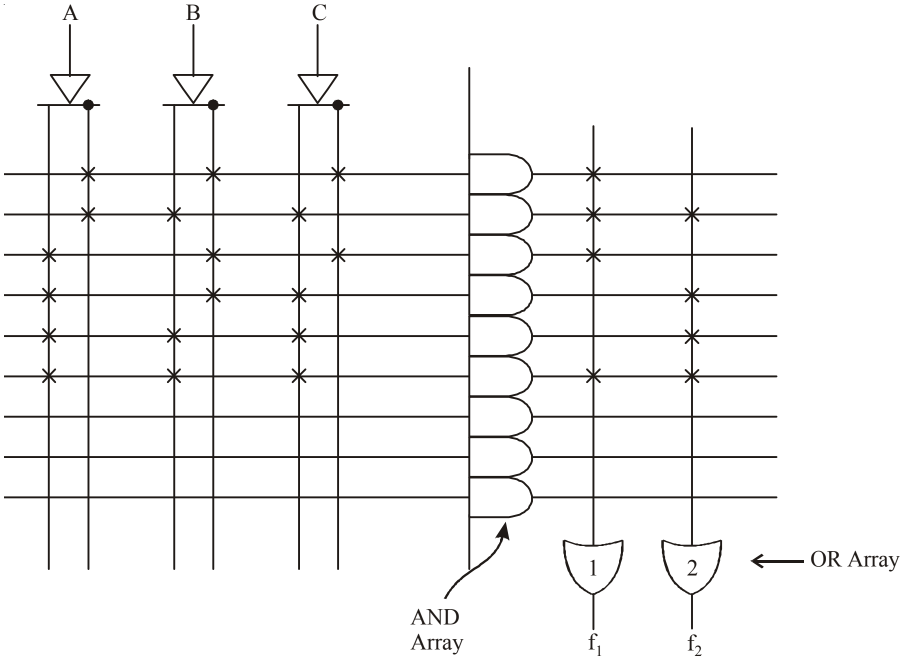


Fig. 12-23.

$$f_1 = \Sigma(0, 3, 4, 7)$$
$$f_2 = \Sigma(3, 5, 6, 7)$$

Here the functions f_1 and f_2 are given in minterm forms. The maximum number of minterm is 7. This is forming a 8 product term. Fig. 12-23, shows the architecture of the PLA for the given output. The number of input variables are 3. ($\therefore$ $2^n = 8, n = 3$). Lets A, B and C are the input variable. Table 12-7. shows the input of the AND gate with respect to the minterms.

From the table we can expand the functions such as

$$f_1 = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$$
$$f_2 = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Table 12-7.

Inputs			Minterm	Output
A	B	C		
0	0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	1	$\bar{A}\bar{B}C$
0	1	0	2	$\bar{A}B\bar{C}$
0	1	1	3	$\bar{A}BC$
1	0	0	4	$A\bar{B}\bar{C}$
1	0	1	5	$A\bar{B}C$
1	1	0	6	$AB\bar{C}$
1	1	1	7	ABC

Example 12-10. *Illustrate how a PLA can be used for combinational logic design with reference to the functions*

$$f_1(A, B, C) = \Sigma\, m\,(0, 1, 3, 4)$$
$$f_2(A, B, C) = \Sigma\, m\,(1, 2, 3, 4, 5)$$

Realize the same assuming, that a 3 × 4 × 2 PLA is available. (*VTU., Jan. /Feb. 2004*)

Solution. Given: The functions are,

$$f_1(A, B, C) = \Sigma\, m\,(0, 1, 3, 4)$$
$$f_2(A, B, C) = \Sigma\, m\,(1, 2, 3, 4, 5)$$

The functions indicate that is output is 1 corresponding to the term indicated 0, 1, 3 and 4 in function f_1 and term indicated 1, 2, 3, 4 and 5 in function f_2.

This is shown in Fig. 12-24.

Inputs			Minterm	Output		AND Gate
A	B	C		f_1	f_2	output
0	0	0	0	1	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	1	1	1	$\bar{A}\bar{B}C$
0	1	0	2	0	1	$\bar{A}B\bar{C}$
0	1	1	3	1	1	$\bar{A}BC$
1	0	0	4	1	1	$A\bar{B}\bar{C}$
1	0	1	5	0	1	$A\bar{B}C$
1	1	0	6	0	0	$AB\bar{C}$
1	1	1	7	0	0	ABC

Fig. 12-24.

From the table we can expend the functions such as,

$$f_1 = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}\,\overline{C}$$

$$f_2 = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} + A\overline{B}C$$

The maximum number of minterm is 7. This is forming a 8 product team. Fig. 12-25 shows the architecture of the PLA for the given output.

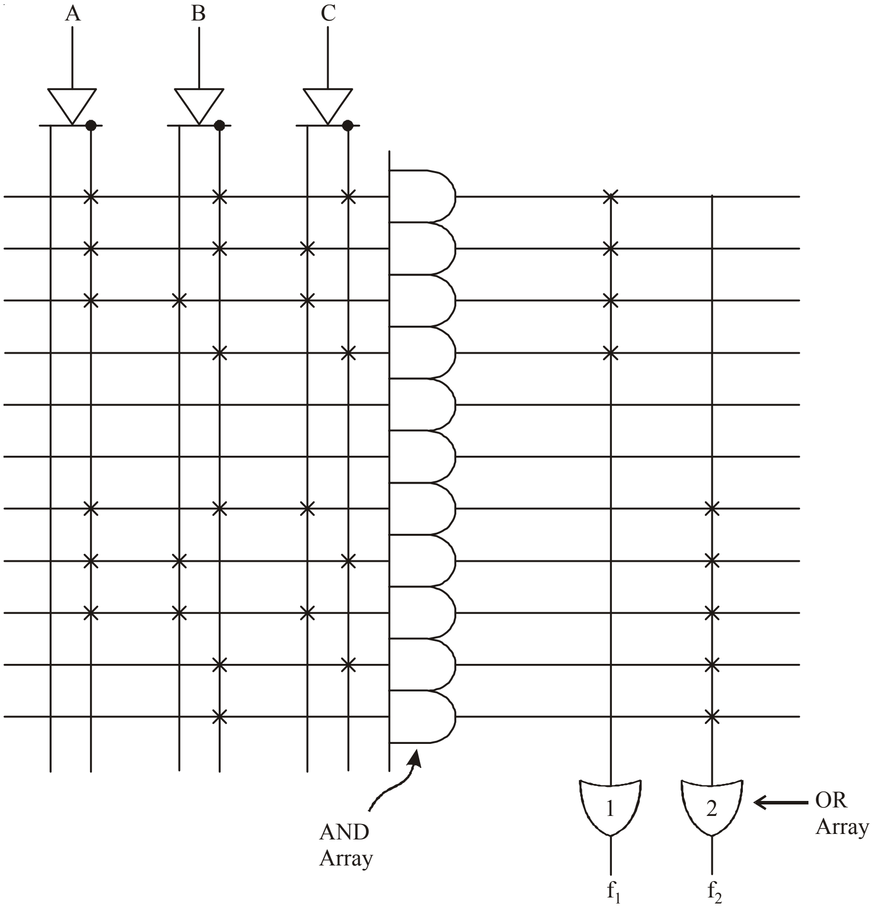
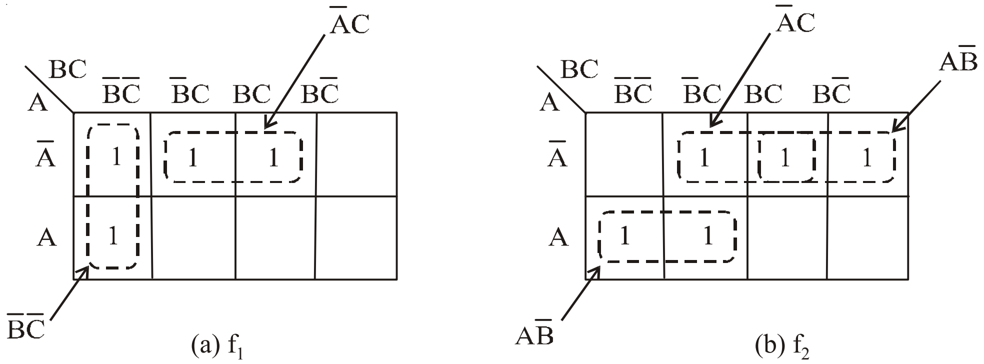


Fig. 12-25.

Let us simplify the function using K-maps. There are three variables A, B and C. The K-map for function f_1 and f_2 is shown in Fig. 12-26 (a) and (b).



(a) f_1 (b) f_2

Fig. 12-26.

To simplify the Boolean expression on the Karnaugh map, group the 1s as shown in K-map. Thus the resulting simplified expression

$$f_1(A, B, C) = \overline{A}C + \overline{B}C$$
$$f_2(A, B, C) = A\overline{B} + \overline{A}C + \overline{A}B$$

In above functions, the term $\overline{A}C$ are common in both function. There are only four product $\overline{A}C$, $\overline{B}C$, $A\overline{B}$ and $\overline{A}B$. These two function can be implement using 3 × 4 × 2 PLA as shown in Fig. 12-27.

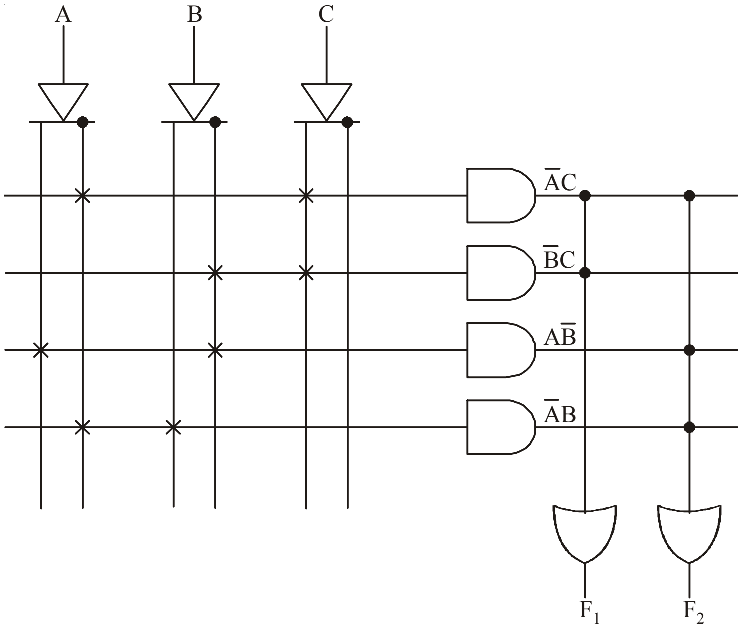


Fig. 12-27.

Example 12-11. *Implement the following multi Boolean function using 3 × 4 × 2 PLA PLD*

$$f_1(a_1, a_2, a_0) = \Sigma\, m\,(0, 1, 3.\ 5) \text{ and}$$
$$f_2(a_2, a_1, a_0) = \Sigma\, m\,(3, 5, 7)$$

(*VTU., July /Aug. 2005*)

Solution. Given: The functions are

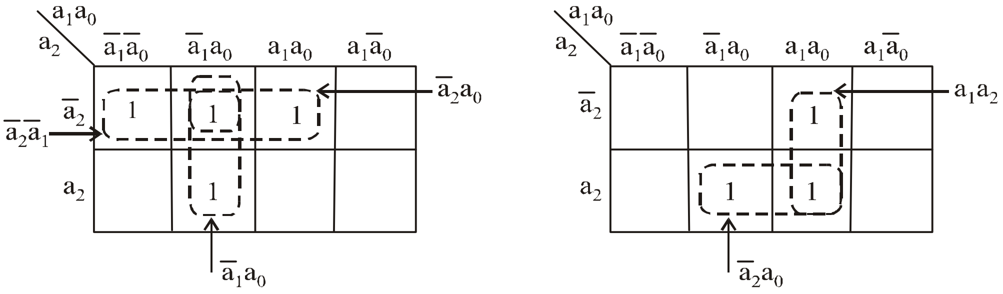
$$f_1(a_1, a_2, a_0) = \Sigma\, m\,(0, 1, 3.\ 5) \text{ and}$$
$$f_2(a_2, a_1, a_0) = \Sigma\, m\,(3, 5, 7)$$

Let us simplify the function using K- maps. There are three variable a_2, a1 and a_0. The output is 1 corresponding to the term 0, 1, 3 and 5 in function f_1 and 3, 5 and 7 in the function f_2. The table for function f_1 and f_2 is shown in Fig. 12-28.

Inputs			Minterm	Output		
a_2	a_1	a_2		f_1	f_2	
0	0	0	0	1	0	$\overline{a}_2\overline{a}_1\overline{a}_2$
0	0	1	1	1	0	$\overline{a}_2\overline{a}_1 a_0$
0	1	0	2	0	0	
0	1	1	3	1	1	$\overline{a}_2 a_1 a_0$
1	0	0	4	0	0	
1	0	1	5	1	1	$a_2\overline{a}_1 a_0$
1	1	0	6	0	0	
1	1	1	7	0	1	$a_2 a_1 a_0$

Fig. 12-28.

In order to simplify the Boolean expression represented on the Karnaugh map, group the 1s as shown in Fig. 12-29.



(a) K-map for function f_1 (b) K-map for function f_2

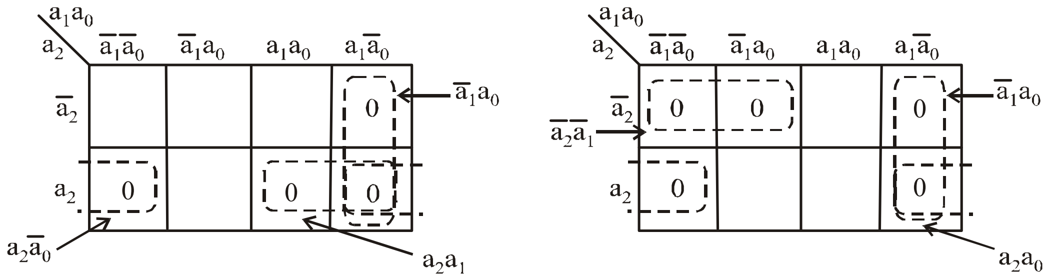
Fig. 12-29.

The functions are,

$$f_1 = \bar{a}_2\bar{a}_1 + \bar{a}_2a_0 + \bar{a}_1a_0$$

$$f_2 = a_2a_0 + a_1a_0$$

To implement the above function it requires 3 × 5 × 2 PLA, but in question we have to implement the functions using 3 × 4 × 2 PLA. So we to consider the product terms by grouping 0_5 in the K-map for function f_1 and f_2. The K-map for 0_5 function is shown in Fig. 12-30.



(a) function f_1 (b) function f_2

Fig. 12-30.

The functions are,

$$f_1 = a_2\bar{a}_0 + \bar{a}_1a_0 + a_2a_1$$

$$f_2 = \bar{a}_2\bar{a}_1 + a_1\bar{a}_0 + a_2\bar{a}_0$$

In above functions, the term $a_2\bar{a}_0$ and $a_1\bar{a}_0$ are common in both function. There are only four product term and function can be implemented using a 3 × 4 × 2 PLA as shown in Fig. 12-31.

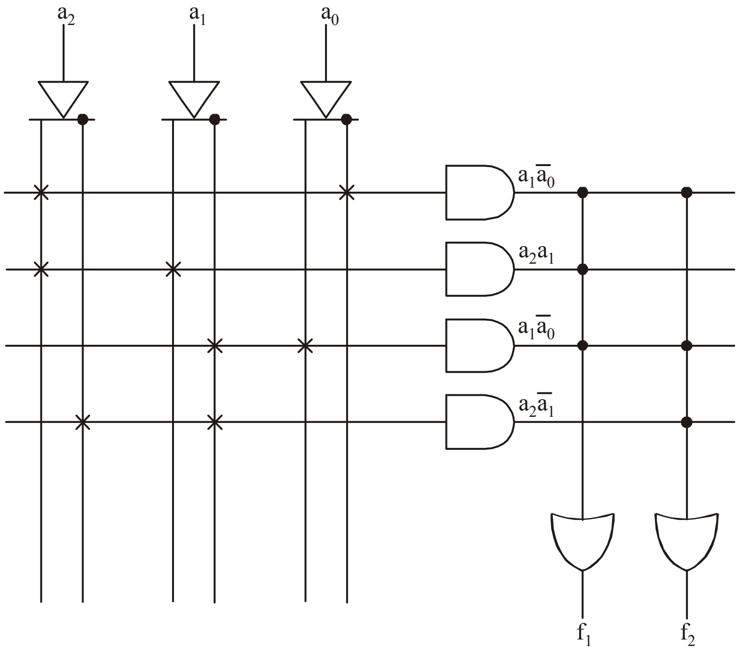


Fig. 12-31.

Example 12.12. *A combinational circuit is defined by function :*

$$F_1\,(A, B, C) = \Sigma\,(3, 5, 6, 7)$$

$$F_2\,(A,B, C) = \Sigma\,(0, 2, 4, 7)$$

Implement the circuit with PLA having three inputs, four product term and two outputs.

(*GTU., May 2011*)

Solution. Given : The given output functions are

$$F_1\,(A, B, C) = \Sigma\,(3, 5, 6, 7)$$

$$F_2\,(A, B, C) = \Sigma\,(0, 2, 4\ 7)$$

Here the functions F_1 and F_2 are given in minterm forms. The maximum number of minterm is 7. This is forming a 16 product team. Fig. 12-32. shows the architecture of the PLA for the given output. The number of input variables are 3 (∵ 2n = 8, n = 3). Let A, B, and C are the input variables. Table 12-8 shows the inputs to the AND gate with respect to the minterms.

From the table we can expand the functions such as,

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

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

For minterms 0, the fuses from $\bar{A}$, $\bar{B}$ and $\bar{C}$ to AND gate are kept intact while the remaining fuses are blown out. Similarly same procedure is applied for all the other minterms as well.

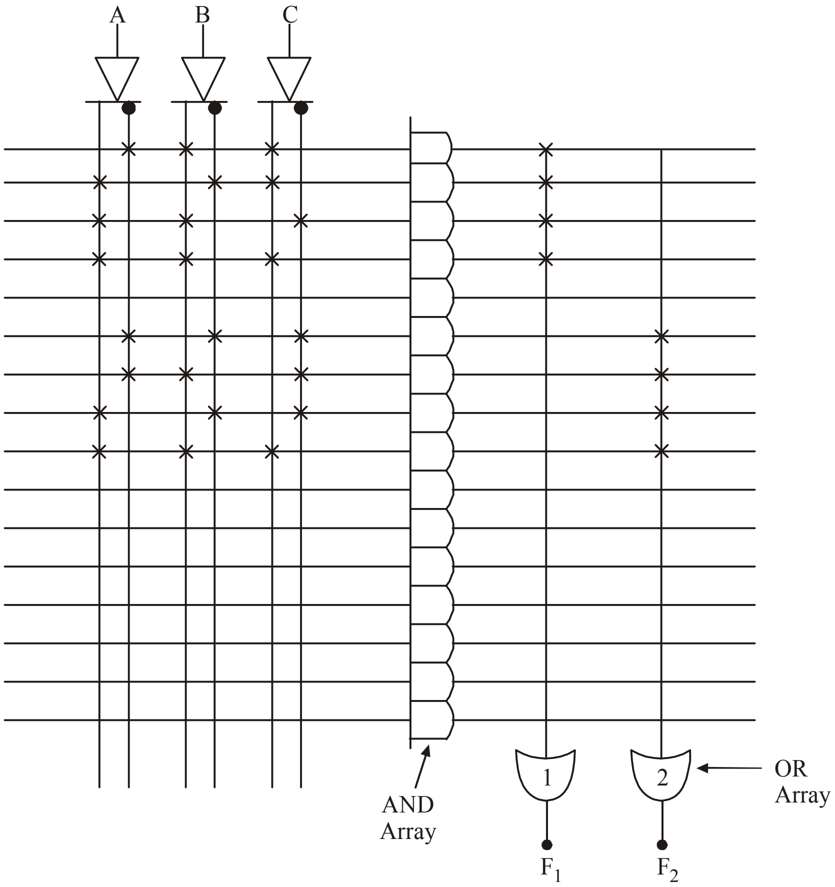


Fig. 12.32.

Table 12-8.

Inputs			***Minterms***	***Outputs***
A	***B***	***C***		
0	0	0	0	$\overline{A}\overline{B}\overline{C}$
0	0	1	1	$\overline{A}\overline{B}C$
0	1	0	2	$A\overline{B}C$
0	1	1	3	$A\overline{B}\overline{C}$
1	0	0	4	$\overline{A}BC$
1	0	1	5	$\overline{A}B\overline{C}$
1	1	0	6	$\overline{A}\overline{B}C$
1	1	1	7	$\overline{A}\overline{B}\overline{C}$

12-11. Generic Array Logic

The continuous development in design and development of PAL device has led to the introduction of configurable PAL (or CPAL) to enhance the output capabilities. The word configurable is also known as generic. Nowadays PALs are no longer used, because GALs provide a superior functionality and can do all the functions of PAL. GALs are relatively small, easily available and inexpensive. Even though GALs are introduced by Lattice Semiconductors, nowadays so many companies including Texas Instruments, Cypress Semiconductors are manufacturing.

A Generic Array Logic (or GAL) is one type of configurable PAL. It can be used as a pin-to-pin replacement for many PAL devices. These devices can be erased and re-programmed whenever a need arises. So these devices are also called as EPLDs (Erasable and Programmable PLDs). Basic block diagram GAL is shown in Fig. 12-33.

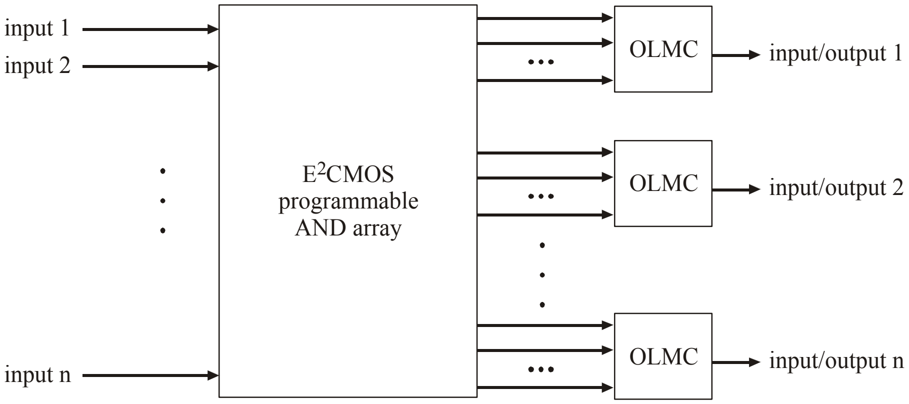


Fig. 12-33. Block diagram GAL.

Most of the GAL's logic functions are performed by programming the AND array. But selecting flip-flops, input/output configurations and polarities of ORs are done by programming a configurable Input/Output and feedback structure. This is called as macrocell. Block diagram of macrocell is shown in Fig. 12-34. The basic operation of macrocell is to determine how the AND/OR Boolean expression is handled and how the macrocell's association with Input/Output pin operates. The multiplexer determines the polarity of the final OR/NOR terms, regardless of whether the term is registered and whether the feedback signal is directly taken from flip-flop's output or at the pin. The behaviour of pins are identified by configuring the macrocell's output.

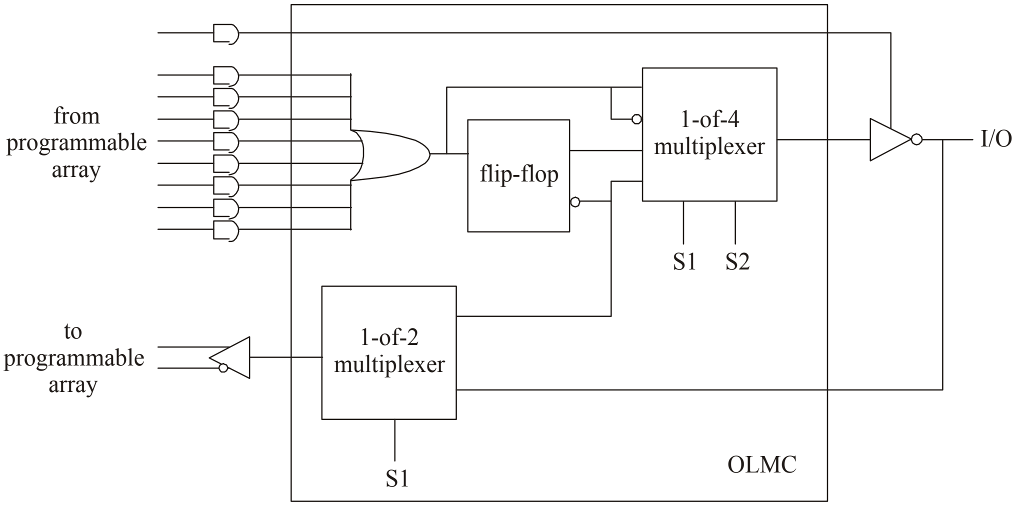


Fig. 12-34. Block diagram of macrocell.

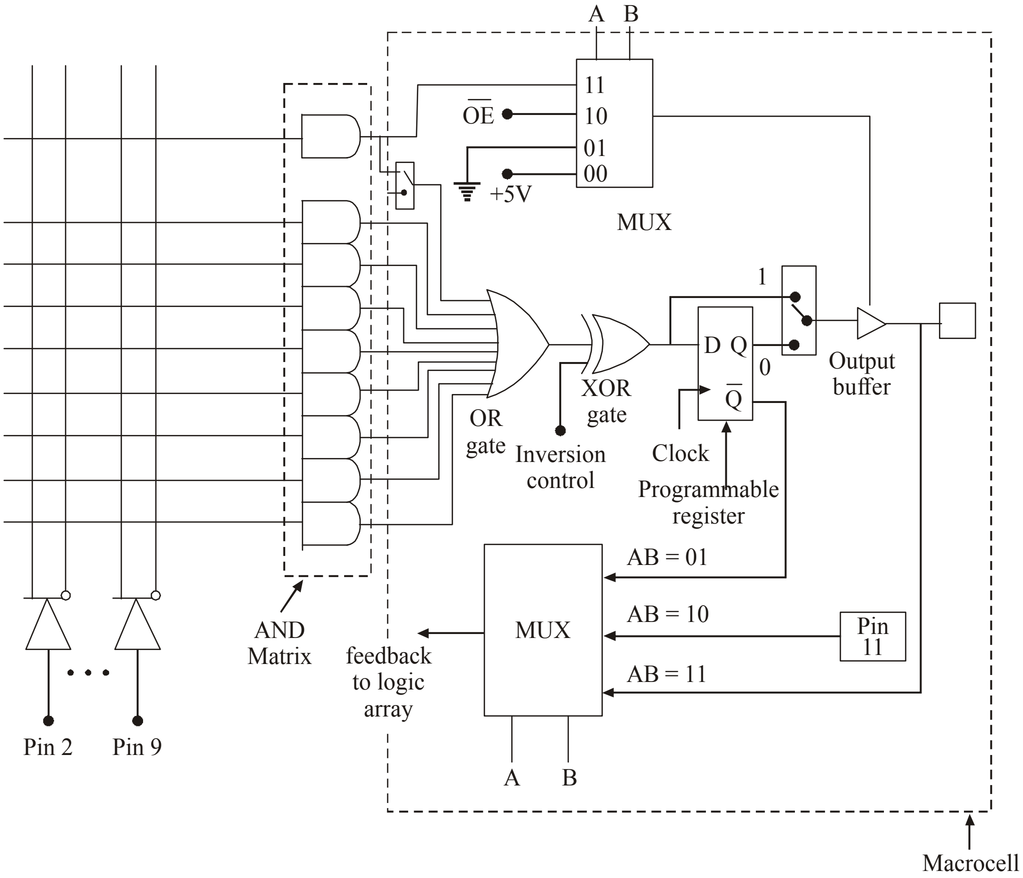


Fig. 12-35. Output Logic Macrocell of GAL16V8.

There are two common GAL devices. Those are GAL 16V8 and GAL 22V10. Now we will discuss about GAL16V8. This device is a 20 pin device, and can be used to replace 20 pin PLAs. Its architecture is similar to PAL device. Out of 20 pins, 8 pins are dedicated inputs (pins 2 to 9). Pins 1 and 11 are the two special function inputs which can be used as both input and output. The major components of GAL devices are the input term matrix, the AND gates which provide the product of input terms and the output logic macrocells (OLMC).

Fig. 12-35 shows the logic diagram of output logic macrocell for GAL16V8. The macrocell includes a product term block with 8 AND input terms feeding an OR gate. One product term is dedicated to the output enable (OE) Control of the tri-state buffer. The macrocell of the GAL16V8A is designed to have three modes namely simple mode, complex mode and the registered mode.

For GAL16V8, pin 10 is the ground pin and pin 20 is the V_{cc} pin (+5V). Pins 12 to 19 are connected to output logic macrocell (OLMC). The OLMC allows these pins to act as inputs, combinational outputs, registered output and simple input/output pins.

When the registered mode is selected, feedback from the programmable register is selected and given to the logic array *i.e.*, A = 0 and B = 0. For this mode pin 1 is a clock input and pin 11 is the output enable pin for the registered outputs. For the remaining modes pin 1 and pin 11 act as a general purpose inputs. For the simple mode, the feedback is taken from pin number 11 (*i.e.*, A = 1 and B = 0). When the combinational mode is selected, feedback comes from the I/O pin through the output buffer *i.e.*, A = 1 and B = 1. Now data is fed into the macrocell's D register on the rising edge of the clock. The inversion control can implement active high or active low to reduce the number of product terms. The output programmable connections is shown in Fig. 12-36.

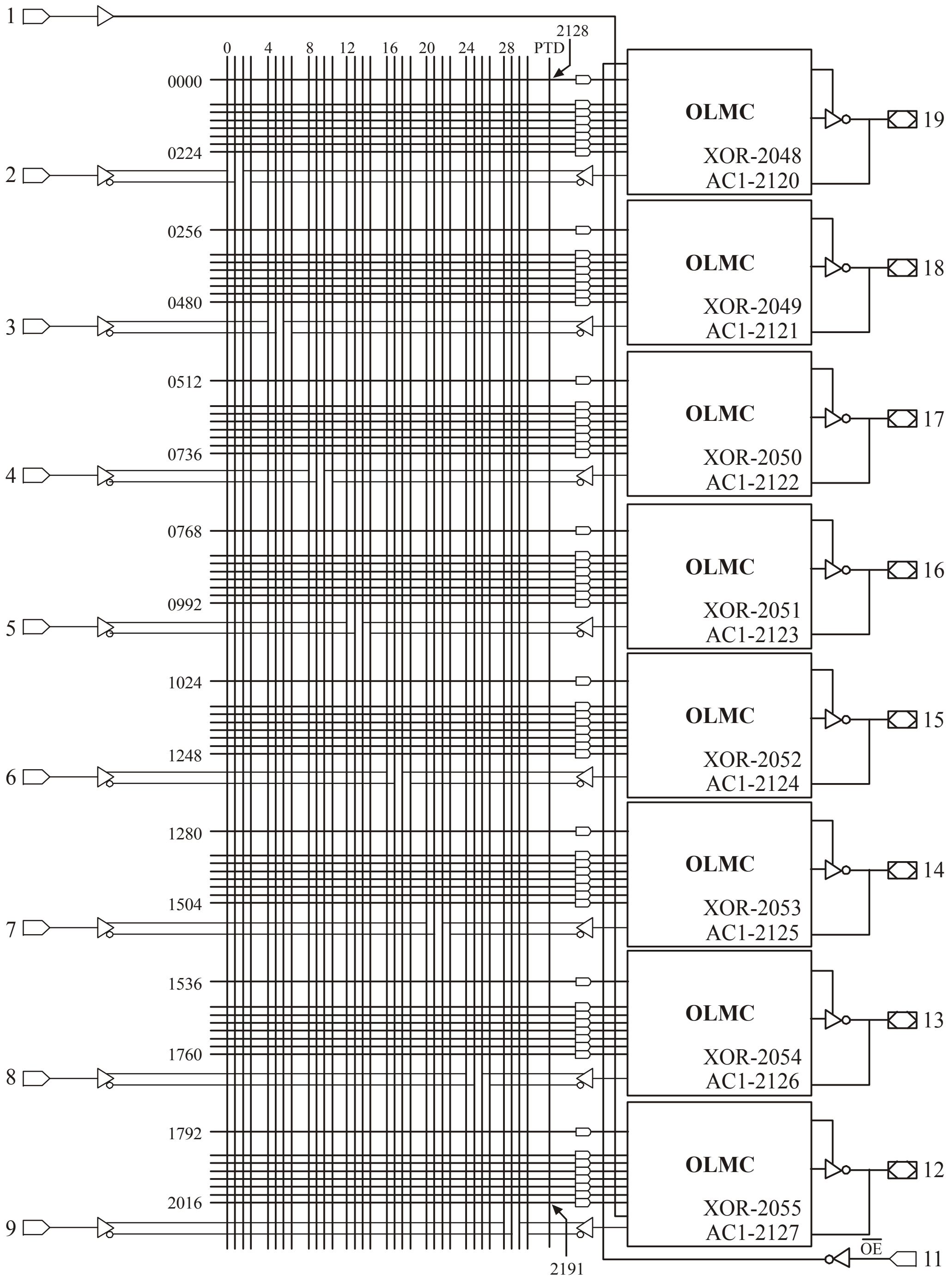


Fig. 12-36. Output programmable connections.

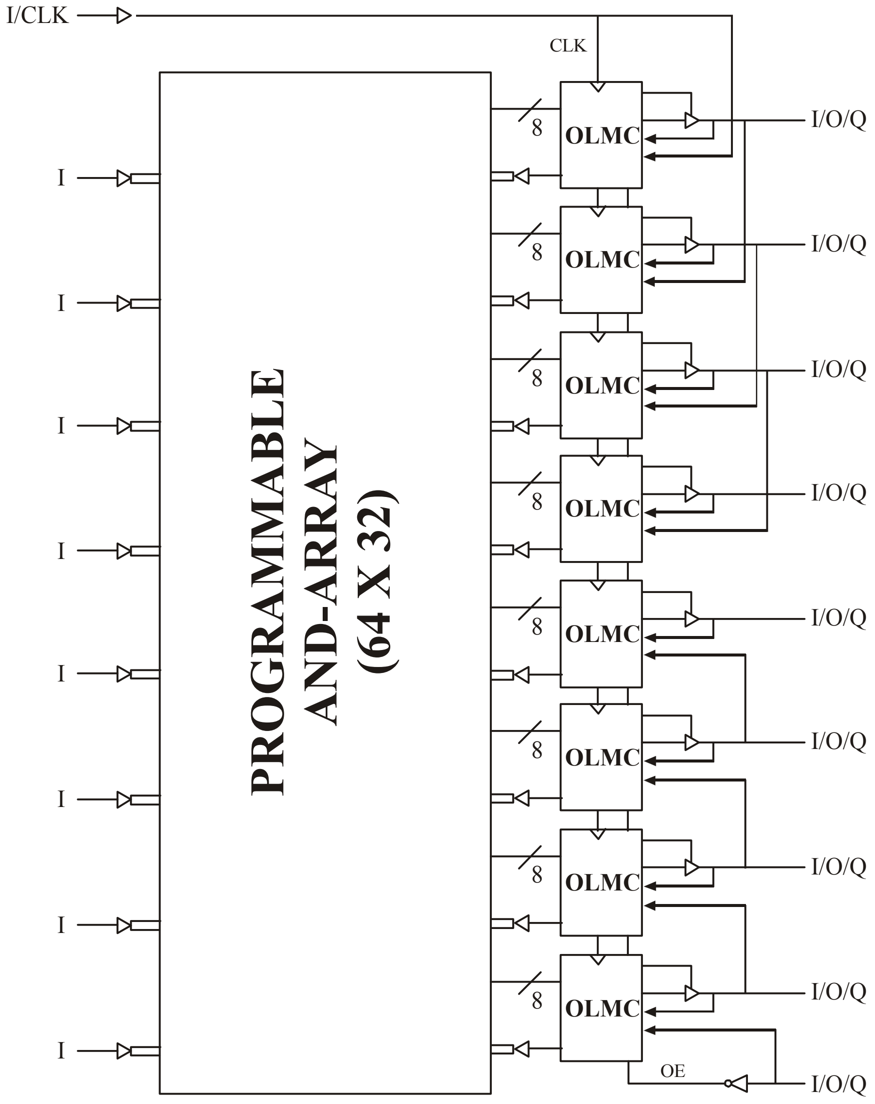


Fig. 12-37. Specification data sheet of GAL16V8 (Functional Block Diagram)

Lattice Semiconductor introduced the GAL technology. Basic details about the GAL16V8 are given in the datasheet shown in Fig. 12-37. For more details please refer to www.latticesemi.com.

GAL16V8 High Performance E^{2}CMOS PLD Generic Array Logic^{TM}

Features

- **HIGH PERFORMANCE E^{2}CMOS TECHNOLOGY**
  - — 3.5 ns Maximum Propagation Delay
  - — fmax = 250 MHz

— 3.0 ns Maximum propagation delay from Clock Input to Data Output

— UltraMOS* Advanced CMOS Technology

- **50% to 75% REDUCTION IN POWER FROM BIPOLAR**

— 75mA Typ. I_{CC} on Low Power Device

— 45mA Typ. I_{CC} on Quarter Power Device

- **ACTIVE PULL-UPS ON ALL PINS**

— E^2 CELL TECHNOLOGY

— Reconfigurable Logic

— Reprogrammable Cells

—100% Tested/100% Yields

— High Speed Electrical Erasure (<100ms)

— 20 Year Data Retention

•EIGHT OUTPUT LOGIC MACROCELLS

— Maximum Flexibility for Complex Logic Designs

— Programmable Output Polarity

— Also Emulates 20-pin PAL* Devices with Full Function/Fuse Map/Parametric Compatibility

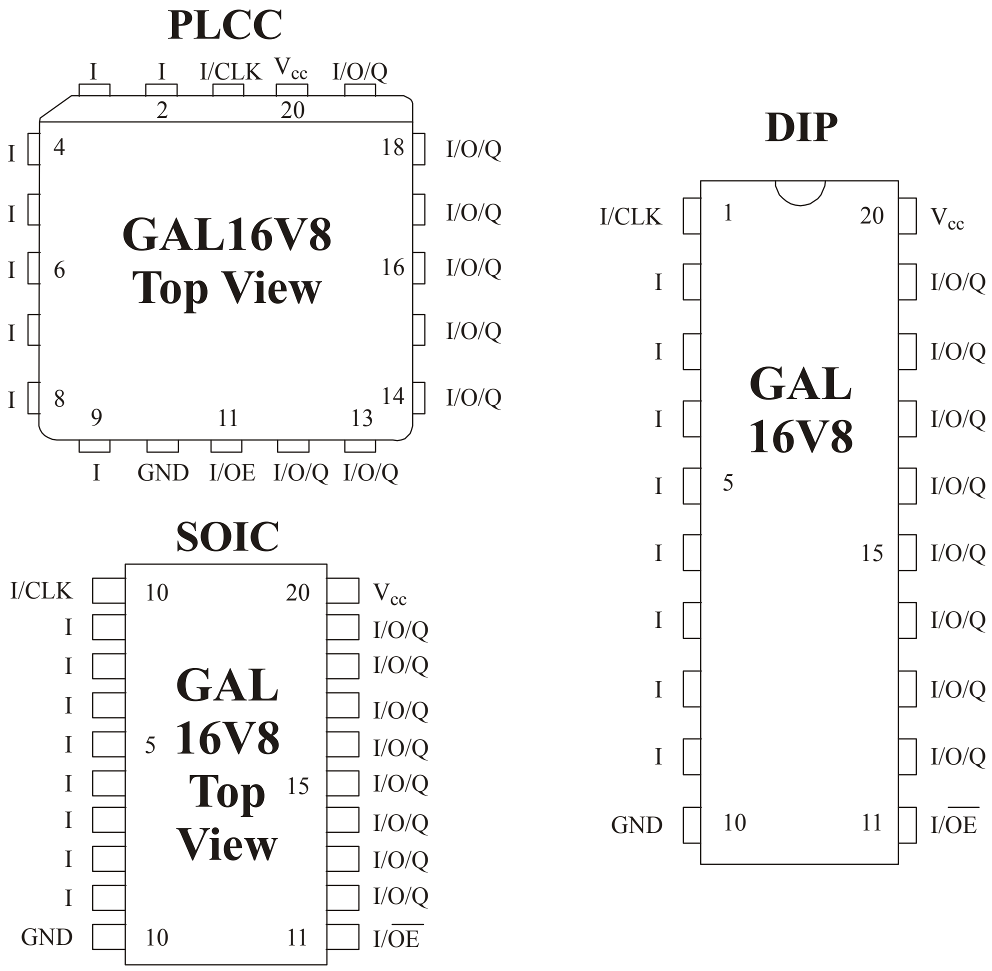


Fig. 12-38. Specification data sheet of GAL16V8 (Pin Configuration.)

- **PRELOAD AND POWER-ON RESET OF ALL REGISTERS**
  —100% Functional Testability
- **APPLICATIONS INCLUDE :**
  — DMA Control
  — State Machine Control
  — High Speed Graphics Processing
  — Standard Logic Speed Upgrade
- **ELECTRONIC SIGNATURE FOR IDENTIFICATION**
- **LEAD-FREE PACKAGE OPTIONS**

Description

The GAL16V8, at 3.5 ns maximum propagation delay time, combines a high performance CMOS process with Electrically Erasable (E^2) floating gate technology to provide the highest speed performance available in the PLD market. High speed erase times (<100 ms) allow the devices to be reprogrammed quickly and efficiently.

The generic architecture provides maximum design flexibility by allowing the Output Logic Macrocell (OLMC) to be configured by the user. An important subset of the many architecture configurations possible with the GAL16V8 are the PAL architectures listed in the table of the macrocell description section. GAL16V8 devices are capable of emulating any of these PAL architectures with full function/fuse map/parametric compatibility.

Unique test circuitry and reprogrammable cells allow complete AC, DC, and functional testing during manufacture. As a result, Lattice Semiconductor delivers 100% field programmability and functionality of all GAL products. In addition, 100 erase/write cycles and data retention in excess of 20 years are specified.

12-12. Complex Programmable Logic Devices (CPLD)

The PALs, PLAs and GALs are simple programmable logic devices. These PLDs have limited number of inputs, product terms and outputs. These can support up to 32 inputs and outputs. For more than 32 inputs either multiple SPLDs (Simple programmable logic devices) can be used as we discussed in article 12-10 about extending the input word length or more sophisticated type of chip (known as CPLDs) can be used.

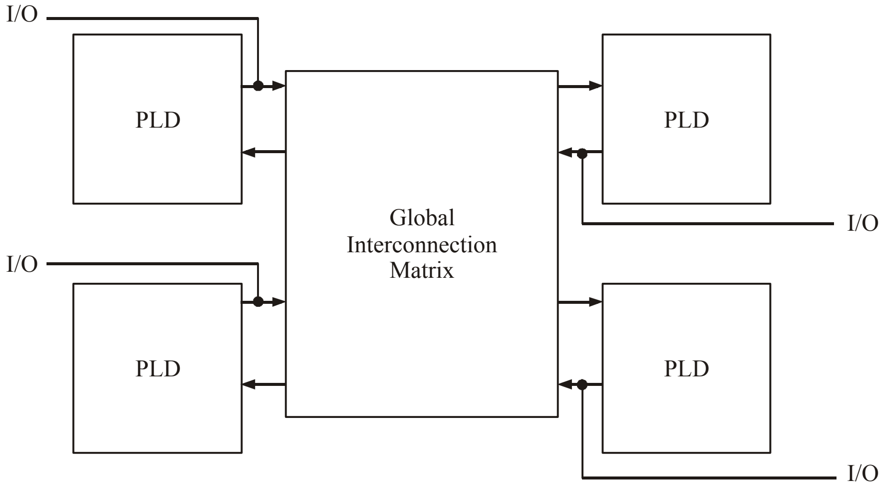


Fig. 12-39. Block diagram of CPLD.

A CPLD contains a bunch of PLD blocks whose inputs and outputs are connected together by a global interconnection matrix. Here a CPLD has two levels of programmability. Each PLD block can be programmed, and then the interconnections between the PLDs can be programmed. Fig. 12-39 shows the block diagram ofCPLD. Normally it consists of PAL-like blocks and the interconnections. A typical PAL-like block consists of 8 microcells. Each microcell consists of an AND-OR configuration, XOR gate, a flip-flop, a Mux and output buffer. Fig. 12-40 shows the internal architecture of PAL-like block. Typical PAL-like block consists of either 8 or 16 microcells.

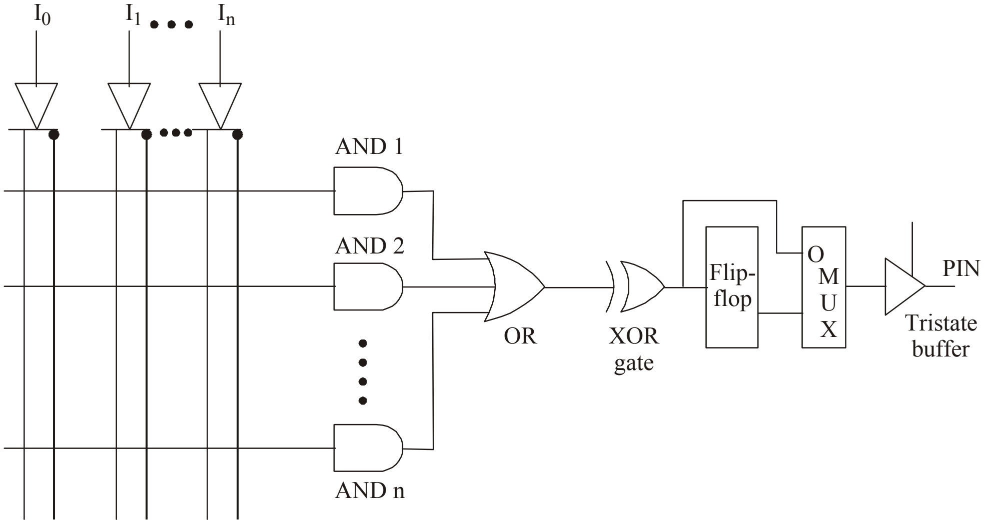


Fig. 12-40. Internal architecture of PAL-like block.

The AND-OR configuration in a PAL-like block usually consists of 5 to 20 inputs to AND and OR gates with 5-20 inputs. The XOR gate programs the inversion of the output function. This is achieved by programming the fused input by programing the PLA. This is discussed in article 12-15. The MUX (multiplexes) selects either the output from flip-flop or the output from the XOR gate. The output buffer acts as a switch that decides the chip's pin to act as an output or an input.

CPLDs are made on the following three technologies. (1) EPROM, (2) EEPROM and (3) FLASH. Normally EPROM based CPLDs are one-time programmable if these devices are not capable of ultra-violet erasable. EEPROM and FLASH based devices are capable of re-programming number of times by erasing the contents. Not all the EEPROM and FLASH devices are capable of re-programming, once these devices are soldered. Some special on-chip programming logic is required to re-program them if those devices are fixed.

Some of the commercially available CPLDs are XPLA (manufactured by Philips), X09500 (manufactured by Xilinx) and ATF, ATV (manufactured by Atnel). A CPLDs provide the capacity up to equivalent of 50 typical SPLD devices.

12-13. Application of CPLDs

As CPLDs offer high speed and range of different capacities, they are very useful for many applications, from implementing a small design to prototyping of small gate arrays. One of the most common uses of CPLDs in the industry is the conversion of designs that consist of multiple SPLDs into a smaller number of CPLDs. Because of these kinds of applications, market for CPLDs is growing steeply.

LAN Controllers, UART and many other network devices use CPLDs heavily. A main advantage of CPLDs is that they provide simple design changes through re-programming. All commercial CPLD products are re-programmable. It is even possible to re-configure hardware without power down.

12-14. Packaging of CPLD Chip

The CPLDs have a large number of pins which make it impossible to pack in DIP (Dual in line packaging) mode. Following methods are most commonly used for packaging purposes.

Plastic Leaded Chip Carrier : A plastic leaded chip package (PLCC) can be used for maximum of 84 pins. It has pins on all the four sides. All the pins are wrapped down near the edge as shown in Fig. 12-41(*a*). An IC socket is used to hold the chip.

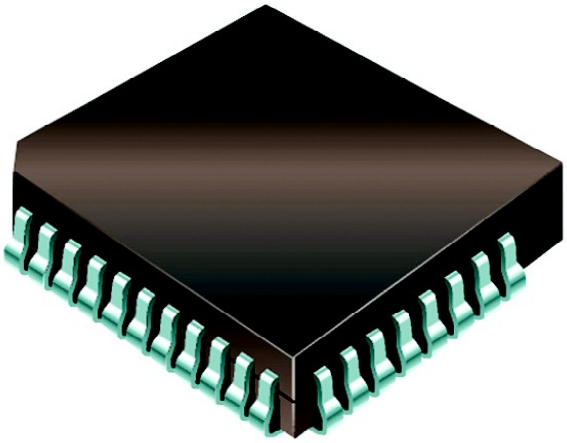


Fig. 12-41(*a*). PLCC Package.

Quarter Flat Package : This package can be used for high pin count (up to 300). It has pins on all four sides like PLCC but pins extending outward from the package downward curve shape. The pin-inter-space is less compared to PLCC. It is shown in Fig. 12-41(*b*). Its pins are very thinner than PLCC.

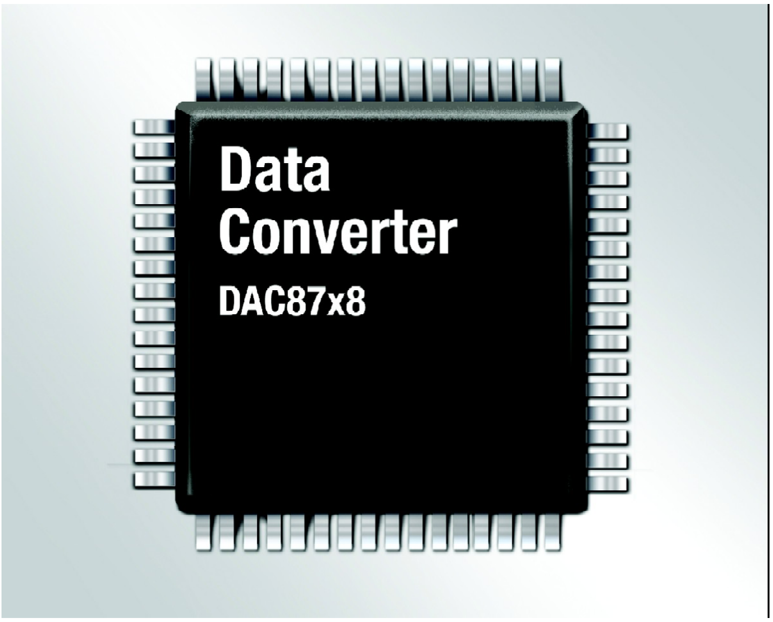


Fig. 12-41(*b*). QF package.

Ceramic Pin Grid Array (CPGA) : It has pins on all the four sides extending downward straight in grid pattern from bottom as shown in Fig. 12-41(*c*). It can be used for few hundred pins.

Ball Grid Array (BGA) : It has pins on all the four sides extended downward straight from bottom. They are small in size and it requires small solder balls to connect to the PCB. It can be used in the chips having more than 300 pins. Fig. 12-41(*d*) shows the one type of BGA package.

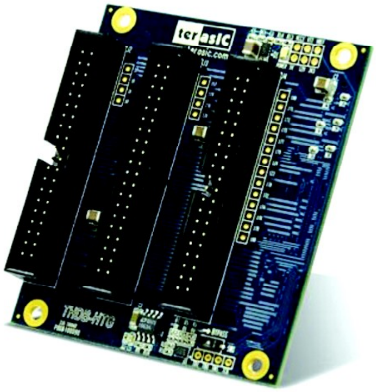


Fig. 12-41(*c*). FPGA package.

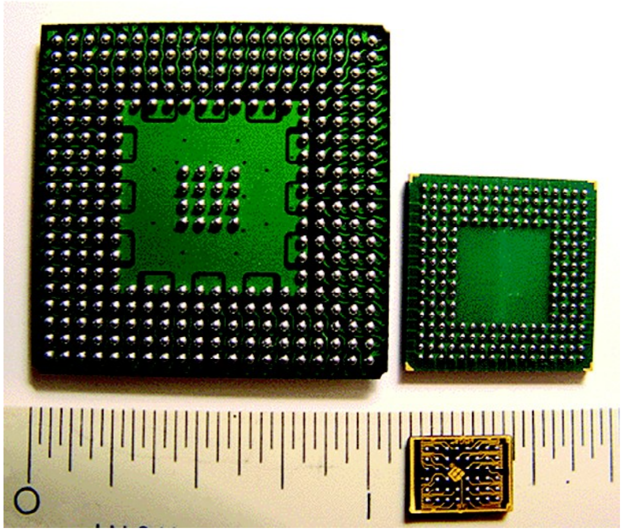


Fig. 12-41(*d*). BGA package.

12-15. Programming of PLDs

Normally in PLDs the programming will be done in two ways. For normal PROM, PAL and SPLDs, a programmer will be used to program (*i.e.*, to blow out the fuses). For this method the designer develops a fusemap which has the information of fusemaps to blow out. This set of information is done on the PC. A programmer is a general purpose machine that can program the PLDs. This program is attached to the PC via a cable as shown in Fig. 12-42. Using a dedicated software, the designer will enter the fusemap details to the PLD, which is connected to the programmer. The program or the fusemap details contains the necessary logic diagrams and the pin-connection diagrams.

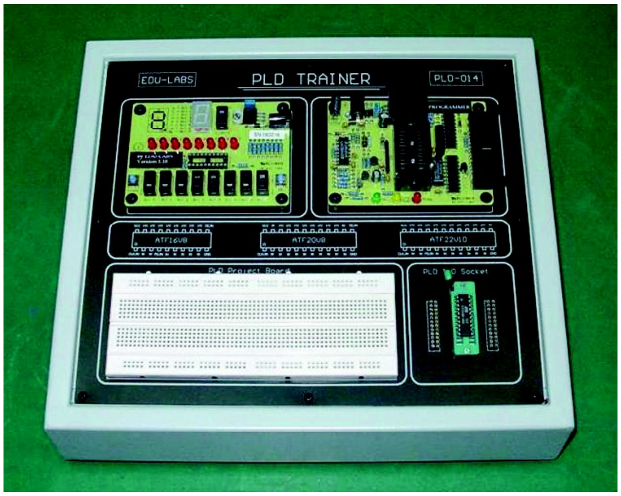


Fig. 12-42. Programming of PLDs.

This kind of programming is not suitable for CPLDs. Because CPLDs are very complex ICs which are having more number of logics than SPLDs and more number of pins. Sometimes the total number of pins exceeds 300. So for this CPLDs, a special technology called 'In system programming' (or ISP) is used. In this kind of programming, the chip will be placed in the actual circuit itself. These kind of programming is done via a JTAG cable which is connected to the PC. In the circuit board, CPLDs are fixed over the sockets. So we can program any number of CPLDs from the board by same fusemap details. This is illustrated in Fig. 12-43. The flow chart for programming the PLD is shown in Fig. 12-44.

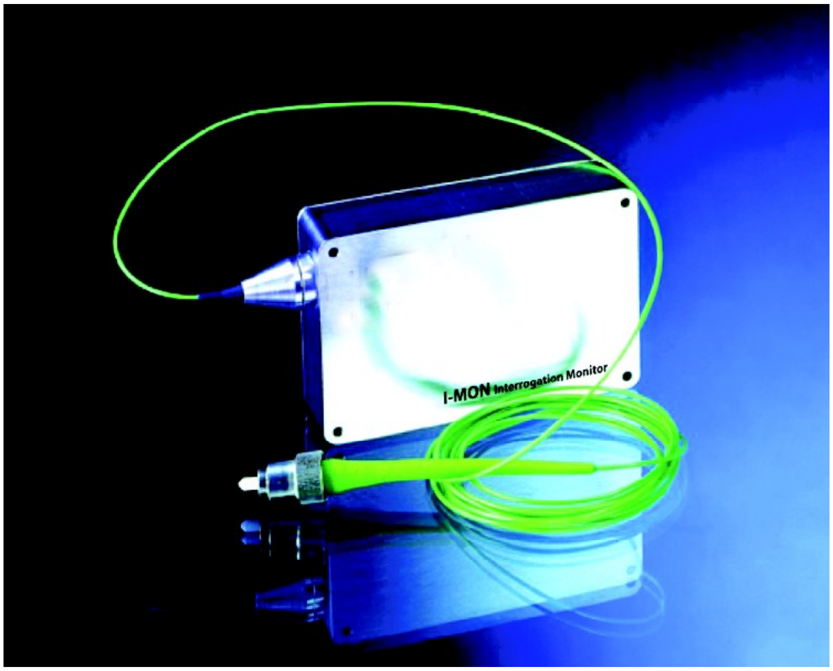


Fig. 12-43. Programming of CPLD.

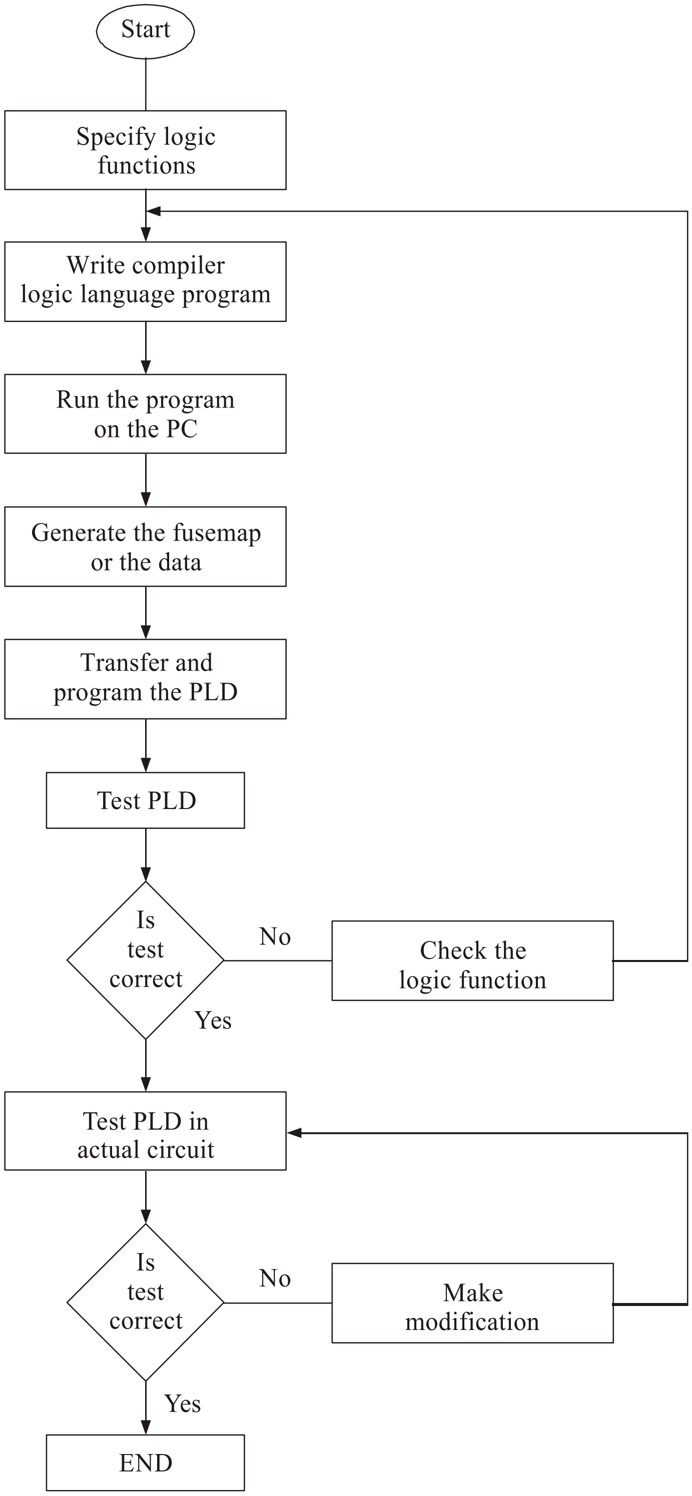

Start
Specify logic functions
Write compiler logic language program
Run the program on the PC
Generate the fusemap or the data
Transfer and program the PLD
Test PLD
Is test correct
No
Check the logic function
Yes
Test PLD in actual circuit
Is test correct
No
Make modification
Yes
END

Fig. 12-44. Flow chart for PLD programming.

12-16. Field-Programmable Gate Array (FPGA)

A field-programmable gate array (FPGA) is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions. In most FPGAs, these programmable logic components or logic blocks also include memory elements, which may be simple flip-flops to complete block of memories.

The FPGA consists of 3 main structures :

1. Programmable logic structure,
2. Programmable routing structure, and
3. Programmable Input/Output.

1. Programmable logic structure : The programmable logic structure FPGA consists of a 2-dimensional array of configurable logic blocks (CLBs). Each CLB can be programmed to implement any Boolean function of its input variables. Typically CLBs have between 4 and 6 input variables. Functions of larger number of variables are implemented using more than one CLB. Fig. 12-45 shows architecture of FPGA and its CLBs.

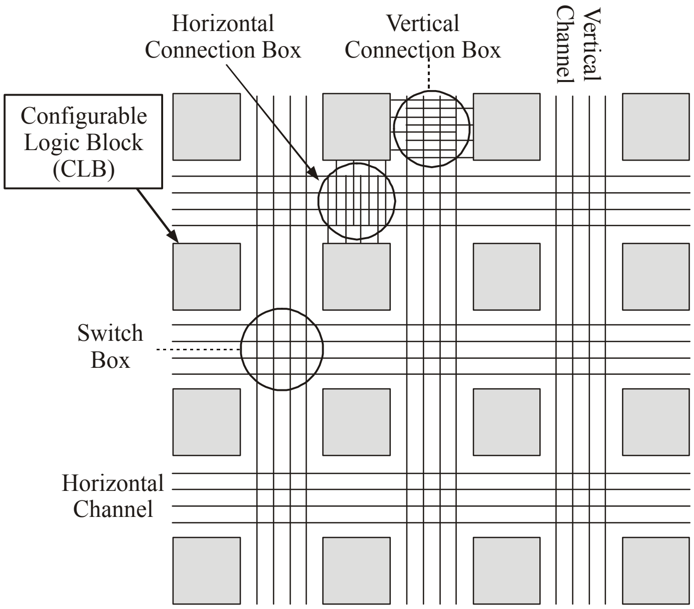


Fig. 12-45. Structure of FPGAs and CLBs.

Each CLB contains 1 or 2 flip-flops to allow implementation of sequential logic and a 4-input look up table (LUT). Large designs are partitioned and mapped to number of CLBs with each CLB programmed to perform a particular function. Fig. 12-46 shows a typical block diagram of a CLB.

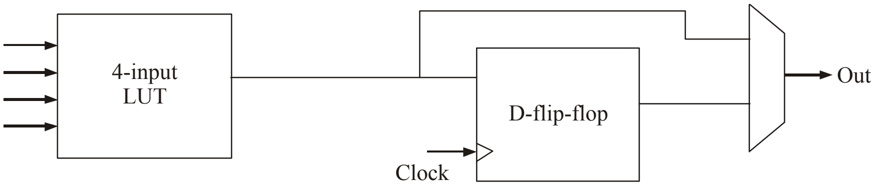


Fig. 12-46. Typical block diagram of a CLB.

Those CLBs are then connected together to fully implement the target design. Connecting the CLBs is done using the FPGA programmable routing structure.

2. Programmable routing structure : To allow for flexible interconnection of CLBs, FPGAs have 3 programmable routing resources :

(*i*) Vertical and horizontal routing channels which consist of different length wires that can be connected together if needed. These channels run vertically and horizontally between columns and rows of CLBs. This is shown in Fig. 12-47.

(*ii*) Connection boxes, which are a set of programmable links that can connect input and output pins of the CLBs to wires of the vertical or horizontal routing channels.

(*iii*) Switch boxes, located at the intersection of the vertical and horizontal channels. These are a set of programmable links that can connect wire segments in the horizontal and vertical channels. When a wire enters a switch box, there are three programmable switches that allow it to connect to hree other wires in adjacent channels. The pattern, or topology, of switches used in this architecture is the planer or domain based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. Fig. 12-48 shows connections in a switch box.

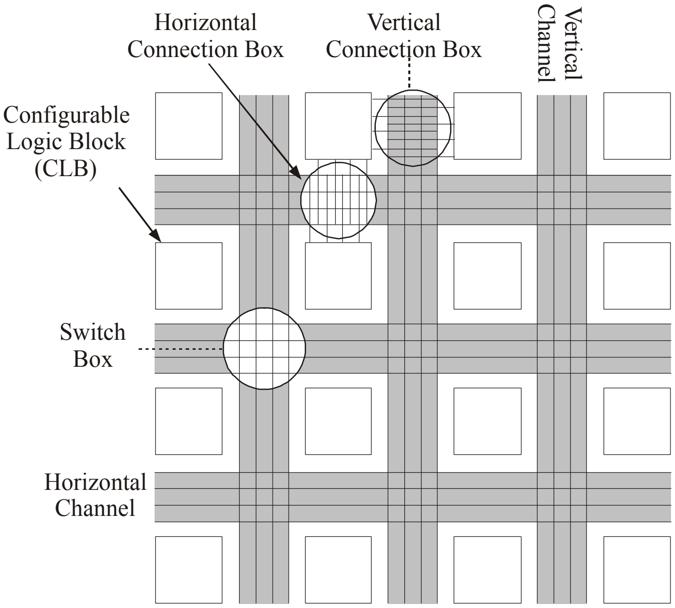


Fig. 12-47. Vertical and horizontal routing channels of FPGA.

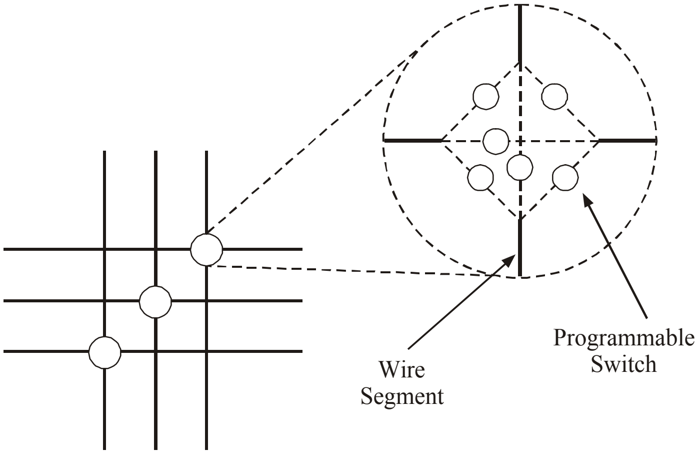


Fig. 12-48. Switch box topology.

3. Programmable input/output. These are mainly buffers that can be configured either as input buffers, output buffers or input/output buffers. They allow the pins of FPGA chip to function either as input pins, output pins or input/output pins. It is illustrated in Fig. 12-49.

Two of the popular FPGA devices are Logic cell array by Xilinx and the multiplexer based cell by ACTEL. We will discuss about these two types in the following pages.

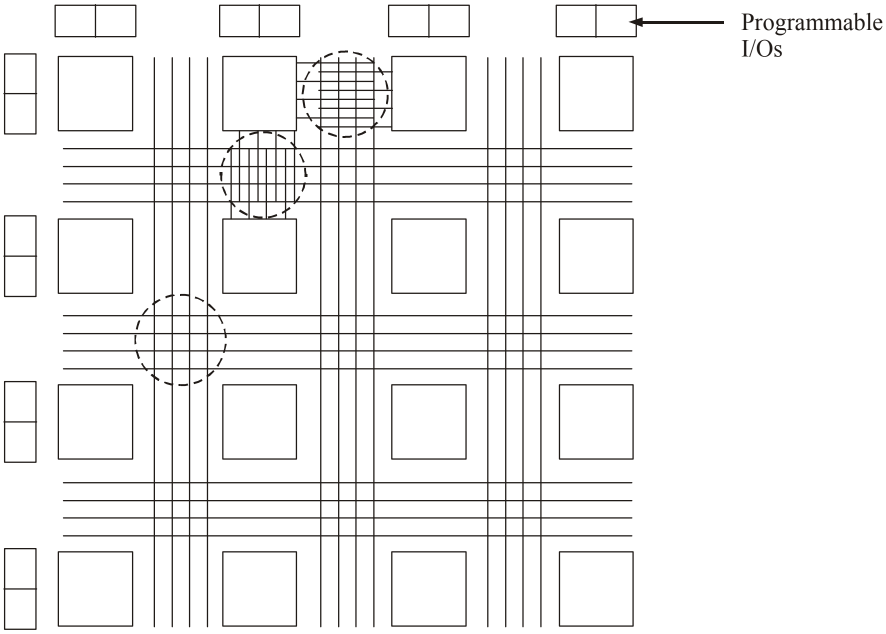


Fig. 12-49. Programmable I/O for FPGAs.

12-17. Logic Cell Array

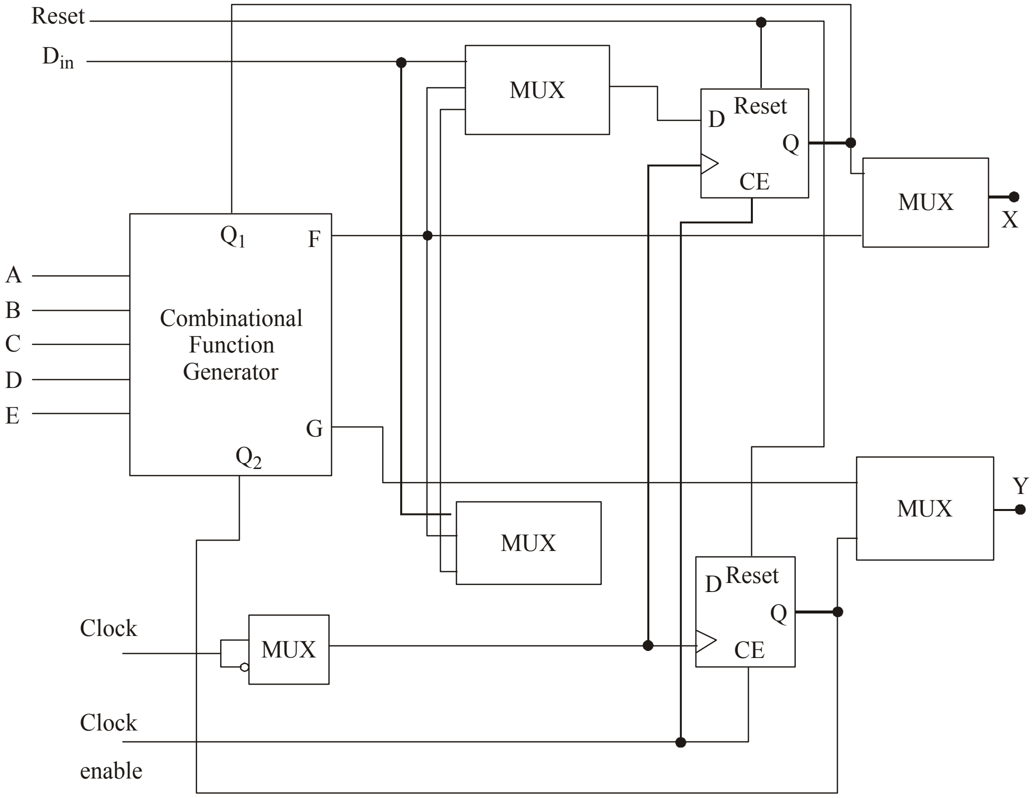


Fig. 12-50. Configurable Logic block.

It is the first FPGA developed by Xilinx in the year 1992. It is an array of configurable logic blocks (CLBs) and connected by a network of programmable interconnections. The circuit interface is controlled by Input/output interface blocks (IOBs). When compared to all the other PLDs, each CLB in the logic cell array is a complex one. Generally it is composed of two logic blocks that can implement any function that uses up to five variables, a set of multiplexer, two D-flip-flops and some feedback loops. It is shown in Fig. 12-50.

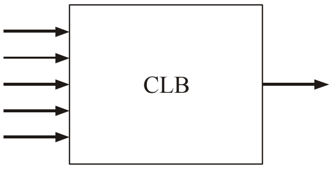


Fig. 12-51(*a*). 5-input CLB.

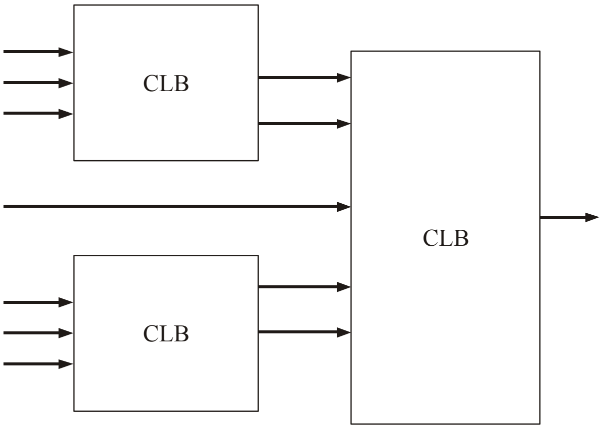


Fig. 12-51(*b*). 7-input CLB.

One of the main advantages of Xilinx devices are reconfigurable. So the interconnections and the logic functions implemented by the CLBs can be programmed as per the design requirements. Sometimes the CLBs are capable of having more than 5 inputs. Some of the block diagrams of CLBs with multiple inputs are shown in Fig. 12-51 (*a*), (*b*) and (*c*).

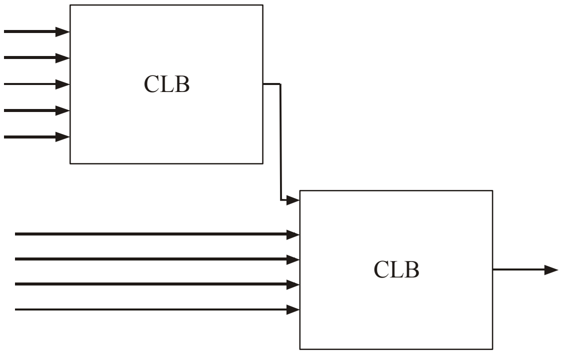


Fig. 12-51(*c*). 9-input CLB.

The programmable interconnection channel gives the necessary routing paths to connect I/O blocks and CLBs for different configurations. This is illustrated in the Fig. 12-52. Following are the possible interconnections namely.

- Direct Connection
- General Connection
- Switching Matrix
- Horizontal/Vertical Long Lines

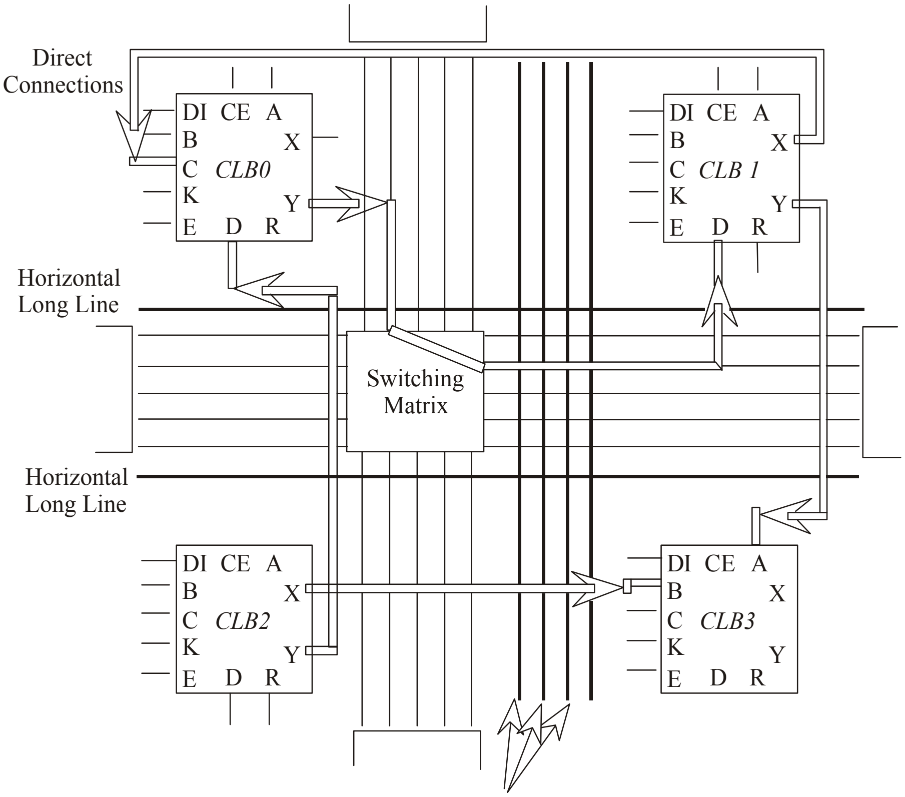


Fig. 12-52. Programmable interconnection.

The general purpose lines are the horizontal and vertical lines. These lines run between rows and columns of CLBs and I/O blocks. At the intersection of these lines, a switching matrix is provided that allows the interconnection of adjacent row and column. Horizontal/vertical long lines run across the device passing through the switching matrix. These lines travel across the device for longer distance. Direct connections are the connections between the adjacent CLBs. These lines provide a minimum propagation delays for the signals. Some of the commonly available Xilinx FPGAs are XC2000, XC3000, XC3100, XC4000 and XC5200. For more information regarding their technical details please refer to *www.xilinx.com.*

12-18. Actel Devices Architecture

Usually Actel FPGA architecture is similar to that of conventional gate array. These devices combine some of the flexibilities of mask-programmable gate and the convenience of field programmable devices. It consists set of rows of logic blocks that are separated by a routing channel. The interconnections between the logic blocks are done by programming the fuses which are connecting the predifused horizontal and vertical wire segments. Fig. 12-53 shows the architecture of the actel device.

The fuse is a special structure that gives a high impedance in its original state. By applying a high voltage to it, we can turn ON. By this we know that programming of these devices is different from that of PROMs, PLDs and LCA. The fuse is relatively small and has low resistance in the conducting state (≃ 500 Ω). Programming of these devices is called antifuse. The antifuse is a two terminal structure, having poly silicon layer on top, N+ doped silicon on bottom and an oxide-nitride oxide dielectric layer in between. The size of the antifuse is of about 1.8 mm^2 and it is one time programmable.

There are two types of antifuses, namely cross fuses and horizontal fuses. The cross fuses control the interconnection of vertical and horizontal segments. Horizontal fuses control the interconnection of horizontal segments. Logic block, the main part of the actel device, is shown in Fig. 12-54.

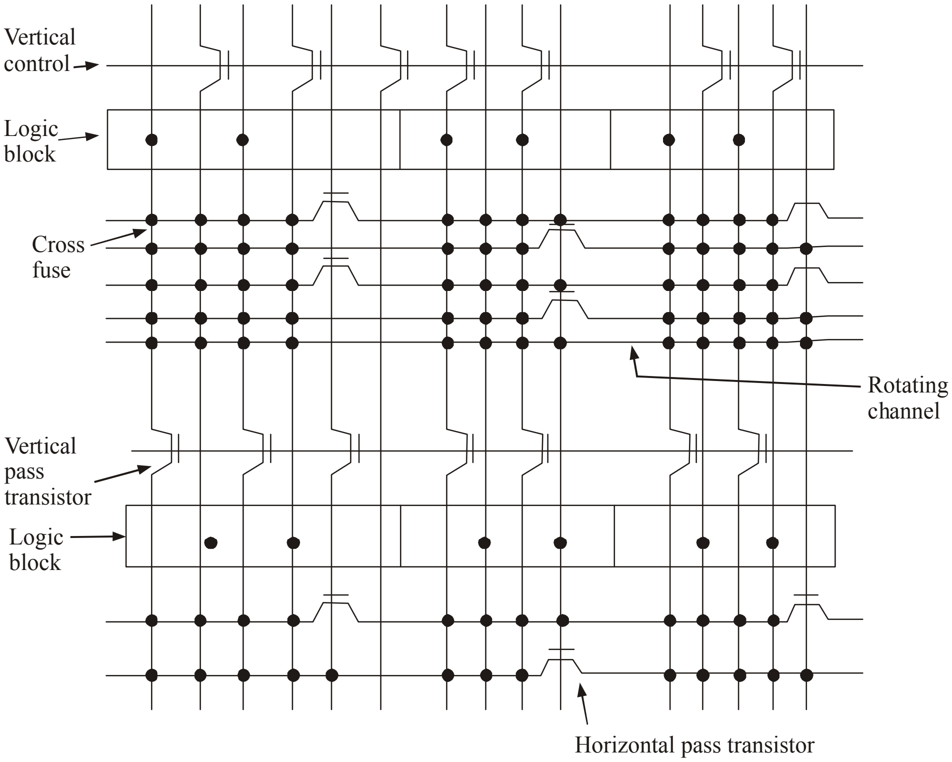


Fig. 12-53. ACTEL device structure.

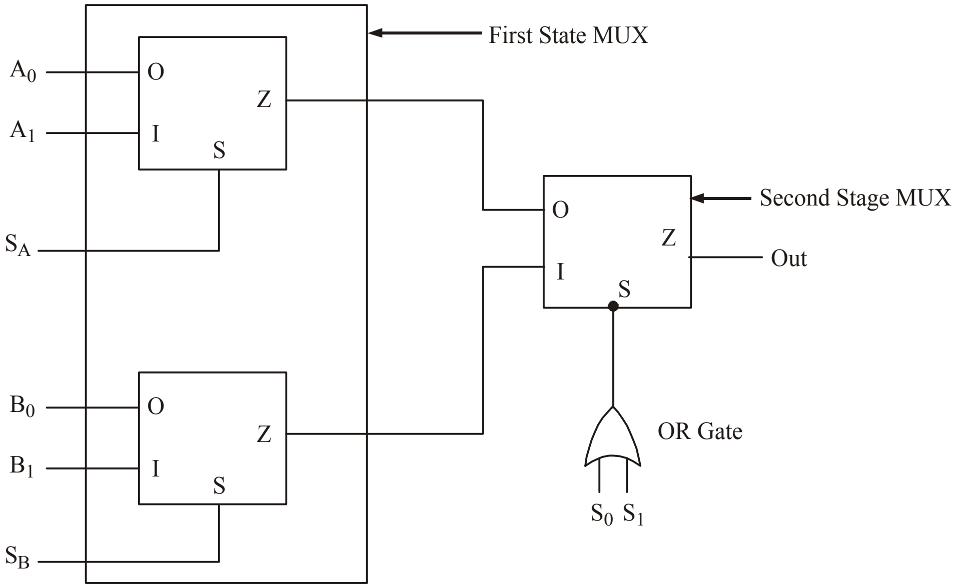


Fig. 12-54. Logic block architecture.

It consists of three numbers of two-to-one multiplexer. The inputs to the first stage of multiplexer and the control signals can be programmed by the user. The control signals which are used in the second stage multiplexer is driven by an OR gate. This will enable the flexibility of the logic block module. By selecting the correct inputs, the users can program the necessary logic functions. The logic blocks operate with inputs ranging from two, three and up to 8.

Some of the main product families of Actel are Actel IGLOO FPGAs, Actel Fusion mixed signal FPGAs, ProASIC 3 and RTAX FPGAs. For more details about this product details please refer www.actel.com.

12-19. Limitation of ACTEL Device

The ACTEL chip has a limitations in regard to the route structure. The fuse resistance together with the capacitance of the segment forms a RC tree. To avoid this kind of behaviour, we have to delay the fuses of that particular route and thus the inputs are limited to three. One more restriction is that the input to the logic blocks are accessible either from the top of the channel or from the bottom of channel but not from both of them.

These restrictions are removed by using a flexible cell where most logic functions are implemented by many ways using very distinct inputs.

12-20. Advantages and Disadvantages of Antifuse Programming

Although the antifuse programming used in Actel devices has many advantages and disadvantages, following are important from the subject point of view.

Advantages

1. Small in size about 1.8 mm^2.
2. Low resistance and low capacitance. This makes the device faster.

Disadvantages

1. One time programmability.
2. A special hardware is required for programming.

3. An antifuse FPLD contains 75,000 fuses. But only 2% of them should be programmed.
4. 5 to 10 minutes are required to program each chip.

12-21. Applications of FPGA

FPGAs are very much used in many fields and applications. Digital signal processing, aerospace, defence, radio controls, speech recognition, medical imaging, bioinformatics, cryptography, high speed computer vision, device controllers, encoding and filtering are some of the main applications in which FPGA is playing a critical role.

Some of the interesting applications of FPGA are prototyping of designs and emulation of entire large hardware systems. Here many FPGAs are connected by interconnection and they are used for the hardware emulation.

FPGAs are used in high performance computing applications where FFT or convolution functions are required frequently.

12-22. Advantages of CPLDs over FPGAs

Even though FPGA is the latest invention than the CPLD, CPLD has some advantages over FPGA. Some of the advantages are listed below.

1. The architecture and timing in the CPLD devices are predictable. But in FPGA these are unpredictable.
2. The compilation and programming time is shorter for CPLDs compared to FPGAs.
3. In-system performance of CPLDs is faster than FPGAs.
4. CPLDs have continuous interconnect style whereas FPGAs have segmented interconnect style.
5. CPLDs can be used for combinational and registered logic applications whereas FPGAs can be used only for the registered logic.
6. CPLDs are manufactured using one of the three process technologies — EPROM, EEPROM or Flash. They do not require any external PROM. But FPGAs use the SRAM technology and require some form of external configuration memory source (PROM).

12-23. Advantages of FPGAs over CPLDs

There are many advantages of FPGA over CPLD. But some of the main advantages are described below.

1. CPLDs have the limited programmability. But FPGAs provide unlimited re-programming feature.
2. Both CPLDs and FPGAs have architectural differences. A CPLD structure is having less flexibility due to one or more programmable sum of products logic arrays that feed to small number of clocked registers. But FPGAs are done by interconnect mechanism. This gives a very flexible to design a complex problem.
3. Most of the FPGAs are having high level of embedded functions such as adders and multipliers and memories for the operations. So in FPGAs the designs can be made on the fly whenever and wherever required.

12-24. Design of FPGAs and Programming

Normally the FPGAs are designed by a hardware description language (HDL) or a Schematic design. VHDL and Verilog are some of the HDL languages that are used to design the FPGA. After the design, using the electronic design automation tool, a netlist is created. This netlist actually carries all the connection and routing details such as pin details and its connections. Then this netlist will be fitted to the actual FPGA structure using a process called place-and-route. This is done by the proprietary software given by FPGA chip development companies. Here the designer validates the map, connection details and connection route through timing analysis and simulation. If there is any modification required, then the designer can re-enter the necessary changes and create a new netlist. Earlier assembly language was used in programming of the design in HDLs. But companies such as Cadence, Synopsys and Celoxica are promoting high level language called System C for speedy design of the FPGAs. Mentor graphics and Impulse accelerated technologies use C++ for their development needs. Annapolis Micro Systems and National Instruments use graphical data flow approach for the FPGA design.

Nowadays complex designs and functions are tested and optimized for future uses. These predefined libraries are called IP cores. So for our design requirements, functions and logics from this IP core can be used. Nowadays using these libraries, FPGA are designed very quickly. Researches are going on in so many technology companies for sophisticated design of FPGAs, since the demand for FPGAs is rapidly growing year after year.

SUMMARY

In this chapter, you have learnt that,

1. PROM has fixed AND gate array and programmable OR array.
2. PAL has programmable AND gate array and fixed OR array.
3. PLA has programmable AND gate array and programmable OR array.
4. Generic array logic (GAL) devices use EEPROM array.
5. PLDs, PALs and PLAs are simple programmable logic devices and are not suitable for more than 32 inputs.
6. Programming of PAL devices is easier than PROM.
7. The main feature of GALs is the programmable output logic macrocell.
8. A PAL-like block consists of about 16 microcells.
9. Many PLDs have a polarity fuse so that the final output can be inverted if desired.
10. Programming a PLD is basically the process of blowing some fuses.
11. Programming of PLA is more difficult to program than PAL and PROM.
12. FPGA architecture is a large cell array.
13. FPGA use SRAM technology, whereas CPLD use EPROM, EEPROM or Flash.

GLOSSARY

PLD : An electronic device used to design reconfigurable digital circuits.

PAL : A programmable array logic that has a programmable AND gate array followed by fixed OR gate array.

PLA : A programmable logic array that has a programmable AND gate array and a programmable OR gate array.

SPLD : It refers to any type of simple PLD, either a PLA or a PAL.

CPLD : A more complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip.

FPGA : A field programmable gate array featuring a general structure that allows very high logic capacity.

Interconnect : The wiring resources in a field programmable device.

Antifuse : A programmable switch invented by ACTEL, known also as PLICE (programmable low impedance circuit element).

GAL : Generic array logic is also one type of configurable PAL.

Macrocell : The basic logic cell used in large PLDs to provide many logic functions such as combinational, registered and simple.

Logic block : A relatively small circuit block that is replicated in an array in field programmable device.

Logic Capacity : The amount of digital logic that can be mapped into a single field programmable device.

Polarity fuse : A programmable element used to invert the output logic.

OLMC : Output logic macrocell which is used to design logic circuit by configuring them.

LCA : A logic cell array developed by Xilinx.

Crossfuse : A programmable element which controls the interconnection of vertical and horizontal segment.

DESCRIPTIVE QUESTIONS

1. What is a PLD?

2. Write a short note on the applications of programmable logic success.

3. What does a × (cross) and • (dot) represent in PLD program?

4. What are the advantages of programmable logic devices?

5. Write a short note on symbolic representation of PLDs.

6. Explain programming of PROM PLDs using logic diagram.

7. How are the programmable logic devices classified?

8. Explain the architecture of PAL. How it differs from PROM.

9. Draw the internal architecture of TI FPLA 84O. Explain the function of each component.

10. How Word Length of PLA can be extended?

11. What is the function of polarity fuse in PLDs?

12. Explain the working of FPGA with a neat diagram.

13. What are the advantages of FPGAs over CPLDs?

14. Write a short note on application of FPGA and CPLD.

15. What is the role of antifuse in Actel devices? Explain the limitation of Actel FPGA.

16. Draw the block diagram of a PLA device and briefly explain each block.
(*Anna University, Apr. /May 2008*)

17. Write short notes on field programmable gate array (FPGA).
(*Anna University, Apr. /May 2008*)

18. Explain erasable programmable read only memory.
(*Anna University, Apr. /May 2008*)

19. What is a combinational PLD?
(*Anna University, Nov. /Dec. 2007*)

20. What is a programmable LOGIC Array (PLA)? Describe with a logic diagram the principle of operation of a PLA. What are its advantages?
(*VTU, Jul. 2006*)

21. Drive the PLA implementation of a full adder and explain.
(*Mahatma Gandhi University, May 2005*)

22. How does the PLA differ from ROM from the point of view of functionality?

(*Mahatma Gandhi University, May 2005*)

23. Explain the organization of PLA.

(*Mahatma Gandhi University, Nov. 2005*)

24. Explain the different structure and application of PLAs.

(*Mahatma Gandhi University, Jan. 2007*)

25. Draw and explain the programmable logic array.

(*GBTU/MTU, 2009-10*)

26. Explain the structure of programmable array logic (PAL). Realize the full adder circuit using the PAL.

(*GBTU/MTU, 2004, 2005*)

27. Write the short note on programmable logic array (PLA). Also describe its various implement forms. Give example in support of your explanation.

(*GBTU/MTU, 2004*)

28. What is PLA ? Explain. How it capacity is specified ?

(*RGTU., June 2011*)

29. With the help of block diagram explain the working of PLA's and compare it with ROM.

(*RGTU., Dec. 2009*)

30. Design a binary to gray code converters using PLA.

(*RGTU., Dec. 2009*)

31. Describe the differences between the PLA and PAL.

(*GBTU., 2010-11*)

32. Explain the working of PAL.

(*RGTU., June 2009*)

33. Explain the architecture and working of field programmable gate array.

(*RGTU., June 2010*)

TUTORIAL PROBLEMS

1. Use a 16 × 8 PROM PLD to form the following outputs with minterm as

$$f_1 = \Sigma m\,(0, 2, 4, 6)$$
$$f_2 = \Sigma m\,(1, 3, 7, 10, 12)$$
$$f_3 = \Sigma m\,(5, 13, 15)$$

2. Design a 4-input, 5-output combinational circuit using a PLA device. The following are the outputs :

$$f_1 = \Sigma m\,(0, 3, 5, 9, 11, 13)$$
$$f_2 = \Sigma m\,(2, 5, 7, 14)$$
$$f_3 = \Sigma m\,(1, 2, 8, 12)$$
$$f_4 = \Sigma m\,(0, 4, 6, 10)$$
$$f_5 = \Sigma m\,(1, 2, 5, 8)$$

3. Draw the logic diagram of PAL to yield the following outputs.

$$f_1 = \overline{\mathrm{A}} + \mathrm{BC}$$

$$f_2 = \overline{\mathrm{A}}\,\overline{\mathrm{B}}\,\overline{\mathrm{C}} + \mathrm{ABC}$$

$$f_3 = \overline{\mathrm{A}}\mathrm{CD} + \overline{\mathrm{A}}\mathrm{B}$$

$$f_4 = \overline{\mathrm{A}}\mathrm{B} + \overline{\mathrm{C}}\mathrm{D}$$

4. A PROM is used to implement the following function.

$$f_1 = \mathrm{ABC}\overline{\mathrm{D}} + \mathrm{A}\overline{\mathrm{B}}\,\overline{\mathrm{C}}\mathrm{D}$$

$$f_2 = (\mathrm{ABC})\,(\mathrm{A} + \mathrm{B} + \mathrm{C} + \mathrm{D})$$

(*a*) What is the minimum size of PROM required?

(*b*) Determine data in each location of PROM.

MULTIPLE CHOICE QUESTIONS

1. A PLA can be used

(*a*) as a microprocessor (*b*) as a dynamic memory

(*c*) to realize a sequential logic (*d*) to realize a combinational logic

(*Gate-1994*)

2. Choose the correct statement from the following :

(*a*) PROM contains a programmable AND array and a fixed OR array.

(*b*) PLA contains a fixed AND array and a programmable OR array.

(*c*) PROM contains a fixed AND array and a programmable OR array.

(*d*) PAL contains a programmable AND array and a programmable OR array.

(*Gate-1992*)

3. A PAL has 10 inputs, the number of the inputs to each of the AND gate is

(*a*) 5 (*b*) 10 (*c*) 20 (*d*) 40

4. The IC package used for ICs with more than 250 pins is

(*a*) DIP (*b*) PLCC (*c*) QFD (*d*) BGA

5. The typical resistance of antifuse is in

(*a*) kilo ohm (*b*) Mega ohm (*c*) Ohm (*d*) Micro ohm

6. The programming scheme for Xilinx Logic Cell Array (LCA) is based on

(*a*) EPROM technology(*b*) Fuses technology

(*c*) DRAM technology (*d*) SRAM technology

7. Which one of the following is not a SPLD?

(*a*) PLA (*b*) PAL (*c*) PROM (*d*) LCA

8. Which one of the following statements is correct?

(*a*) The architecture timing of CPLD is predictable whereas in case of FPGA, it is unpredictable.

(*b*) Complexity of logic function is more in CPLD than FPGA.

(*c*) In system performance of FPGA is fast than CPLD.

(*d*) FPGA has limited programmability whereas CPLD has relatively unlimited programmability.

9. The use of X-OR gate in PALs is to implement

(*a*) high driving capability (*b*) inversion of input

(*c*) inversion of output logic (*d*) Not used in PALs

10. The GAL16V8 replaced most — pin PAL devices.

(*a*) 20 (*b*) 24 (*c*) 16 (*d*) 8

11. The GAL 22V10 has

(*a*) 10 inputs and 22 outputs (*b*) 22 inputs and 10 outputs

(*c*) 11 inputs and 10 inputs/outputs (*d*) 11 inputs and 10 outputs

12. The GAL16V8 has

(*a*) 8 inputs and 16 outputs (*b*) 16 inputs and 8 outputs

(*c*) 8 inputs and 8 inputs/outputs (*d*) 8 inputs and 8 outputs

13. A Counter can be implemented in a PLD using ABEL by

(*a*) equation entry (*b*) truth table entry

(*c*) state diagram entry (*d*) all of the above

(*e*) (*a*) and (*b*) only

14. The ABEL expression A = [0, 1, 0]

(*a*) means that state A is defined by the binary value 010

(*b*) means that variable A has a sequence of values 0, 1, 0

(*c*) must be proceeded by a state definition statement

(*d*) both (*a*) and (*c*)

15. • CLK and • AR are examples of

(*a*) special constants (*b*) ABEL variables

(*c*) dot extensions (*d*) none of the above

ANSWERS

1. (*d*)	**2.** (*c*)	**3.** (*c*)	**4.** (*c*)	**5.** (*c*)	**6.** (*d*)
7. (*d*)	**8.** (*a*)	**9.** (*c*)	**10.** (*a*)	**11.** (*a*)	**12.** (*b*)
13. (*c*)	**14.** (*c*)	**15.** (*c*)			

13

FUNDAMENTALS OF MICROPROCESSORS

OUTLINE

Objectives

After completing this chapter, you should be able to,

- define microprocessor and know the different devices manufactured in the past 40 years.
- explain the difference between address bus, data bus and control bus.
- describe the operation of microprocessor.
- explain the architecture of 8085 architecture.
- describe different sets of microprocessor.
- explain pin details of 8085 microprocessor.
- explain the architecture of 8086 microprocessor.

13-1. Introduction

A microprocessor is a digital machine capable of executing many different software instructions and controlling various types of electronic devices. A microprocessor is a multipurpose, programmable logic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as output. A multipurpose device means it can be used to perform various complicated computing tasks as well as simple tasks. A programmable device means that it can be instructed to perform given tasks within its capability. All the microprocessors are designed to understand and execute many binary instructions. Microprocessor is also called as Central Processing Unit (or CPU), since it is the main part in any electronic computer system and processes the data. Nowadays most of the electronic devices, which we are using, are all microprocessor-based systems only. Some of the devices are,

1. Calculators 2. Cell phones 3. MP3 players 4. Digital watch
5. DVD players 6. Digital cameras 7. Personal Computers and so on

In this chapter, we will discuss about the microprocessor and its types, operation and uses.

13-2. History of Microprocessor

In the early days of computers, vacuum tubes were used. But the vacuum tubes are not able to perform the good work. After that transistors were used in place of vacuum tubes. During that time, the computers were shrinked and required less power. After that integrated circuits came into the picture. That allowed hundreds of transistors which can be replaced by a single IC chip.

The first microprocessor was introduced by Intel during the year 1971. It was named Intel 4004 and it was 4 bit wide length. It was not a powerful one and it did only addition and subtraction. It was used in pocket calculators. The Intel 4004 is shown in Fig. 13-1(*a*). It was followed by 8008, which is a 8 bit wide. This is shown in Fig. 13-1(*b*). In the year 1974, Intel 8088 was introduced and that was used in the IBM PC for the computer development. These microprocessors are shown in Fig. 13-1(*c*). Nowadays Pentium processors are introduced and that is 32 bit wide. It is shown in Fig. 13-1(*d*). Like Intel, so many companies manufactured different types of microprocessors. Table 13-1 shows different sets of microprocessors and its types. Zilog manufactured one processor called Z-80 in 1975. It has 8 bit data width and is shown in Fig. 13-1(*e*).

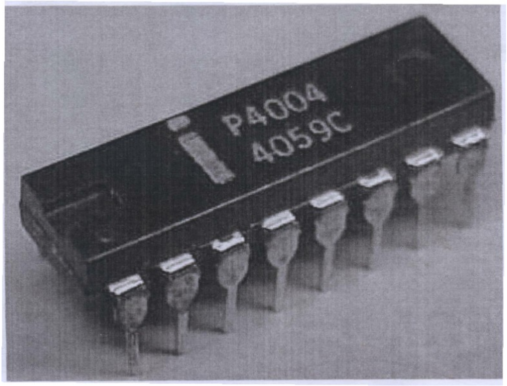


Fig. 13-1(*a*). Intel 4004

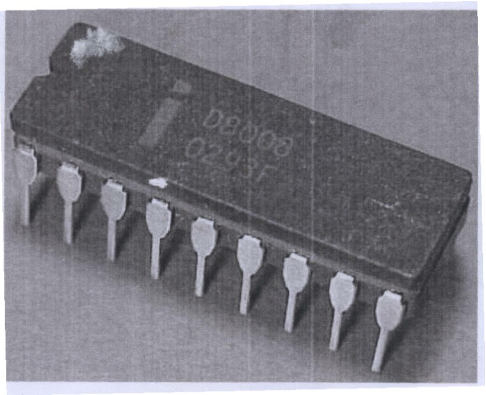


Fig. 13-1(*b*). Intel 8008

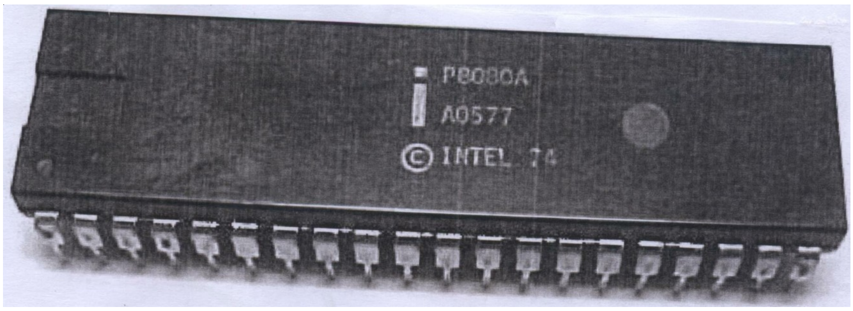


Fig. 13-1(*c*). Intel 8080

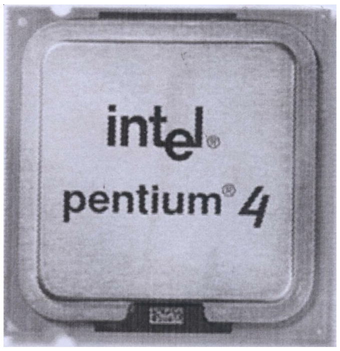


Fig. 13-1(*d*). Intel Pentium 4

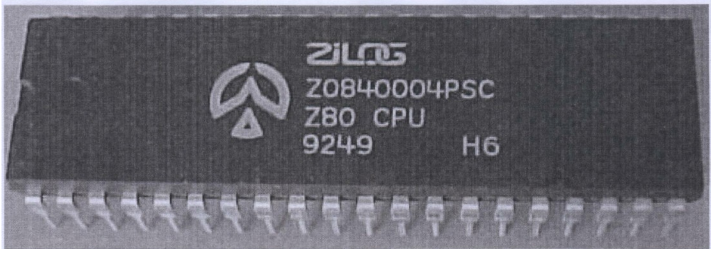


Fig. 13-1(*e*). Zilog Z-80

Table 13-1. Some Popular Microprocessor Chips

Microprocessor Name	Manufacturer	Description
Intel 4004	Intel Corporation	The first microprocessor (4 bit wide) developed in 1971
Intel 8008	Intel Corporation	8-bit microprocessor developed in 1972
Intel 8080	Intel Corporation	8-bit microprocessor developed in 1974
Z-80	Zilog, USA	8-bit microprocessor developed in 1979
Intel 8088	Intel Corporation	First 8 bit processor with 16-bit internal architecture developed in 1979
M 68000	Motorola	16-bit microprocessor developed in 1983
Intel 80186	Intel Corporation	16-bit microprocessor developed in 1983
M 68020	Motorola	32-bit microprocessor developed in 1984
Intel 80386	Intel Corporation	32-bit microprocessor developed in 1992
Pentium	Intel Corporation	32-bit, 64 bit wide processor developed in 1993
Pentium II	Intel Corporation	32-bit 64 bit wide processor developed in 1998
Pentium III	Intel Corporation	32-bit 64 bit wide processor developed in 1999
Pentium 4	Intel Corporation	32-bit, 64 bit wide processor developed in 2000
Pentium 4	Intel Corporation	32-bit, 64 bit wide processor developed in 2004

All these processors are different from one another by the following measures namely total number of transistors, clock speed, Data width and MIPS (Million instructions per second). Nowadays the Pentium 4 processors are having MIPS of about 3.6 GHz. Every year new microprocessors are introduced into the market with more instruction sets and faster speeds. Nowadays a technology called RISC (Reduced Instruction Set Computer) is used in the microprocessor development. Here only a small portion of the entire instruction set is used. By designing a machine, that uses common instructions for different operations, the processing speed of the processor increases. Nowadays 64 bit width processors are introduced and the researches are still going on in so many technology companies. Some of the major companies which are manufacturing microprocessors are listed below.

- ❍ Intel
- ❍ Texas Instruments
- ❍ ST Microelectronics
- ❍ ARM Holdings and so on.
- ❍ AMD (Advanced Micro Devices)
- ❍ Analog devices
- ❍ VIA Technologies

13-3. Basics of Microprocessor

Microprocessor is a unit that processes both arithmetic and logical operations. Therefore it essentially contains arithmetic-logic-unit (or ALU). These kinds of processing should be controlled by a control unit. The processed data should be stored in a particular place. So it is having a register array unit. Fig. 13-2 shows a simple micro-processor representation.

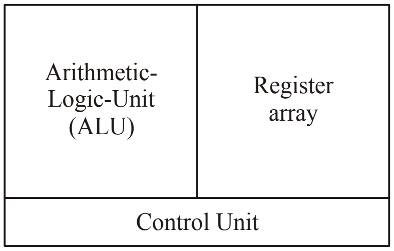


Fig. 13-2. Representation of microprocessor.

Arithmetic logic unit performs all arithmetic operations such as addition, subtraction, multiplication, division etc. and logical operations such as AND, OR, X-OR etc. The result of these operations is stored in a register array. The register array contains a number of general purpose registers and special purpose registers as well. Some of the registers are accessible to the users by the instructions. But some other registers are not accessible. The accessible registers are called as general purpose registers. The registers which are accessible to the user will get the instructions from memory and some temporary registers.

Control Unit provides timing and control signals required for execution of any operations inside or outside of the processor. It also controls and synchronizes all the operations performed by various sections of the microprocessor.

13-4. Microprocessor Buses

To communicate with memory or any other peripheral devices which are attached to the microprocessor, a large number of input, output and control pins are required. These pins are divided into three categories, namely, address bus, data bus and control bus, and shown in Fig. 13-3.

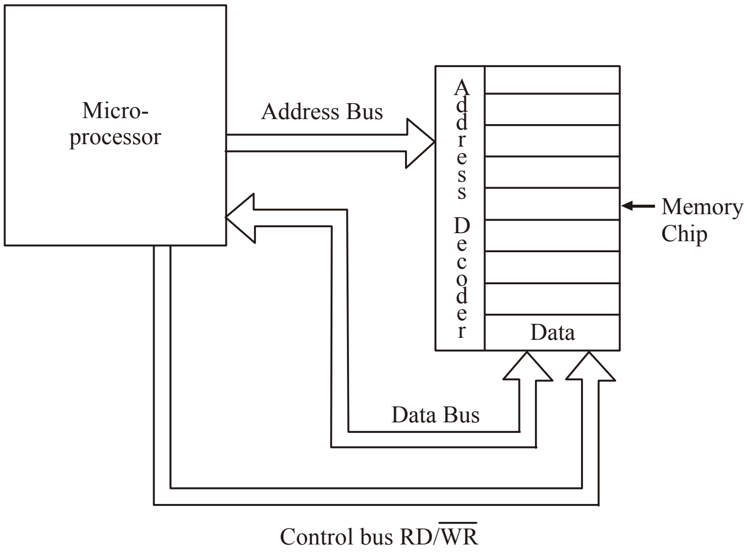


Fig. 13-3. Different types of buses.

Here the address bus is unidirectional. That means, data is flowing in one direction only, *i.e.*, from microprocessor to memory and peripherals. Each memory location has a particular address. The 8085 microprocessor has 16 address lines or pins. So this can address to $2^{16} = 65536$ memory locations.

The data bus is bidirectional. That means data can flow in both the directions, *i.e.*, between microprocessor and memory or peripheral. The number of pins forming the data bus is known as bus width. The data is transferred in the form of binary digits (bits). The number of bits that a microprocessor can process at a time is called as word size or word length. This is very important for the design of the microprocessors. So microprocessors are classified into 8-bit, 16-bit, 32-bit or 64-bit system according to its word length.

The control bus is used to synchronize the operation of the microprocessor with the operation of the external circuitry. Normally address bus and data bus have a group of pins. But control bus consists of individual pins.

To understand the function of a microprocessor bus, let us assume that microprocessor has to read a data from a certain memory location. Now the microprocessor places the address bits on the address bus. Now the address bit format of the address bus is decoded by an external logic circuit and the memory location is identified. Now the control bus in the microprocessor will send a read signal to activate the memory chip for a read mode. Once the read signal is received, the memory chip will place the data on the data bus and this will be transferred to the microprocessor.

13-5. Microprocessor-based System

Any basic microprocessor based system will have components like memory, input/output and timing. Some advanced systems will be having components like Video Controllers, A/D and D/A Converters, USB Controls and so on. Fig. 13-4 shows a basic microprocessor-based system. As we studied in the earlier chapter, the system has three buses, namely, address bus, data bus and control bus. These three buses are called as System bus. The microprocessor executes the instructions which are stored in the memory. All devices attached to the system bus will communicate with the processor with a specified period of time. This timing circuitry maintains the necessary time for proper functioning of the hardware. The timing circuit has a crystal oscillator that maintains the operating speed of the processor.

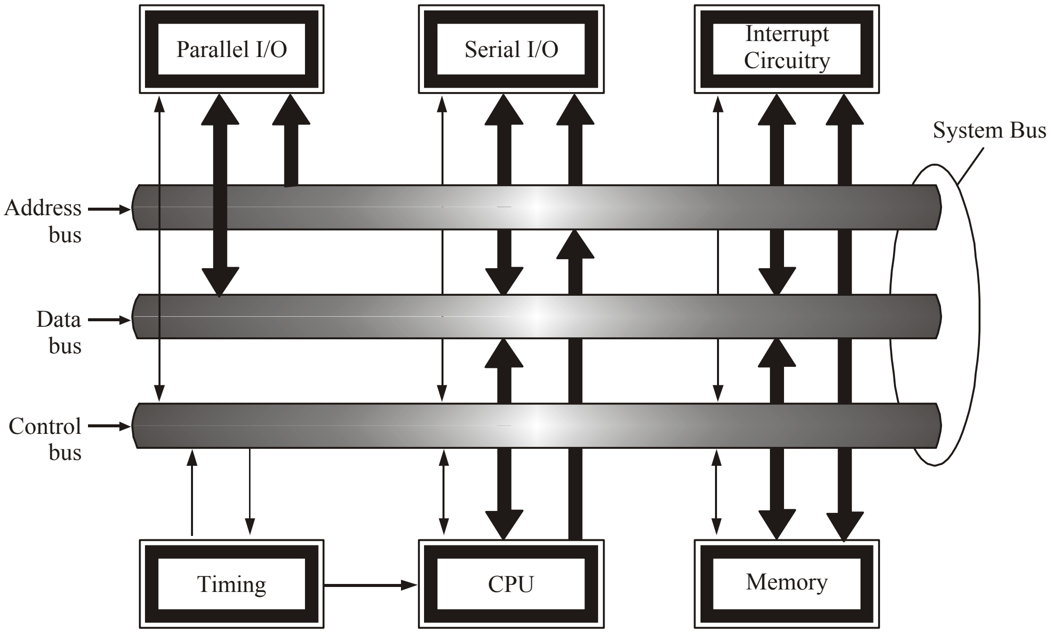


Fig. 13-4. Block diagram of microprocessor-based system.

Here the CPU subsystem will be having a microprocessor and the necessary associate logic circuit functions to enable the CPU to communicate with the System bus. These additional logic circuit functions will be bus drivers, bus controls and the control signals.

The microprocessor used in the system normally depends on the functionality of the system or our requirements. Some simple tasks or functions can be performed by an 8-bit microprocessor (*i.e.*, 8 bits of data at a time). But for very complicated systems, we need to have higher word length microprocessors, either 16 or 32 bit microprocessors.

Read-only memory (ROM) and random access memory (RAM) are part of the memory section. The ROM will be having the data to start up the system, since in most of the microprocessor-based systems, startup programs will be stored in the ROM. Normally this code will configure or initialize the peripherals of the system. In some cases only ROM will be there. All the entire operations will be done with codes in the ROM. In some systems, use the ROM program to download the main program into RAM from a large and most complicated systems. It has the provision to store the data to the external devices too such as hard-discs, CD-ROMs and FLASH drives.

Sometimes when the microprocessors are used in the control applications, during that time it may have to take some external control also. For example, real word examples, like during the operation of lifts in a building, if a power failure occurs, there will be some software program to handle such an issue. So this event interrupts the microprocessor from its normal operation to execute the unexpected event. So an interrupt system is used to inform the microprocessor about an expected event to be serviced first and after that let it continue the normal operation where it stopped. The interrupt control circuitry will vary from system to system. Some systems may be having only one interrupt, but some advanced applications will have so many interrupts.

Some systems will be having Serial Input/Output for communication with the realtime systems. Many devices will be connected to the microprocessor system by a serial communication. Here data is transmitted one bit at a time. Its speed is very slow when compared to other communication systems like parallel I//O and USB. But the design of the system is very simple since it requires only two wires. On the other hand, parallel Input/Output requires a minimum of eight wires. But it is very fast and used in so many applications like transferring data to A/D and D/A converters, printers, controlling lights and so on.

Mother board which we are using in our personal computers are Intel Pentium microprocessor based systems only. It is shown in Fig. 13-5.

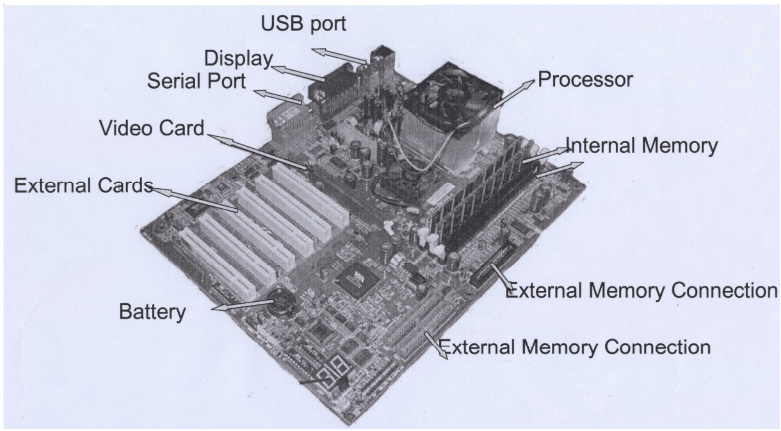


Fig. 13-5. Schematic Mother board

We have one more device called as microcontroller. It is a simple device and has many features. Actually microprocessors are made of microcontrollers with in-built features such as RAM, ROM, parallel Input/Output ports and timing intervals. Some controllers are having A/D converters too. But microcontrollers are not used for large systems, because they are having small memory addressing capability and limited processing speed.

13-6. Operation of Microprocessor

While executing a program, microprocessor normally executes three kinds of operations. Those are fetch, decode and execute instruction cycles. During the fetch operation, the processor loads an instruction from the memory into its internal instruction register. Nowadays the advanced microprocessors load more than one instruction into a special buffer to decrease the program execution time. While the microprocessor decoding the current instruction, other instructions are read from the memory and stored into a special type of internal memory called cache. Thus the microprocessor saves time by doing the two operations at the same time. In the decode cycle, the microprocessor determines what type of instruction has been fetched. Then this data is passed into the execute cycle. In the execution cycle, if necessary data may be read from the memory or data will be written into the memory based on the instructions. This operation is shown in Fig. 13-6.

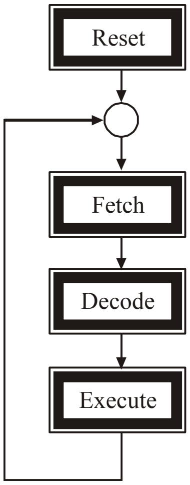


Fig. 13.6. Microprocessor's operation.

Here the operation is starting from the Reset mode. Once the power is switched on or whenever there is a severe failure of the system, then these cycles will be repeated. Normally this process will be repeated until the power is turned off or the processor is stopped while executing.

13-7. Microprocessor Instructions

The entire group of instructions, known as instruction set, determines the processor to do certain operation. The instruction set is a binary pattern designed for the microprocessor. The sequence of instruction for the microprocessor to perform a particular task is called a program. Each instruction will have the following information.

1. **The Operation Code (or Opcode) :** This will specify what operation is going to be performed. It is specified by some names called mnemonics. For example, SUB for the Subtraction Operation.
2. **Source of data :** This specifies the data which is used with the Opcode. Some instruction requires only one operand. For example, the instruction MOV A, B transfer data from B to A. Here the source of data lies with register B. Some instructions may involve two operands. For example ADDC, add the content of register C to the content of accumulator and store the result in the accumulator. Here the operation involves two operands. One operand operation is called as monodic and two operand operation is called dynaic.
3. **Destination of result :** The destination of result may be a register, memory-location or an output device. The destination is usually specified in the instruction. For example, MOV C, A, transfer the contents of register A to register C. Here destination of result is register C. In some instructions, destination is not specified and it is implicitly taken as an accumulator.
4. **Address of next instruction :** This part of instruction specifies the address of next instruction. Fig. 13-7 shows the format of the complete instruction.

OP-code	Address of operand 1	Address of operand 2	Address of destination result	Address of next instruction

Fig. 13-7. Instruction format.

The length of the instruction may be one byte, two bytes, three bytes or four bytes long. A single byte instruction has only Opcode. Two bytes instruction has Opcode and 8 bits of operand address whereas three byte instruction has opcode and 16 bits of operand or address. This is illustrated in Fig. 13-8.

OPcode

(*a*) one byte instruction

OPcode	Operand/ address

(*b*) two byte instruction

OPcode	Lower bits of operand	Higher bits of operand

(*c*) three byte instruction

Fig. 13-8. Instruction lengths.

13-8. Microprocessor 8085 Architecture

The internal structure of microprocessor is called its architecture. The intel 8085A is an 8-bit microprocessor. That means it has 8-bit bidirectional data lines which are used for data input as well as data output. Fig. 13-9 shows the architecture of 8085. The address bus is 16 bit wide. So it can address 64K memory locations or I/O peripherals. The lower eight bit address line ($A_0 - A_7$) are used as data line and hence multiplexed and referred as AD ($AD_0 - AD_7$) bus. The time-multiplexed bus system is used to save the number of pins in IC package. The intel 8085A uses a positive single supply. It has 40 pins.

The details of different blocks are given below.

1. **Arithmetic Logic Unit :** It is the most important block of microprocessor. It consists of large and very complex combinational blocks which perform all arithmetic and logic functions. The inputs to ALU are the contents of the accumulator and the temporary register. After any operation, the results are stored into the accumulator through an 8-bit internal bus.
2. **Accumulator :** It is the most active register of all the registers in the architecture. It is an 8-bit register which stores one operand during an ALU operation. It is also called as register A. The bidirectional arrow between the data bus and the accumulator means that the accumulator can send or receive the data. The data from the accumulator can be transferred to any other 8-bit temporary register or to any memory location.
3. **Flags :** There are five flags available in the 8085. They can set or reset based on the result of ALU operation. The flags indicate certain conditions that arises during the arithmetic and logic operation. The five flags are S, Z, AC, P and CY.

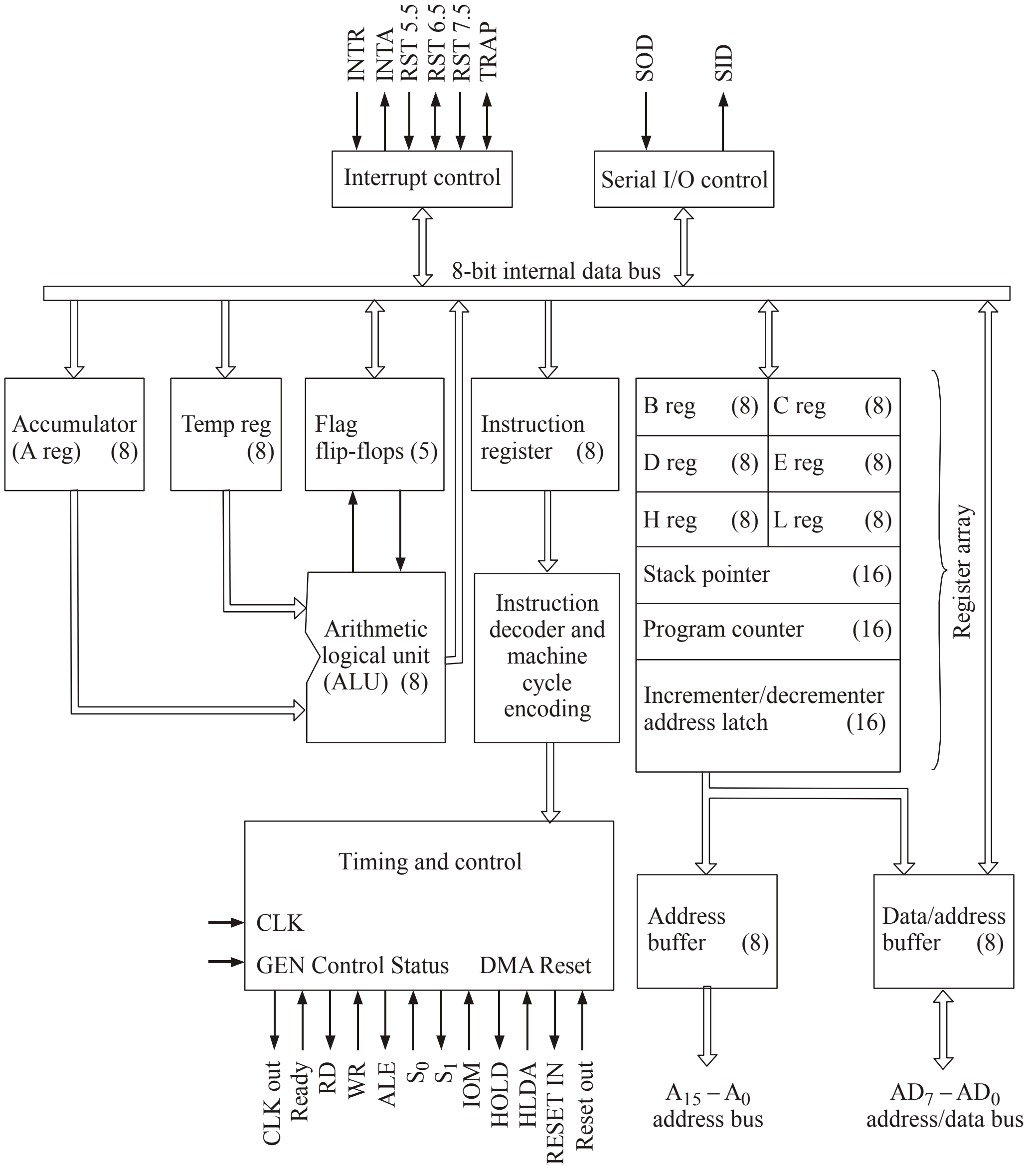


Fig. 13-9. Block diagram of 8085 architecture.

(*i*) **Sign flag (S) :** The flag is set when the contents of accumulator becomes negative during the execution of an instruction. If this flag is 0, the result of accumulator is a positive number. Bit 7 of flag register indicates the status of this flag.

(*ii*) **Zero flag (Z) :** It the result of ALU operation is 0, this flag is set to 1. If the result is non-zero, the flag will reset. Bit 6 of flag register indicates the status of zero flag.

(*iii*) **Auxiliary flag (AC) :** This flag is also known as auxiliary carry flag. If the bit 3 of accumulator generate carry and passed on bit 4 of accumulator, this flag is set. This flag is used for BCD operation. Bit 4 of flag register indicates the status of AC flag.

(*iv*) **Parity flag (P) :** This flag indicates whether the total number of 1's in the accumulator after execution of an instruction is odd or even. If number of 1's become even, then this flag is set otherwise this flag will reset. Bit 2 of flag register indicates the status of this flag.

(*v*) **Carry flag (CY) :** Bit 0 of the flag register indicates the status of this flag. This flag is set when an addition operation in ALU results in carry or when the subtraction operation results in borrow otherwise this flag is reset.

Out of 8 bits of flag register shown in Fig. 13-10, only 5 bits are used as flags. The remaining 3 bits are used for the internal processing of microprocessor and treated as don't care.

4. **Timing and control unit :** This unit is used to generate proper timing and control signals which are controlled and synchronized all the operation. Most of the microprocessors use an external crystal oscillator to determine the clock frequency from which the time and control signals are generated. The speed of the microprocessor directly depends on the clock frequency.
5. **Instruction register :** Any instruction fetched by the microprocessor is directly stored in the instruction register (IR). The contents of IR drives the instruction decoder and machine cycle encoder. This generates the necessary timing and control signals as per the instructions.
6. **General purpose register (or Register array) :** Microprocessor has six general purpose registers B, C, D, E, H and L. These registers can be used as single 8-bit registers or may be combined to form register pairs to hold 16-bit data. The allowed pairs are BC, DE and HL. The upper eight bits are stored in the first register of pairs *i.e.*, B, D or H. The lower eight bits are stored in the second register of pairs *i.e.*, C, E or L. The other registers are W and Z. These registers are known as temporary register. These registers are not accessible by programmer and perform only special function assigned to them by the architectural designer.
7. **Interrupt control :** Sometimes it becomes necessary, that if an unexpected event occurs while doing the normal execution, then we have to stop the normal routine and we have to serve the unexpected event. The interrupt control in the processor does the same function. Interrupt signal to the interrupt control block informs the microprocessor that some I/O peripheral wants to communicate or send/receive the data. The 8085 has five interrupts. If more than one interrupt is active, then based on their priority, the interrupts will be executed. Table 13-2 shows the five interrupts with their priority.

Table 13-2. Microprocessor 8085 Interrupts

Interrupt	Priority
TRAP	1
RST 7.5	2
RST 6.5	3
RST 5.5	4
INTR	5

When an interrupt occurs, microprocessor completes the execution of current instruction. Then it services the interrupt routine and returns to the main program once completion of service subroutine.

8. **Stack pointer (SP) :** It is a 16-bit register. The content of this register always point to the address of the top of the task. The stack pointer is not accessible physically. This can only access through instruction.
9. **Program Counter (PC) :** It is also a 16-bit register. Its content always point to the address of next instruction to be fetched. When an instruction is fetched the program counter is immediately incremented to point to the next location for the next fetching cycle. This register is accessible through the instruction only. The programmer can use certain instruction to change the content of the program counter (PC). These are called JUMP Instruction before they make the microprocessor to jump from one particular location to another location instead of executing the program in sequence.
10. **Address buffer and Address-data buffer :** These are 8-bit buffer registers. The output of these buffer registers drives the external address bus and data bus. The internal 8-bit data bus is also connected to these buffer register through bidirectional connection. Thus these registers can send or receive data from 8-bit internal data bus.
11. **Serial I/O Control :** This control is used to make serial communication between microprocessor and I/O peripheral. Serial input data has to be converted to 8-bit parallel data by this block before the microprocessor can use it. For this conversion, RIM instruction is used. Similarly SIM is used to convert data received from microprocessor into serial 8-bit data. The serial data enters at SID (serial input data) and serial output data leaves at SOD (serial output data).

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
S	Z	X	AC	X	P	X	CY

Fig. 13-10. Flag register details.

13-9. Instruction Set of 8085

The entire group of instructions designed in the microprocessor are called instruction sets. The instructions of 8085 are classified into five functional catagories. Those are,

1. Data transfer operations
2. Arithmetic operations
3. Logical operations
4. Branching operations
5. Machine control operations.

13-10. Data Transfer Operations

This group of instructions transfer or copy the data from one location called source to another location called destination. Even though it has many types of instructions, the following types of data transfer instructions are important from subject point of view.

1. MOV r_1, r_2 : The contents of register r_2 are moved to register r_1. The registers r_1 and r_2 may be any one of the general purpose registers or accumulator.
2. MOV r, M : The contents of memory location (whose address is in register H and L) are moved to register r.

3. MOV M, r : The contents of register r are moved to memory location whose address is in register H and L.
4. MOV r, data : The data is moved to register r.
5. MOV M, A : The contents of accumulator are moved to memory location whose address is in register H and L.
6. LHLD addr : Load lower bits of address in register H and higher bits of address in L register.

You can refer the data sheet of 8085 for other data transfer instructions.

13-11. Arithmetic Operations

The instructions under this group perform arithmetic operations such as addition, subtraction, increment and decrement.

(*i*) **Addition :** Any 8-bit data or the content of any one of the general purpose registers or the content of memory location can be added to content of accumulator and the result is stored into the accumulator. The contents of two general purpose registers cannot be added directly. Some examples of this operation are ADDC, ADDE, ADDH, ADDL, ADDM, etc. The instruction DAD is an exception. It adds the 16-bit data directly into register pairs.

(*ii*) **Subtraction :** Any 8-bit number or contents of register or contents of memory location can be subtracted from contents of accumulator and the result is stored in the accumulator. Like addition operation, the subtraction operation cannot be performed between the two registers directly. If the result stored in the accumulator is negative then it is expressed in 2's complement. Some examples of this operation are SOBB B, SUBB C, SUBB M, SUBB D, etc.

(*iii*) **Increment/Decrement :** These instructions are used to increment/decrement the contents of any general purpose register, memory location or 16-bit content of register pair (such as HL) by I. These instructions differ from addition and subtraction instructions in an important way *i.e.*, they can perform in any one of registers or in memory location. Some examples of these operations are INR A, INR C, INR M, DCR M, DCR C, INX BC, DCX DE, etc.

The increment/decrement operation affects all the flags except carry flag whereas all flags including carry flags are affected in the addition and subtraction operation.

13-12. Logical Operations

These instructions perform various types of logical operations with the contents of accumulator. These are explained as follows :

1. **AND, OR, XOR :** These are the instructions for performing the ANDing, ORing and comparison operation with the contents of accumulator. The result of operation is stored in the accumulators. All the flags are affected but carry flag and the auxiliary carry flags are setting depends upon the instruction being used. Table 13-3 gives the flag settings of these instructions.

Table 13-3. Flag setting in AND, OR and XOR Operation

Instruction	Operation	Flag Setting	
		Carry Flag (CY)	Auxiliary Flag (AC)
ANA r	AND Register	0	1
ANA M	AND Memory	0	1
ANI data	AND Immediate	0	1
ORA r	OR Register	0	0
ORA M	OR Memory	0	0
ORI data	OR Immediate	0	0
XRA r	XOR Register	0	0
XRA M	XOR Memory	0	0
XRI, data	XOR Immediate	0	0

2. **Rotate Instructions :** These instructions are also known as shift instructions. Each bit in the accumulator can be shifted either left or right to the next position. The rotating instructions are listed below.

   (*i*) **RRC (rotate right) :** The contents of accumulator are rotated right one position. The highest order bit and carry flag (CY) are set equal to value shifted out of lower order bit position as shown in Fig. 13-11(*a*).

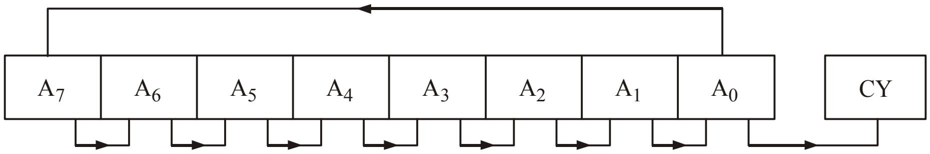


Fig. 13-11(*a*). RRC method

   (*ii*) **RLC (rotate left) :** The contents of accumulator are rotated left one position. The lower order bit and carry flag (CY) are set to the value shifted out of higher order bit position as shown in Fig. 13-11(*b*).

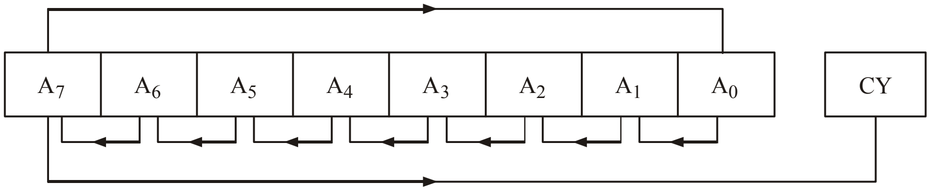


Fig. 13-11(*b*). RLC method

   (*iii*) **RAR (rotate right through carry) :** The contents of accumulator are rotated right through carry flag. The lower order bit of contents are shifted to carry flag and the bit of carry flag is shifted to MSB contents of accumulator as shown in Fig. 13-11(*c*).

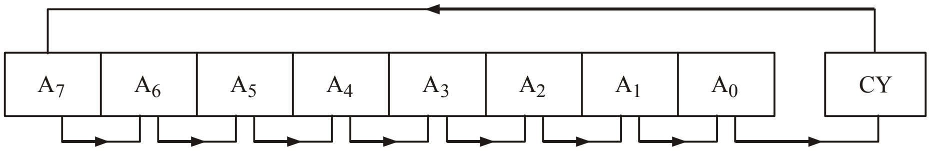


Fig. 13-11(*c*). RAR method

(*iv*) **RAL (rotate left through carry) :** The contents of accumulator are rotated left through carry flag. The higher order bit of accumulator is shifted to carry flag and the bit of carry flag is shifted to LSB of contents of accumulator as shown in Fig. 13-11(*d*).

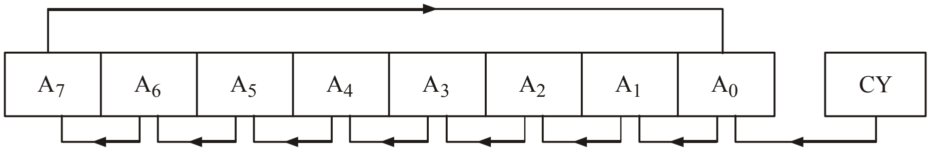


Fig. 13-11(*d*). RAL method

3. **Compare Instructions :** Any 8-bit number or the contents of a register or a memory location can be compared for equality, greater than or less than with the contents of accumulator. Table 13-4 gives various types of compare instruction with setting of flag registers.
4. **Complement Instructions :** The contents of the accumulator can be complement *i.e.*, all 0's are replaced by 1's and all 1's are replaced by 0s. Example of this instruction is CMA. The CMA instruction does not affect any one of the flag. Another complement instruction is CMC. This instruction complements the carry flag.

Table 13-4. Compare Instructions

		Flag Setting	
Instruction	Operation	Carry Flag (CY)	Zero flag (Z)
CMP r	Compare	For A < r, 1	For A < r, 0
	Register	For A = r, 0	For A < M, 0
CMP M	Compare	For A < M, 1	For A < M, 0
	Memory	For A = M, 0	For A = M, 1
CPI	Compare	For A = data, 0	For A = data, 1
	Immediate	For A < data, 1	For A < data, 0

13-13. Branch Operations

The instructions of this group alter the sequence of program as per requirement either conditionally or unconditionally. Following are the instructions of this group.

1. **Jump instruction :** The jump instruction may be conditional or unconditional. The conditional jump instructions test for a certain condition and alter the program sequence when the condition is met. The various conditional jump instructions are :

JNZ	Jump if	zero flag is not set
JZ	Jump if	zero flag is set
JNC	Jump if	carry flag is not set
JC	Jump if	carry flag is set
JPO	Jump if	parity of byte is odd
JPE	Jump if	parity of byte is even
JP	Jump if	sign flag is not set
JM	Jump if	sign flag is set

Conditional jump instructions will cause the program counter to be loaded with address specified in the instruction only if the condition is true whereas unconditional jump instructions transfer the control directly to the instruction whose address is specified with it. JMP, addr is unconditional Jump instruction.

2. **Call instructions :** These instructions change the sequence of a program by calling a subroutine. The call instruction can also be conditional or unconditional. A call instruction contains the starting address of the subroutine in second byte and third byte for the instruction. In execution of call instruction, the control jumps to the address specified in the instruction only if the condition is true. Following are the conditional call instructions:

CNZ	Call if	Z = 0
CZ	Call if	Z = 1
CNC	Call if	CY = 0
CC	Call if	CY = 1
CPO	Call if	P = 0
CPE	Call if	P = 1
CP	Call if	S = 0
CM	Call if	S =1

CALL addr, is an unconditional call instruction. Upon execution of this instruction, the program counter is directly loaded with the address specified in the instruction without any condition check.

3. **Return Instructions :** These instructions are used to get the control back to the original sequence after execution of subroutine is over. These instructions do not contain the address and the program counter will be loaded with address specified in the stack pointer register. The return instruction can also be conditional or unconditional. The various conditional return instructions are listed below :

RNZ	return if	Z = 0
RZ	return if	Z = 1
RNC	return if	CY = 0
RC	return if	CY =1
RPO	return if	P = 0
RPE	return if	P = 1
RP	return if	S = 0
RM	return if	S = 1

RET is an unconditional return instruction.

4. **Restart Instructions :** These instructions are special purpose call instructions designed for use with interrupt. It is a single byte instruction and rarely appears as an instruction in the program. 8085 has eight restart instructions as listed below.

Instruction	**Address to which control is transferred**
RST0	0000H
RST1	0008H
RST2	0010H
RST3	0018H
RST4	0020H

RST5	0028H
RST6	0030H
RST7	0038H

5. **PCHL Instruction :** This instruction moves the content of HL pair register to the program counter. The execution of this instruction causes the control to jump to the address which was stored in H-L register pairs. The flags are not affected by this instruction.

13-14. Stack, I/O and Machine Control Operations

The instructions under this group control the machine functions such as Halt, Interrupt or do nothing.

1. **I/O Instructions :** IN and OUT are the I/O instructions. These instructions are two-byte. The IN instruction input an 8-bit data from a specified port to accumulator whereas OUT instruction output 8-bit data from accumulator to the port specified in the instruction. These instructions do not affect any flag.
2. **Stack Instructions :** The instructions mentioned here will perform the stack operations.

   PUSH Instruction : The PUSH instruction is used to push the content of B-C, D-E and H-L pair registers or program status word to the Stack Data is stored in stack on the basis of Last-in-first-out.

   POP Instruction : The POP instructions are used to move the contents from top of stack to the BC, DE and HL pair register of PSW (program status word).

XTHL and SPHL are two more instructions that perform stack operation. XTHL exchange the two bytes of data from the top of Stack with the contents of HL pair register. The SPHL instruction moves the contents of HL pair register with the stack pointer register.

3. **Interrupt Enable Instruction :** This instruction is used to disable the interrupt and it is written as DI.
4. **Interrupt Disable Instruction :** This instruction is used to disable the interrupt and it is written as DI.
5. **Halt Instruction :** This instruction stops the microprocessor. The microprocessor can come out of HALT state by hold and interrupt. If the hold line pin is high, then microprocessor leaves the halt state. As soon as hold line goes low, the microprocessor enters into halt state.

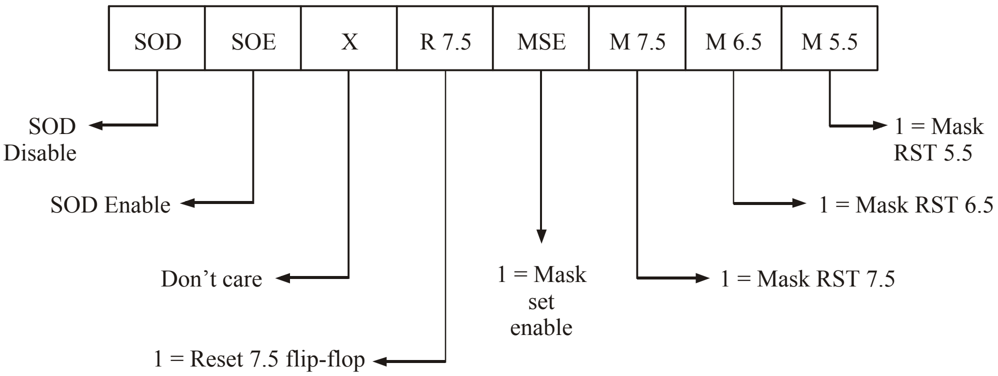


Fig. 13-12. Bit format of SIM (set interrupt mask).

6. **NO operation or NOP instruction :** This instruction is used to create delay of one machine cycle. It does not have any flag.
7. **SIM Instruction :** This instruction is executed to use the content of accumulator to perform the following functions :
   — Interrupt mask for RST 5.5, RST 6.5 and RST 7.5
   — Reset RST 7.5 interrupt
   — Load serial-output-data latch with bit 7 of accumulator.
   It is mentioned in Fig. 13-12.
8. **RIM Instruction :** This instruction is executed to load the data related to interrupts and serial input. It contains the following information :
   — Current interrupt mask status for RST 5.5, 6.5 and RST 7.5
   — Hard wire interrupts pending for RST 5.5, 6.5 and RST 7.5
   — Serial input data
   Fig. 13-13 shows the bit format of RIM instruction.

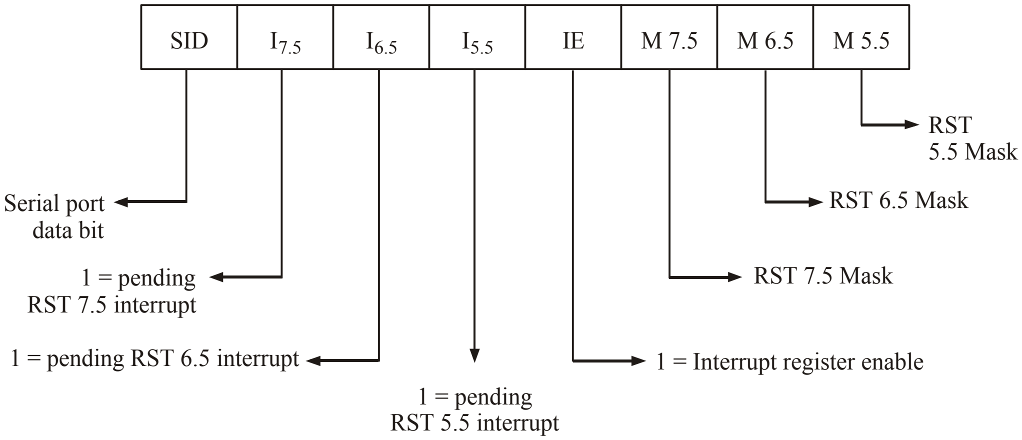

SID
$I_{7.5}$
$I_{6.5}$
$I_{5.5}$
IE
M 7.5
M 6.5
M 5.5
RST 5.5 Mask
RST 6.5 Mask
Serial port data bit
RST 7.5 Mask
1 = pending RST 7.5 interrupt
1 = pending RST 6.5 interrupt
1 = Interrupt register enable
1 = pending RST 5.5 interrupt

Fig. 13-13. Bit format of RIM (Read interrupt mask).

13-15. Addressing Modes

Each instruction used by the microprocessor copies 8-bit data from a source into a destination. The source and destination can be a register, an input/output port or an 8-bit number and is known as operand. The method of specifying the operand is called addressing mode. The 8085 microprocessor has five addressing modes. Those are,

1. Direct addressing mode
2. Register addressing mode
3. Register indirect addressing mode
4. Immediate addressing mode
5. Implicit addressing mode

1. **Direct addressing mode :** In this addressing mode, the address of operand is specified in the instruction. In 8085 microprocessor, the instruction used with this mode consists of 3 bytes. The first byte is decoded as opcode, 2nd and third bytes are the address of the operand. Some examples of direct addressing are,

LDA FFA	—	Load accumulator direct with contents of memory location whose addressing is FFA.
STA FFA	—	Store the contents of accumulator into the memory location whose address is FFC.

This method is also called as absolute addressing method. IN and OUT instructions are also direct addressing instructions but these are only two bytes long.

2. **Register addressing mode :** In this addressing mode, the address of operand is the address of general purpose register. This type of instructions are of one byte long. Some examples of register addressing mode instructions are,

MOV A, D	→	Move contents of register D to accumulator
MOV C, A	→	Move content of accumulator to register C
ADD, C	→	Add contents of register C to accumulator.

3. **Register indirect addressing mode :** In this addressing mode, the address of operand is specified by the contents of a register pair. A typical instruction using register indirect address is

MOV A, M → Move contents of memory location, whose address is in H-L register pair to accumulator.

4. **Immediate addressing mode :** In this addressing mode, the operand is placed in the memory immediately after the opcode. This addressing mode is very fast. For 8-bit microprocessor, it has two bytes long. Here first byte is opcode and 2nd byte is the data. For 16-bit microprocessors, these can be 3 bytes long. Some examples of immediate addressing modes are listed below.

MVI A, 03H	→	MOVE 03H to accumulator immediately.
LD A 07H	→	Load accumulator with 07H
ADI 05H	→	Add 05H to contents of accumulator immediately
LXIH, 0050H	→	Load 0050H in HL register pair.

5. **Implicit addressing mode :** In this addressing mode, the operand is assumed in accumulator. Sometimes operand is implicitly implied in the instruction itself. These instructions are one byte long. Some examples of implicit addressing instructions are,

CMA	→	Complement the contents of accumulator
STC	→	Set carry flag
HLT	→	Halt or stop processing.

Table 13-10 gives a list and a brief description of all the instructions of Intel 8085.

13-16. 8085 Machine Cycles and Bus timing

The 8085 microprocessor is designed to execute 74 instruction types. Each instruction has two parts. One is operation code (Opcode) and the other is operand. The opcode is a command such as AND and the operand is an object to be operated such as a byte or contents of the register. An instruction can be a single byte or multibyte instruction. All actions in the microprocessor are controlled by either loading or trailing edge of the system clock. Basically a microprocessor performs the following two operations. Those are :

1. Fetch an instruction from memory, and
2. Execute the instruction.

The time taken by the microprocessor to perform fetch and execute operations is called fetch and execute cycle. The sum of fetch and execute cycle is called instruction cycle.

i.e. $T_{IC} = T_{FC} + T_{EC}$

where T_{IC} = Time required for instruction cycle

T_{FC} = Time required for fetch cycle

T_{EC} = Time required for execution cycle.

Each instruction cycle consists of one or more numbers of machine cycles and each machine cycle is divided into 3-6 T-states (clock). The first machine cycle of an instruction is always an op-code cycle. It is of at least 4 T-states. The 8085 uses the first three states T_1-T_3 to fetch the code and

T_4 to decode and execute the opcode. During the fetch cycle, the process reads a machine code from memory or I/O device. The opcode fetch cycle is identified by the status signals $(IO/\overline{M} = 0, S_1 = 1, S_0 = 1)$. The active low $IO/\overline{M}$ signal indicates that it is a memory operation. Similarly S_1 and S_0 equal to 1 indicates that it is an opcode fetch cycle. A typical fetch cycle is shown in Fig. 13-14.

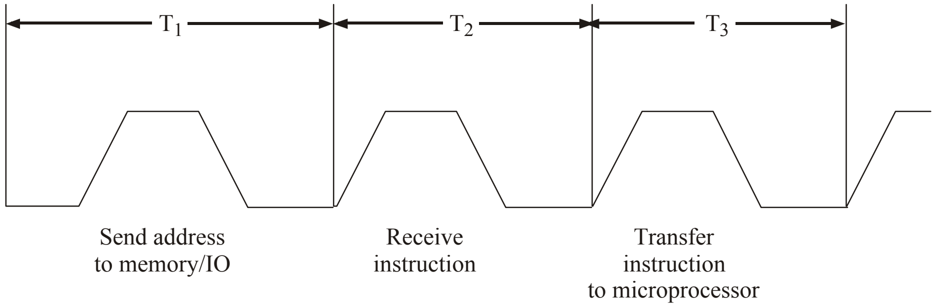

T1
T2
T3
Send address to memory/IO
Receive instruction
Transfer instruction to microprocessor

Fig. 13-14. Fetch cycle.

The execute cycle may be of one or more machine cycle depending upon the length of the instruction. During this cycle, the instruction is decoded and translated into specific activities. The instruction cycle, fetch cycle and execute cycle are related as described in Fig. 13-15.

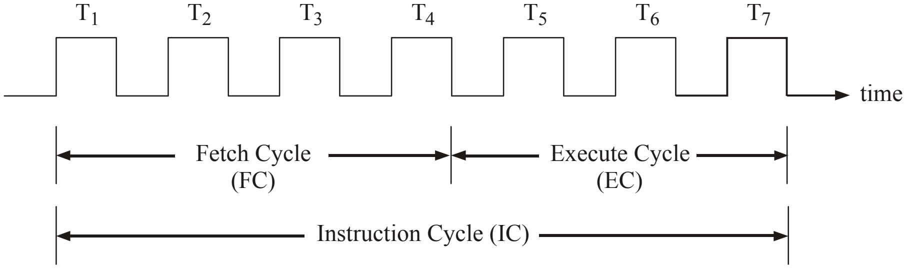
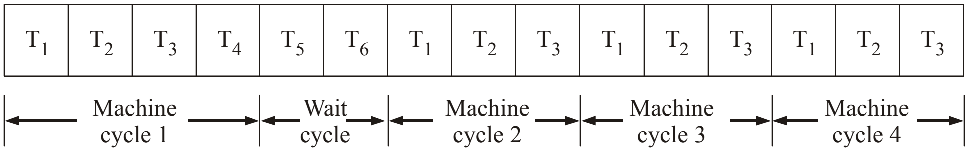


Fig. 13-15. Relationship among IC, EC and FC.

Since instruction cycle can be of 1 to 3 bytes long, the instruction fetch is 1 to 3 machine cycle long. As the access time of the memory may vary, some wait clocks are required for the opcode fetch cycle to cope with the slow memories or I/O devices The machine cycle including the wait state is shown in Fig. 13-16.



T_1	T_2	T_3	T_4	T_5	T_6	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3
Machine cycle 1				Wait cycle		Machine cycle 2			Machine cycle 3			Machine cycle 4		

Fig. 13-16. Machine cycle including wait states.

13-17. Microprocessor Programming

A program is a sequence of instructions written to tell a microprocessor to perform a specific function. The instruction and data are fed to the microprocessor through input device. The commonly used input devices are keyboard and mouse. A program may be one or more than one instruction. After execution of an instruction, the results are displayed by the output device. Printers, LEDs, LCDs, CRT, etc. are the output devices. A microprocessor cannot complete a full functionality block unless it is connected with memory and Input/Output devices.

The 8085 programming model has six registers, one accumulator and one flag register. In addition, it has two 16-bit registers, stack pointer and a program counter. Microprocessor can understand program counters in bits 0 and 1. But to write a program using 0 and 1 is difficult. Each microprocessor has a symbolic code for each instruction called mnemonics. A program can be written very conveniently in mnemonic code and symbol and data can be written in hexadecimal notations. This type of language is called assembly language. There are different mnemonic codes and their equivalent hexadecimal codes are available.

A special device called assembler is required to translate this assembly language to machine language. Development of program involves the following basic steps :

1. Specify the task to be performed precisely. Note down the Input/Output devices and system constants.
2. Draw a flowchart showing the way of computations to accomplish the given task.
3. Write a program using mnemonic codes obtained from the instruction set supplied by the manufacturer.
4. Find the hexadecimal number for each instruction by searching through the set of instructions.
5. Enter/Load the program in the user memory in a sequential order by using a Hex Keyboard.
6. Execute the program by pressing the execute key. The results will be displayed by the output device.

The steps from 3 to 6 will be done in the microprocessor kit. This will be applicable for smaller programs. For larger programs, using the computer we can send the data.

13-18. The 8085 Interrupts

The interrupt I/O is a process of data transfer whereby an external device can inform the microprocessor that it requests attention. However the response to the interrupt request is directed or controlled by the microprocessor. The interrupt lines are the special input to the microprocessor control logic.

The 8085 microprocessor has five interrupts. Four of them are maskable interrupts and one is non-maskable interrupt. Among the four maskable interrupts, three interrupts are vectored interrupts. So when any interrupt, out of these three, occurs, then microprocessor branches or jumps to a predetermined address location. This predetermined location is called VECTOR. Each vectored interrupt has a different vector as shown in Table 13-5. The other non-vectored interrupt requires external hardware to give the address vector or location.

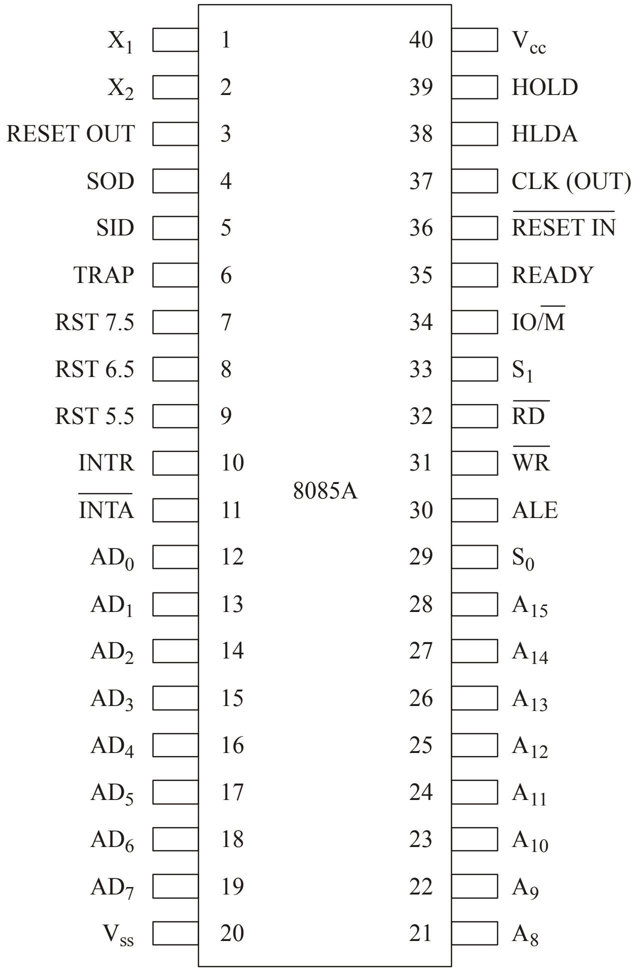
The microprocessor can ignore or delay a request of maskable interrupt, if it is performing some critical task. However if a non-maskable interrupt occurs (like TRAP), the microprocessor has to respond quickly. Whenever we reset the microprocessor, all the maskable interrupts are disabled. To make them more effective, the interrupts are controlled by the software. The instructions like EI (Enable interrupt) and DI (Disable interrupt) are used to enable or disable the interrupts respectively. If more than one interrupts occurs, then microprocessor attends the non-maskable interrupt first and afterwards based on the interrupt priority, other interrupts will be serviced.

Table 13-5. 8085 Interrupts

Sl. No.	Interrupt	Maskable/ Non-maskable	Vector Location	Priority
1.	TRAP	Non-maskable	0024H	Highest
2.	RST 7.5	Maskable	003CH	↓
3.	RST 6.5	Maskable	00344	↓
4.	RST 5.5	Maskable	002CH	↓
5.	INTR	Maskable	—	Lowest

13-19. PIN Description of IC 8085

The Intel 8085 is a 40-pin dual-in-line package device implemented with approximately 6200 transistors. Fig. 13-17 shows pin diagram of 8085 microprocessor. The functions of these pins are described below.



8085 Pinout

Fig. 13-17. Pin diagram of 8085 processor.

Pins 1 and 2 : The 8085 has crystal or R/C network connections to set the internal clock generator XI. It can also be an external clock input instead of crystal. For frequency stability crystal is preferred. The input frequency is divided by 2 to give the internal operating frequency. So to operate the 8085 at 3 MHz, a crystal of 6 MHz is required. These pins are labelled as X_1 and X_2.

Pin 3. The pin is labelled as RESET OUT and connected to peripheral chips for their initializations. A high signal on this pin resets the microprocessor, *i.e.*, all the registers and counters will be reset.

Pins 4 and 5 : Pin 4 is labelled as SOD (serial out data) and Pin 5 as SID (serial in data). These two pins are used for communication between microprocessor and external peripheral device. These pins are specially designed for software controlled serial input/output and eliminate the need for an input port and output port. SID is a 1-bit input port and SOD is a 1-bit output port. Data transfer through these pins are controlled by RIM and SIM instructions.

Pins 6 to 11 : These pins are interrupt controlled and labelled as TRAP, RST 7.5, RST 6.5, RST 5.5, INTR and $\overline{\text{INTA}}$. Already we have discussed about all the interrupts except INTR and $\overline{\text{INTA}}$. The INTR is a lowest priority interrupt and is used as general purpose interrupt. When the microprocessor is executing a program, it checks the INTR line during the execution of each instruction. If the INTR pin is high and the interrupt is enabled, the microprocessor completes its current instruction and disable the interrupt enable flip-flop and send an active low acknowledge signal called $\overline{\text{INTA}}$. The 8085 cannot accept any interrupt request until the interrupt flip-flop is enabled again. The signal $\overline{\text{INTA}}$ is used to insert a RET or call instruction through external hardware. As soon as the microprocessor receives the call instruction, it saves the address of next instruction on stack and jump to call location. At the end of the subroutine, the RET instruction receives the address where the microprocessor was interrupted and from there it will continue the execution.

Pins 12 to 19 : These pins are labelled as AD_0 to AD_7. These lines transmit the lower order bits of address and 8 data bits in the time division multiplexing. During 1st clock of each machine cycle, the lower order bits are transmitted and for remaining clock it becomes as a data bus. The lower order byte can be used to address $2^8 = 256$ I/O devices. These pins are bidirectional.

Pin 20 : Ground connection to chip or supply ground.

Pins 21 to 28 : These are the 8 unidirectional signal lines (higher order address bits) and labelled as A_8 to A_{15}. These remain in high impedance state during HOLD, HALT and RESET modes.

Pin 29 and Pin 33 : These pins are labelled as S_0 and S_1 respectively and provide input/output status control. The S_0 and S_1 are used to differentiate HALT, write, read and opcode fetch operation of the microprocessor. Table 13-6 shows different combinations of S_0 and S_1.

Table 13-6. Different modes of S_0 and S_1

S_1	S_0	Operation performed by microprocessor
0	0	HALT
0	1	Write
1	0	Read
1	1	Opcode fetch

Pin 31, 32 and 34 : These pins are labelled as $\overline{\text{WR}}$, $\overline{\text{RD}}$ and $\text{IO}/\overline{\text{M}}$ respectively. These pins work together and decide which operation should take place like memory read/write or I/O read/write. $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are active low signals. Here low $\overline{\text{RD}}$ represents read operation, low $\overline{\text{WR}}$ represents write operation. If $\overline{\text{RD}}$ is low and $\overline{\text{WR}}$ should be high or vice-versa. Table 13-7 shows the combination of these signals and the operation.

Table 13-7. Different Combination of $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\text{IO}/\overline{\text{M}}$.

$\overline{\text{RD}}$	$\overline{\text{WR}}$	$\text{IO}/\overline{\text{M}}$	Operation
0	1	0	Memory read
1	0	0	Memory write
0	1	1	Input read
1	0	1	Output write

Pin 30 : This pin is labelled as ALE (address latch enable). It is a positive going pulse and generated each time during the operation of microprocessor. This signal is essential for multiplexed operation of address/data bus. When the address bits (lower order bits) are put on to AD_0 to AD_7 pins, the ALE signal is activated and the address gets latched into the latch. Then if the ALE is deactivated, then the bus can carry the data.

Pin 35 : This pin is labelled as "READY". It is connected to a peripheral device. The speed of the peripheral device may or may not be matching the speed of the microprocessor. So if the peripheral is not for data transfer, it sends a low signal to READY pin of microprocessor. By receiving the low signal at READY, the microprocessor goes to Wait or No operation (NOP) mode. This will happen till the microprocessor receives a high signal at READY pin. After that data transfer will happen.

Pin 36 : This pin is labelled as $\overline{\text{RESET IN}}$. When $\overline{\text{RESET IN}}$ is low, the processor goes into the reset mode, (*i.e.*, instruction registers and program counters are made to reset). The microprocessor again starts processing after receiving the signal high on this pin.

Pin 37 : This pin is used to synchronize the microprocessor operations with the peripheral device. It is labelled as CLK.

Pin 38 and 39 : These pins are labelled as HLDA and HOLD respectively. These signals are used in DMA (direct memory access) operations to speed up the data transfer. Fig. 13-18 shows the direct memory access operation.

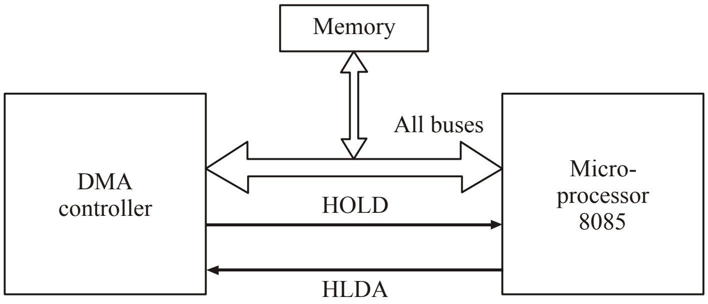


Fig. 13-18. DMA operation.

When DMA controller is ready to take over the control, it sends a logic HIGH signal to HOLD pin of microprocessor. With high HOLD signal, the microprocessor gives its control to address and data bus to DMA controller by sending a high acknowledge signal on HLDA pin. The DMA controller transfers the data at high speed and returns the control back to the microprocessor by sending back a low HOLD signal.

Pin 40 : V_{CC} Connection (+ 5V) to chips.

Instruction Set of 8085 with Flag Status and Machine Cycles

B — Bytes	MD — Flag is Modified	I — flag set
M — Machine Cycle	Blank — No change in flag	O — flag reset
T — T-states		

Instruction		Description	B/M/T	S	Z	AC	P	CY
ACI	DATA	Add 8 bit and CY to A	2/2/7	MD	MD	MD	MD	MD
ADC	REG	Add register and CY to A	1/1/4	MD	MD	MD	MD	MD
ADC	M	Add Memory and CY to A	1/2/7	MD	MD	MD	MD	MD
ADD	REG	Add reg to A	1/1/4	MD	MD	MD	MD	MD
ADI	DATA	Add 8 bit	2/2/7	MD	MD	MD	MD	MD
ANA	M	Add Memory with A	1/2/7	MD	MD	1	MD	0

(*Contd.*)

Instruction		Description	B/M/T	S	Z	AC	P	CY
ANI	Data	Add 8-bit data with A	2/2/7	MD	MD	1	MD	0
ANA	REG	AND register with accumulator	1/1/4	MD	MD	1	MD	0
CALL	Addr	Call an unconditional	3/5/18					
CC	Addr	Call on CY	3/5/9-18					
CM	Addr	Call on sign flag	3/5/9-18					
CMA		Complement A	1/1/4					
CMC		Complement Carry	1/1/4					
CPI	data	Compare 8-bit with A	2/2/7	MD	MD	MD	MD	MD
DAA		Decimal adjust A	1/1/4	MD	MD	MD	MD	MD
DAD	RP	Add register pair to HL	1/1/4	MD	MD	MD	MD	MD
DCR	REG	Decrement Register	1/3/10	MD	MD	MD	MD	MD
DCX	RP	Decrement reg. pair	1/1/4					
D2		Disable interrupt	1/1/4					
EI		Enable interrupt	1/1/4					
HLT		Stop	1/2/5					
IN	PORT	Input from 8-bit port	2/3/10					
INR	M	Increment memory content	1/3/10	MD	MD	MD	MD	MD
INR	REG	Increment register	1/1/4	MD	MD	MD	MD	MD
JC		Jump conditional — on carry	3/3/7-10					
JM		Jump conditional — on minus	3/3/7-10					
JMP	Addr	Jump unconditional	3/3/7-10					
JNC		Jump conditional — no carry	3/3/7-10					
JNZ		Jump conditional — no zero	3/3/7-10					
JP		Jump conditional — on positive	3/3/7-10					
JZ		Jump Conditional — on zero	3/3/7-10					
LDA,	addr	Load A direct	3/4/13					
LDAX	rp	Load A from M, memory address in BC/DE	1/2/7					
LHLD		Load HL direct	8/5/16					
MOV	Rd, Rs	Move from Rs to Rd register	3/3/10					
MOV	R, data	Move data to register	1/2/7					
MVI	R, data	Load 8-bit data in register	2/2/7					
MVI	M, data	Load 8-bit data in memory	2/3/10					
NOP		No operation	1/1/4					
ORA	R	OR reg with A	1/1/4	MD	MD	O	MD	O
ORI	data	OR 8-bit data with A	2/2/7	MD	MD	O	MD	O
OUT	Port	Output to 8-bit port	2/3/10					
PCHL		Move HL to program counter	1/1/6					
POP	RP	POP register pair	1/3/10					
PUSH	RP	Push register pair	1/3/12					
RAL		Rotate A left through carry	1/1/4					MD
RAR		Rotate A right through carry	1/1/4					MD
RET		Return	1/3/10					

(*Contd.*)

Instruction	Description	B/M/T	S	Z	AC	P	CY
RIM	Read interrupt mask	1/1/4					
RLC	Rotate left	1/1/4					MD
RRC	Rotate right	1/1/4					MD
RM	Return — On minus	1/3/6-12					
RNC	Conditional — Not carry						
RNZ	— Not zero						
RP	— On positive						
RPE	— even parity						
RP O	— odd parity						
RZ N	— on zero						
RST N	Restart						
SBB R	Subtract reg. from A with borrow	1/1/4	MD	MD	MD	MD	MD
SBB M	Subtract Mem from A with borrow	1/2/7	MD	MD	MD	MD	MD
SBI data	Subtract 8-bit from A	2/2/7	MD	MD	MD	MD	MD
SHLD data	Store H-L direct	3/5/16	MD	MD	MD	MD	MD
SIM	Set interrupt mask	1/1/4					
SPHL	Move HL to Stack pointer	1/1/6					
STAX rp	Store A in M, Memory address is in BC/DE	1/2/7					
STA Addr	Store A direct	3/4/13					
STC	Set carry	1/1/4					
SUB R	Subtract reg from A	1/1/4	MD	MD	MD	MD	MD
SUI Data	Subtract 8-bit from A	2/2/7	MD	MD	MD	MD	MD
XCHG	Exchange DE with HL	1/1/4	MD	MD	MD	MD	MD
XRA R	Exclusive OR Reg. with A	1/1/4	MD	MD	O	MD	O
XRA M	Exclusive OR Memory with A	1/2/7	MD	MD	O	MD	O
XRA, data	Exclusive OR 8-bit with A	2/2/7	MD	MD	O	MD	O
XTHL	Exchange Stack with HL	1/4/16					

Example 13-1. *A set of 16-bit reading is stored in memory location starting at 2050H. Each reading occupies two memory locations. The low order byte is stored first, followed by the higher order byte. The number of reading stored is specified by the contents of register B. Write a program to add all the reading and store the sum in the output buffer memory. The maximum limit of sum is 24 bits.*

(*UPSC Engg. Services 2003*)

Solution : The program to do the given task is as under,

Sr. No.	Label	Instruction	Comments
1.		LXI SP,2050H	Stack pointer points to the location 2050H
2.		MVI B,NUMCOUNT	Load number count in register B.
3.		XRA A	Clear accumulator
4.		LXI H,0	Clear HL
5.	Loop :	XCHG	Exchange the contents of H-L register with D-E register

6.	POPH	Load the contents of location specified by stack points to HL pair. Now stack points the new location (previous location + 2)
7.	DAD D	Add contents of HL pair register and DE pair register and store sum into HL pair
8.	ACI 0	Add 0 and carry (if any) to accumulator
9.	DCR B	Decrement the counter (num count)
10.	JNZ : Loop	If contents of register B is not equal to zero then Jump to label loop
11.	SHLD OUTBUFF	Store the contents of HL pair at location OUTBUFF
12.	STA OUTBUFF + 2	Store the MSB at location outbuff + 2
13.	HLT	Stop

The XCHG instruction used in above program facilitates in referencing two memory locations at a time. The instructions 5 to 10 will be executed repeatedly until all the numbers are added (the number count is decided by contents of register B. The lower 8-bit of total sum is stored in location OUTBUFF. The next higher 8-bit of total sum is stored in location OUTBUFF + 1. The most higher 8-bit is stored in location OUTBUFF + 2.

Example 13-2. *A BCD number is stored in memory location 2040H in packed form. Write a program using 8085 assembly language to unpack it and store in memory location 2041H and 2042H. Least significant digit should be stored first.*

(*UPSC Engineering Services — 2000*)

Solution : In BCD code, decimal 0 to 9 are coded by their natural binary equivalent using 4 bits and each decimal digit is represented by an individual bit code. For example, $(15)_{10}$ is represented by $(000\ 1\ 0101)_2$ using a BCD code.

Packed BCD form → $(0001\ 0101)_2 \Rightarrow (15)_{10}$.

Unpacking the BCD number means, separating each decimal digit. So these two numbers will be made to the unpacked form by the following way.

$$\left.\begin{array}{llll} 0000 & 0001 & \longrightarrow & 1 \\ 0000 & 0101 & \longrightarrow & 5 \end{array}\right\} \text{Unpacked BCD form}$$

The program will be written in the assembly language as follows.

Sr. No.	Instruction	Comments
1.	LDA, 2040H	Load accumulator with contents of address 2040H.
2.	MOV B, A	Store the contents of accumulator in register B.
3.	ANI OFH	AND the contents of accumulator with 8-bit data (OFH)
4.	LXI 2041H	HL pair points at location 2041H.
5.	MOV M, A	Store the contents of accumulator in memory location, *i.e.*, 2041H.
6.	MOV A, B	Store the contents of reg. B in accumulator.
7.	ANI FOH	AND the contents of accumulator with 8-bit data (FOH).
8.	RRC	Rotate accumulator to the right.
9.	RRC	Rotate accumulator to the right.
10.	RRC	Rotate accumulator to the right.
11.	RRC	Rotate accumulator to the right.
12.	INX H	HL pair points to location 2042H
13.	MOV M, A	Store the contents of accumulator in memory location *i.e.*, 2042H.
14.	HLT	Stop.

The instruction 1 is used to store the BCD number into accumulator from memory location 2040H. This BCD number is stored in B register after executing the instruction number 2. During the instructions 3, 4 and 5, the BCD number is AND ed with OFH to unpack the lower four digit and stored in location 2041H.

$$\Rightarrow \quad (0001\,0101)_2 \quad \text{AND} \quad (0000\,1111)_2 \quad \Rightarrow \quad (0000.0101)_2$$
$$(15)_{BCD} \quad \text{AND} \quad (OF)_H \quad = \quad (05)_H$$

∴ Number stored in location 2041H = 05H.

In instruction 7 and 8, the BCD number is AND ed with FOH.

$$\Rightarrow \quad (0001\,0101)_2 \quad \text{AND} \quad (1111\,0000)_2 \quad = \quad (0001\,0000)$$
$$(15)_{BCD} \quad \text{AND} \quad (FO)_H \quad = \quad (1\,0)_H$$

The contents of accumulator will be rotated to right for four times after executing the instructions 8 to 11. Here, after execution of instruction 8, contents of accumulator is 0000 1000.

After execution of instruction 9, contents of accumulator is 0000 0100.

After execution of instruction 10, contents of accumulator is 0000 0010.

After execution of instruction 11, contents of accumulator is 0000 0001.

01H will be stored in the location 2042H after executing the instructions 12 and 13.

Example 13-3. *Calculate the delay in the following loop when the system clock period is 30 µsec.*

Label	*Mnemonics*	*T States*
	LXI B, 12FFH	*10*
Delay	*DCX, B*	*6*
	XTHL	*16*
	NOP	*4*
	NOP	*4*
	MOV A, C	*4*
	XRA B	*4*
	JNZ DELAY	*10/7*

(*UPSC Engineering Services—2003*)

Solution : The comments for the above program will be written as mentioned below :

Sr. No.	**Label**	**Instruction**	**Comments**
1.		LXI B,12FFH	Load 12 FFH in register pair BC (B = 12, C = FF)
2.	Delay	DCX, B	Decrement register pair BC
3.		XTHL	Exchange Stack with HL
4.		NOP	NO operation
5.		NOP	NO operation
6.		MOV A,C	Store the contents of reg C in accumulator
7.		XRA, B	Exclusive OR, accumulator's contents with register B
8.		JNZ Delay	If contents of accumulator is nonzero then jump to instruction 2.
9.		HLT	STOP.

The instructions 2 to 8 are executed repeatedly till the contents of register B equals to the contents of register C (i.e. when the contents of register C become 12H). So totally $(237)_{10}$ times the instructions 2 to 8 are executed.

The delay generated by the given program is calculated as below :

Total number of T states

$$= 10 + 237 \times (6 + 16 + 4 + 4 + 4 + 4) + 236 \times (10) + 7$$

$$= 11383 \text{ T states}$$

$$\text{One T-state} = 1 \text{ clock period}$$

$$1 \text{ clock period} = 30\ \mu\text{sec}$$

$$\therefore \text{Delay} = 11383 \times 30\ \mu\text{sec}$$

$$= 341490\ \mu\text{sec}$$

$$\text{Delay} = \mathbf{342} \text{ msec.}$$

Example 13-4. *An 8085 microprocessor polls on 8-bit input port having an address of 03H. If the LSB of the data is high, the microprocessor operates a relay by sending a high MSB signal to an output port with an address 02H. Otherwise the polling continues. Write a program to implement it.*

(*UPSC Engg. Services — 2002*)

Solution : The hardware connect to implement the above task is shown in Fig. 13-19. A logic circuit is used to operate the relay when the MSB bit of output port goes high, the logic circuit will turn ON the transistor and the relay.

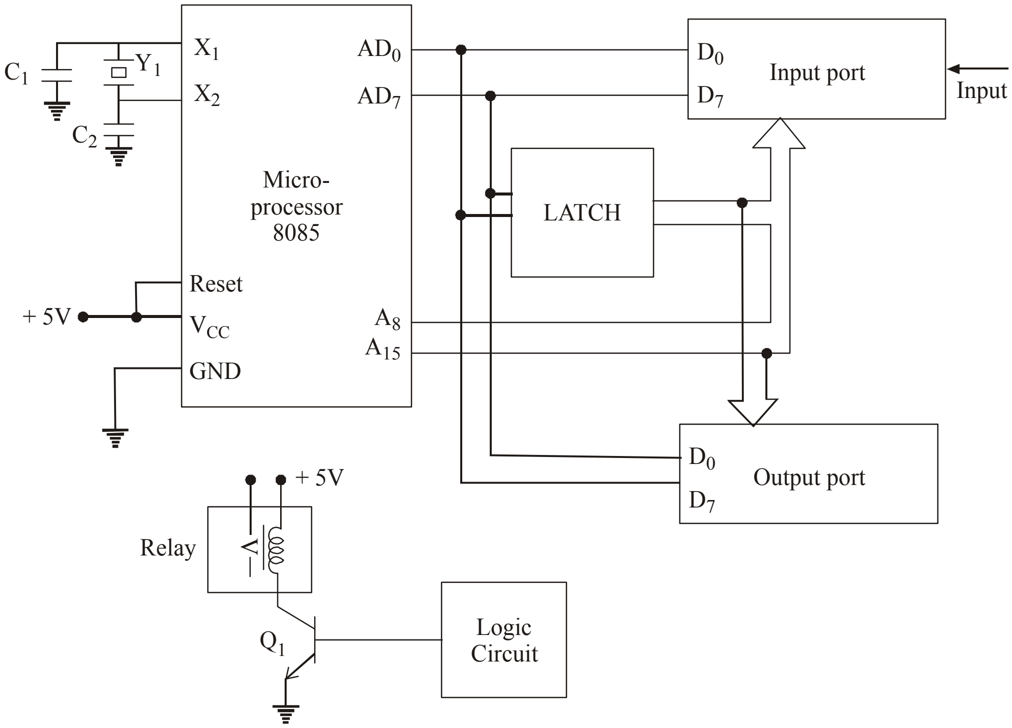


Fig. 13-19.

The program may be written as follows :

Sr. No.	Instruction	Comments
1.	IN 03H	The contents of input port whose address is 03H, is moved to accumulator.
2.	MOV D, 01H	Store 01H in register D.

3.	ANA D	AND the contents of register D with contents of accumulator.
4.	JZ label	If result is zero, jump to label.
5.	MOV 1, 10H	Store 10H in register A.
6.	OUT 02H	Transfer the content of accumulator to the output port whose address is 02H.
7.	HLT	Stop.

Here the instruction number 3 is used to AND the contents of accumulator with 01H data to check whether the LSB from input port is high or low. The fourth instruction is used to polling from input port if LSB is low. As soon as LSB from input port comes high, the instruction numbers 5 and 6 will execute and the microprocessor will be stopped at the instruction number 7.

Example 13-5. *Write a program in assembly language to add four numbers available in the memory locations 2500H, 2501H, 2502H and 2503H. Store the result in the memory location 2510H. Also write an algorithm for solving the problem.*

Solution : The program for adding the four numbers in four different locations will be,

Sr. No.	Label	Instruction	Comments
1.		LXI H, 2500H	H-L pair points at location 2500H.
2.		MOV A,M	Store contents of memory into accumulator.
3.		STC	Clear the array
4.		CMC	Carry flag CY is reset
5.		MOV C, 03H	Store count 03H into register C.
6.	Loop :	INX H	HL pair points at next higher location.
7.		ADC, M	Add memory carry to accumulator.
8.		DCR C	Decrement register C.
9.		JNZ Loop	If result (contents of C) is not zero, then jump to loop location.
10.		LXD H, 2510H	HL pair points at location 2510H.
11.		MOV M, A	Move the contents of accumulator.
12.		HLT	Stop.

The algorithm for the given problem is,

1. Read first number from memory location 2050H and store it in accumulator.
2. Set counter C equal to 03H (initial value).
3. Clear carry *i.e.*, CY = 0.
4. Read next location and store number in M.
5. A = A + M + CY.
6. Decrement count C.
7. If C # 0, then repeat steps 4, 5 and 6.
8. Store contents of accumulator (A) into allocation 2510H.
9. Stop.

To add the four numbers, the counter C is loaded with 03H, *i.e.*, initial value of counter 4-1 = 3. So three times, the steps 4, 5 and 6 are executed.

Example 13-6. *Write a program to sort out the minimum and maximum from a data table, which are stored at locations 2040H, 2041H, 2042H and 2043H. Store the minimum value at location 2050H and maximum at 2051H. Draw a flowchart for the given problem also.*

Solution : The program will be,

Sr. No.	Label	Instruction	Comments
1.		LXI H, 2040H	HL pair points to location 2040H.
2.		MOV C, 04H	C register is set as counter with initial value 04H.
3.		MOV D, M	Move the contents of memory location to D register.
4.		MOV E, M	D register.
5.	Loop 1 :	MOV A,D	Move the contents of register D to register A.
6.		INXH	HL pair points to next location *i.e.*, 2041 H.
7.		MOV B, M	Move second number from memory location to register B.
8.		SUB, B	Subtract second data from first data.
9.		JC Loop 2	Jump to label loop 2 if carry.
10.		MOV A, E	Move contents of register E to accumulator.
11.		SUB B	Subtract contents of register to accumulator.
12.		JNC Loop 3	Jump to label loop 3 if no carry.
13.		JMP Loop 4	Jump to label loop 4.
14.	Loop 2 :	MOV D, B	Move contents of B-register to register D
15.		JMP Loop 4	Jump to Label Loop 4
16.	Loop 3 :	MOV E, B	Move contents of register B to register E.
17.	Loop 4 :	DCR C	Decrement register C
18.		JNZ Loop 1	Jump to label loop 1 if not zero.
19.		LXI H, 2051H	HL pair points to location 2051H.
20.		MOV M, D	Store the contents of register D (Max. value) to location 2051H.
21.		DCX, H	HL pair points to location 2050H
22.		MOV M, E	Store minimum value (contents of register E) to location 2050H.
23.		HLT	Stop.

The flowchart is shown here for the above problem. Let OA, O3, O4, and OB are four numbers that are stored at locations 2040H, 2041H, 2042H and 2043H respectively.

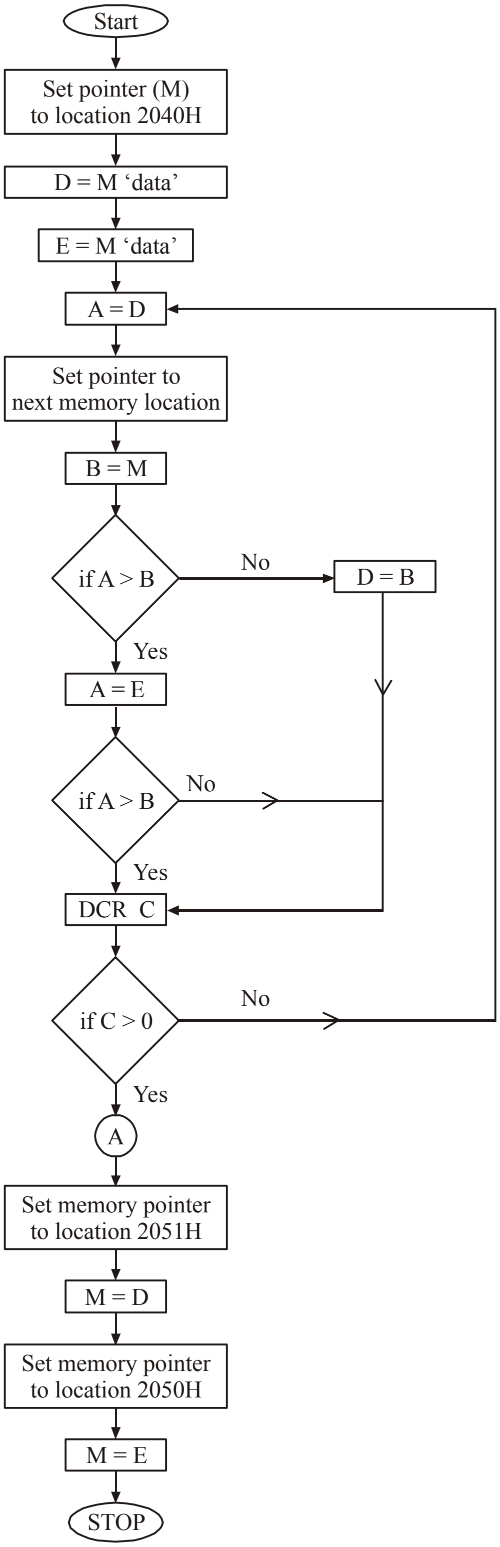

Start
Set pointer (M) to location 2040H
D = M 'data'
E = M 'data'
A = D
Set pointer to next memory location
B = M
if A > B
No
D = B
Yes
A = E
if A > B
No
Yes
DCR C
if C > 0
No
Yes
A
Set memory pointer to location 2051H
M = D
Set memory pointer to location 2050H
M = E
STOP

Fig. 13-20.

Example 13-7 : *Write a short assembly program, without using any arithmetic instruction, to store hexadecimal 1D in the flag register of 8085 microprocessor. Data in other register of the processor must not alter upon executing the program.*

Solution : The flag register of 8085 is shown below,

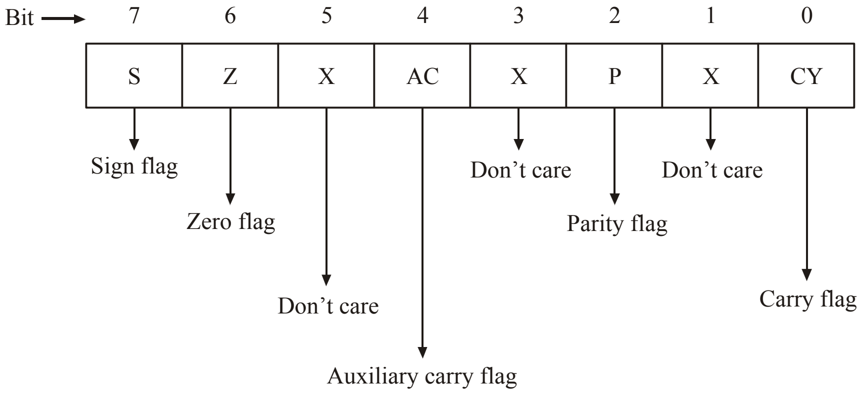


Fig. 13-21

While storing the hexadecimal 1D in flag register by not changing the content of 8085 register, the following flag registers are changed.

Bit 7	Bit 6	Bit 4	Bit 2	Bit 0
S = 0	Z = 0	AC = 1	P = 1	CY = 1

To make the flag AC, P and CY to 1, the following operation should be made :

$$\begin{array}{rrcll} & 1\ 0\ 0\ 0 & 1\ 1\ 0\ 0 & \Rightarrow\ 8C & \rightarrow\ \text{(Assume data 7)} \\ & 1\ 0\ 0\ 1 & 1\ 1\ 0\ 1 & \Rightarrow\ 9D & \rightarrow\ \text{(Assume data 2)} \\ + & \hline 1\ 0\ 0\ 1\ 0 & 1\ 0\ 0\ 1 & \Rightarrow\ 29 & \end{array}$$

The flag status after the above condition will be S = 0, AC = 1, P = 1 and CY = 1. To write the program for the above operation, we will have,

```
LDA,   8CH   ;   Store data 1 (8CH) into accumulator
ADI    9DH   ;   Add data 2 (9D) to accumulator
HLD          ;   Stop.
```

Example 13-8. *In 8085 microprocessor, a number of the form 000 XXXX0 stored in the accumulator is processed by the program. (Assume CY = 0) as follows :*

```
ANI      OFFH
RAL
MOV      B,A
ANI      OFFH
RAL
ANI      OFFH
RAL
ADD      B
```

What is the operation carried out by the above program?

(*UPSC Engg. Services — 2001*)

Solution : Here the program is re-written with comments as below :

Sr. No.	Instruction	Comments
1.	ANI OFFH	AND content of accumulator with FFH.
2.	RAL	Rotate accumulator through carry towards left.
3.	MOV B,A	Store contents of accumulator in register B
4.	ANI OFFH	AND contents of accumulator with FFH.
5.	RAL	Rotate accumulator through carry towards left.
6.	ANI OFFH	AND contents of accumulator with FFH.
7.	RAL	Rotate accumulator through carry towards left.
8.	ADD B	Add the contents of register B in accumulator and store result in accumulator.

In the problem statement, the contents of accumulator given as 000 XXXX 0. Here consider don't care (X) bit as 1.

So now the contents of accumulator before start of the program.

$$A = 00011110 = (30)_{10} \text{ or } (IE)_H.$$
$$CY = 0$$

After executing the first instruction, we have

$$\begin{array}{rcccccccccl} & 0&0&0&1&&1&1&1&0 & \rightarrow (IE)_H \\ \text{Logical AND} & 1&1&1&1&&1&1&1&1 & \rightarrow (FF)_H \\ \hline A = & 0&0&0&1&&1&1&1&0 & \end{array}$$

After executing the second instruction,

$$\begin{array}{ccl} CY & \overbrace{\begin{array}{ccccccccc}0&0&0&1&&1&1&1&0\end{array}}^{A} & \\ 0 & \begin{array}{ccccccccc}0&0&1&1&&1&1&0&0\end{array} & \leftarrow \text{RAL} \\ \hline & A = \begin{array}{ccccccccc}0&0&1&1&&1&1&0&0\end{array} & \end{array}$$

After the third instruction, we have

$$B = A = 00\ 11\ 11\ 00.$$

After executing instruction 4,

$$\begin{array}{rccccccccc} & 0&0&1&1&&1&1&0&0 \\ \text{Logical AND} & 1&1&1&1&&1&1&1&1 \\ \hline A = & 0&0&1&1&&1&1&0&0 \end{array}$$

After executing instruction 5, we have

$$\begin{array}{ccl} CY & \overbrace{\begin{array}{ccccccccc}0&0&1&1&&1&1&0&0\end{array}}^{A} & \\ 0 & \begin{array}{ccccccccc}0&1&1&1&&1&0&0&0\end{array} & \leftarrow \text{RAL} \\ & A = \begin{array}{ccccccccc}0&1&1&1&&1&0&0&0\end{array} & \end{array}$$

After executing the instruction 6,

$$\begin{array}{rccccccccc} & 0&1&1&1&&1&0&0&0 \\ \text{Logical AND} & 1&1&1&1&&1&1&1&1 \\ A = & 0&1&1&1&&1&0&0&0 \end{array}$$

After execution of instruction 7,

$$\begin{array}{ll}
\text{CY} & \qquad\qquad\text{A} \\
0 & \overbrace{0\;1\;1\;1\quad 1\;0\;0\;0} \\
0 & 1\;1\;1\;1\quad 0\;0\;0\;0 \;\leftarrow \text{RAL} \\
\text{A} = & 1\;1\;1\;1\quad 0\;0\;0\;0
\end{array}$$

After execution of instruction 8, we have

$$\begin{array}{rl}
\text{A} \rightarrow & 1\;1\;1\;1\quad 0\;0\;0\;0 \\
\text{B} \rightarrow & 0\;0\;1\;1\quad 1\;1\;0\;0 \\
+ & \overline{\hphantom{1\;1\;1\;1\quad 0\;0\;0\;0}} \\
\text{A} \rightarrow \underset{\underset{\text{CY}}{\downarrow}}{1} & 0\;0\;1\;0\quad 1\;1\;0\;0 = \text{12CH} = (300)_{10}
\end{array}$$

At the completion of execution of the program, it is seen that the result is $(300)_{10}$, which is 10 times multiplication of contents of accumulator before start of program. Thus this program does the multiplication of accumulator content by 10.

Example 13-9. *Write a program in 8085 assembly language to store the contents of its flag register in the memory location 2000H.*

(*UPSC Engg. Services — 2000*)

Solution : The contents of the flag register can be accessed along with the accumulator contents by a single processor status word as shown below,

$$\underbrace{\boxed{A_7}\boxed{A_6}\boxed{A_5}\boxed{A_4}\boxed{A_3}\boxed{A_2}\boxed{A_1}\boxed{A_0}}_{\text{Accumulator}}\underbrace{\boxed{S}\boxed{Z}\boxed{X}\boxed{AC}\boxed{X}\boxed{P}\boxed{X}\boxed{CY}}_{\text{Flag register contents}}$$

Fig. 13-22. Organisation of PSW.

The program to store the contents of flag register at location 2000H may be written as below :

Sr. No.	Instruction		Comments
1.	PUSH PSW	;	push the contents of program status word at stack
2.	XTHL	;	exchanging the top of stack with HL pair
3.	XCHG	;	Exchanging the contents of HL pair with DE register.
4.	LXIH, 200H	;	HL pair points to location 2000H.
5.	MOV M, E	;	Store status of flag register at location 2000H.

After the execution of instructions 1 and 2, the following registers will have,

H → Contents of accumulator
L → Contents of flag register

After the execution of instruction 3, the following registers will have

D → Contents of accumulator
E → Contents of flag register.

After the execution of instructions 4 and 5, the contents of flag register will be at 2000H location.

13-20. 8086 Microprocessor

The Intel 8086 is a 16-bit microprocessor that has both 8-bit and 16-bit attributes. In an 8-bit microprocessor, speed of operation, direct memory addressing, data handling capacity, etc. are very limited to use. But the 16-bit microprocessors have higher speed of operation than 8085 microprocessors and can be used with high level languages. The 8086 has a 20-bit address bus, that can access up to 2^{20} memory locations. That means it is capable of addressing 1 megabytes of memory. The other main important features of 8086 microprocessors are :

1. 8 and 16 bit signed and unsigned arithmetic operations
2. Multiply and divide instructions
3. Fourteen 16-bit registers
4. 24 operand addressing modes
5. Powerful instruction sets.

The Internal block diagram of 8086 microprocessor is shown in Fig. 13-23. Basically it is divided into two blocks, namely, Bus Interface Unit (BIU) and Execution Unit (EU). The concept of dividing the block is to increase the speed of the operation of a processor.

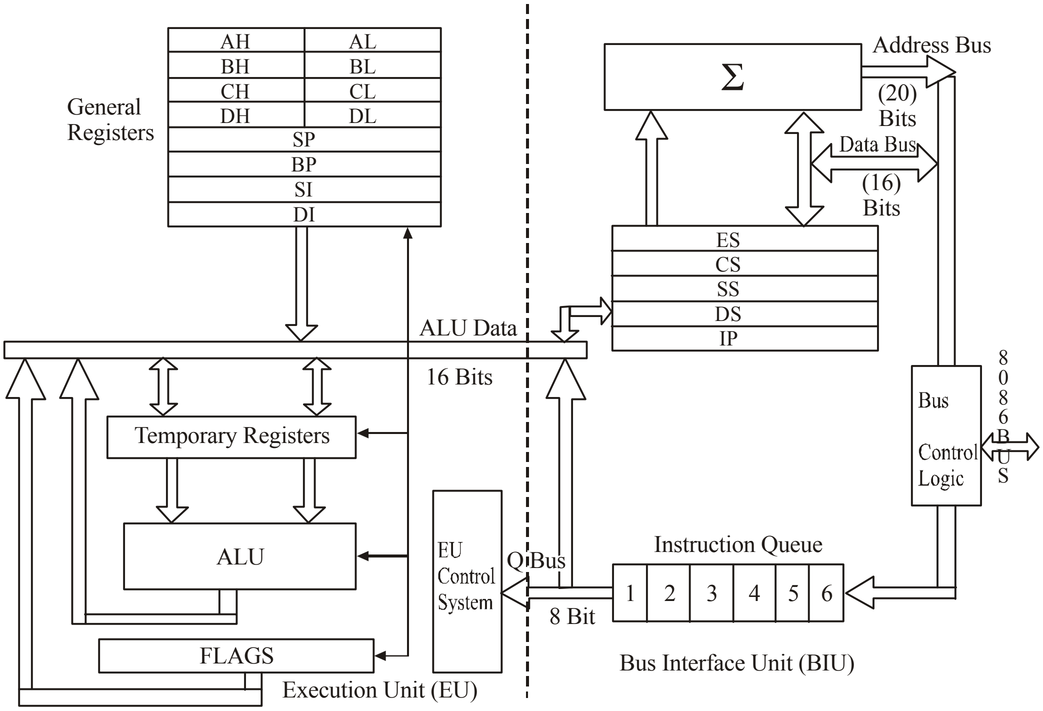


Fig. 13-23. Internal block diagram of 8086.

Here the Bus Interface Unit (BIU) performs all the bus operations such as instruction fetching reading and writing operands for memory and calculating the addresses of the memory operands. Then the instruction bytes are transferred to the instruction queues to execute. The Execution Unit (EU) executes the instructions from the instruction system queue. So both the units operate asynchronously to give the processor an overlapping instruction fetch and execution mechanisms which is called pipelining. This results in efficient use of the system bus and its performance. Normally the BIU contains instruction queue, segment registers, instruction pointer and address adder. The Execution Unit (EU) contains control circuitry, instruction decoder, ALU, pointer, index register and flag register.

The 8086 processor contains four 16-bit general purpose registers : AH, BH, CX and DX, which are used as operands in arithmetic and logical operation. Each of these registers can be combined as eight bit registers like AH & AL, BH & BL, CH & CL and DH & DL. The AH register is called as the accumulator. The next four 16-bit registers are SP (Stack pointer), BP (base pointer), SI (source index) and DI (Destination index). The stack pointer register points to the top of the active Stack segment. In 8086 base system, there can be several stack segments in the memory. But only one of them is active at any given instant. BP, SI and DI registers are new tie additions compared to the 8085 microprocessor and are used to hold the 16 bit offset of a data word in data segment (DS) register or extra-segment (ES) register.

The BIU unit contains four 16-bit segment registers, namely, the code segment (CS) register, the stack segment (SS) register, the extra segment (ES) register and the data segment (DS) register. The segment registers are combined with memory pointers to form a 20-bit memory address. The default combination is shown in Table 13-8.

Table 13-8. Default combination with segment register

Segment Registers	Offset register
Code Segment (CS)	Instruction pointer
Stack Segment (SS)	Stack pointer (SP) and base pointer (BP)
Data Segment (DS)	Source index, Destination index and B_x register
Extra Segment (ES)	Source index (SI), Destination index (DI) and B_x register

The memory address in 8086 microprocessor based system can be specified as three formats, namely, physical, logical and offset. Here the physical address is a 20-bit address, the offset address is the address of an instruction in a segment register and the logical address is the combination of a segment and an offset address. Instruction pointer (IP) register is a 16-bit register, analogues to program counter in 8085 microprocessor. It holds the address of next byte of instruction code within the code segment registers. The value contained in register IP is added to 20-bit base address from the CS register and then a 20-bit physical address will be generated.

The 8086 microprocessor has nine flags. Out of these nine flags, five flags are same as that of 8085 microprocessor and four are new additions. These flags are divided into two groups, namely, six data flags and three control flags. The data flags are :

1. Carry flag (CF)
2. Parity flag (PF)
3. Auxiliary Carry flag (ACF)
4. Zero flag (ZF)
5. Sign flag (SF)
6. Overflow flag (OF).

The overflow flag is used for sign purposes (*i.e.*, when the result of a signed number operation is too large causing the most significant bit to overflow into the sign bit. At that time this flag is set). The three control flags are :

1. Trap flag (TF) : It is used for single stepping instructions.
2. Interrupt flag (IF) : This flag is used to enable/disable external maskable interrupt requests.
3. Direction flag (DF) : It is used to control increment/decrement of string operations.

The instruction sets of 8086 microprocessor are divided into six categories :

(1) Data transfer instructions

(2) Arithmetic instructions

(3) Logical instructions

(4) Branch instructions

(5) String manipulation instructions

(6) Machine or processor control instructions.

Out of these six categories, five are very familiar to us since we have covered in 8085 microprocessor and only new instruction set is string manipulation.

The basic concept of programming for 8086 is same as 8085 processor. In addition to the powerful instruction set, the chip is designed for modular programming which is highly desirable for high level languages.

13-21. Pin Description of 8086

The 8086 is a 16-bit 40-pin microprocessor. Fig. 13-24 shows the pin connection details of 8086. Fig. 13-24(*a*) shows the condition when $\mathrm{MN}/\overline{\mathrm{MX}}$ pin is connected to V_{CC} (High). This configuration is called as Minimum mode. The minimum mode is used for single processor environment. Fig. 13-24(*b*) shows the condition when $\mathrm{MN}/\overline{\mathrm{MX}}$ pin is connected to ground (Low). The configuration is called as maximum mode. This mode is used for multiprocessor environment such as having a coprocessor in a system. In the maximum mode, eight pins are assigned different functions compared to that of minimum mode as shown in Table 13-9.

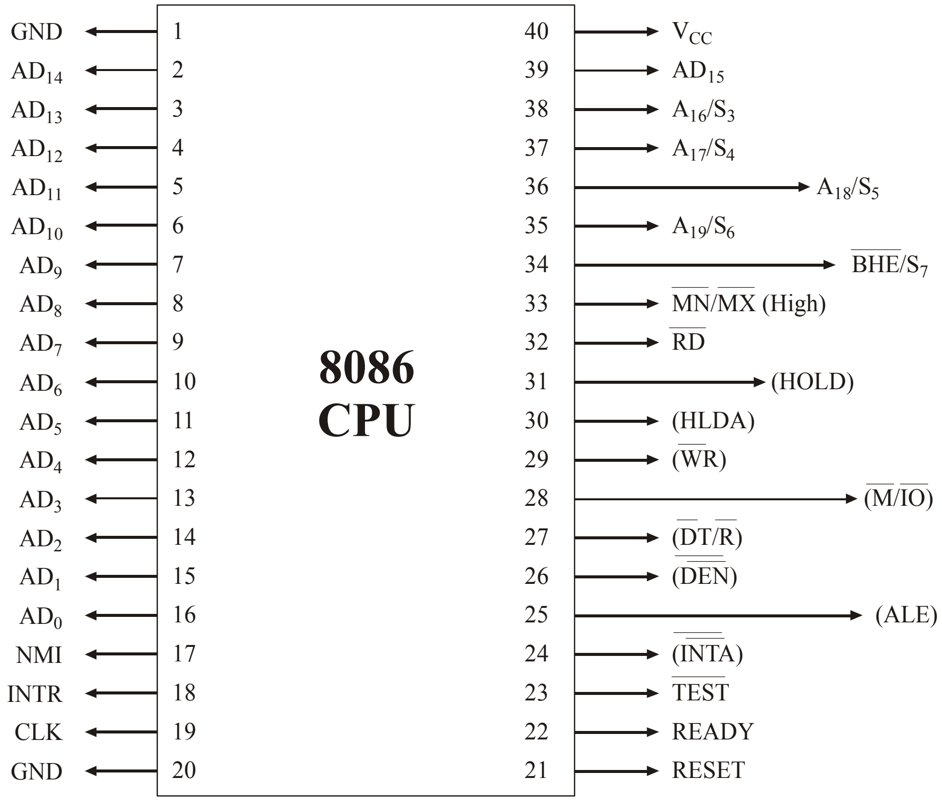


Fig. 13-24(*a*). Pin Diagram of 8086 — Minimum mode.

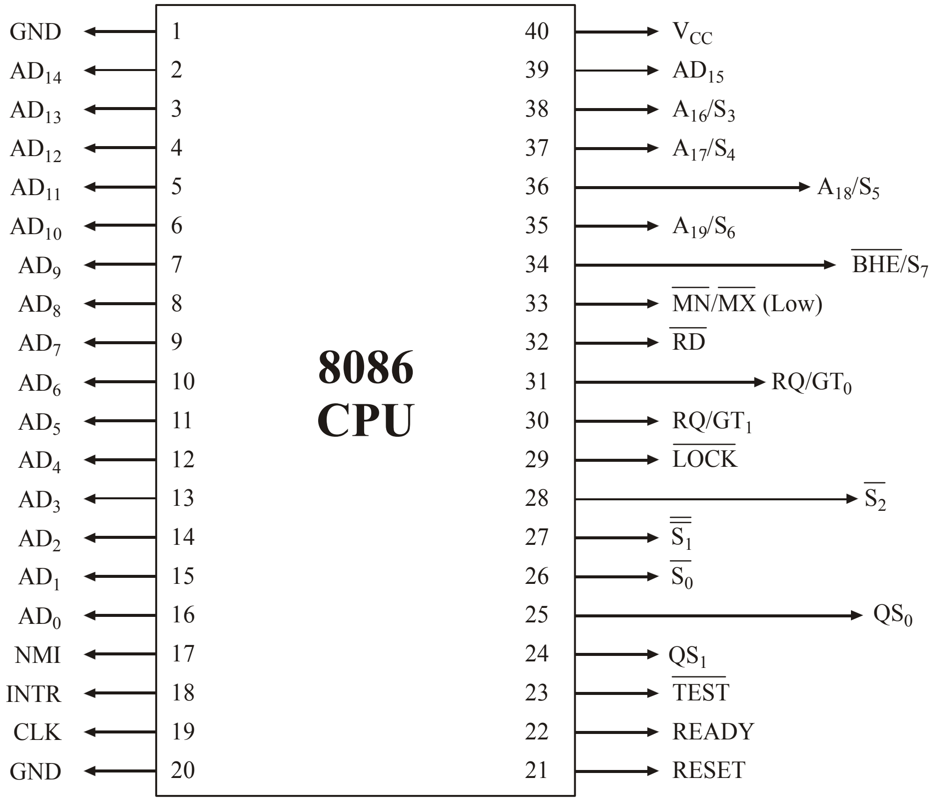


Fig. 13-24(*b*). Pin Diagram of 8086 — Maximum mode.

Table 13-9. Minimum and maximum mode control signals

PIN	Minimum mode function	Maximum mode function
24	$\overline{\text{INTA}}$: Interrupt acknowledge	$\overline{\text{QS}_1}$: Queue status signal
25	ALE : Address latch enable	$\overline{\text{QS}_0}$: Queue status signal
26	D$\overline{\text{E}}$N : Data enable	$\overline{\text{S}_0}$: Machine cycle status
27	DT / $\overline{\text{R}}$: Data transmit/receive	$\overline{\text{S}_1}$: Machine cycle status
28	$\overline{\text{IO}}$ / M : Memory or input/output device selection	$\overline{\text{S}_2}$: Machine cycle status
29	$\overline{\text{WR}}$: Write	$\overline{\text{LOCK}}$: Bus hold control
30	HLDA : Hold acknowledge	RQ / $\overline{\text{QT}}_0$: Local bus priority control
31	HOLD : Hold request	RQ / $\overline{\text{QT}}_1$: Local bus priority control

The description and pin assignments for other pins are as follows :

AD_0-AD_{15} : These pins are bidirectional pins and represented as data/address bus.

A_{16}/S_3, A_{17}/S_4 : These pins are unidirectional pins and used as output signals. These are address/segment identifier pins.

A_{18}/S_5, A_{19}/S_6 : These pins are represented as address/interrupt enable status pins.

BHE/S_7 : It is high order byte/status signal. It is an output pin and used as status line to decode current cycle.

$\overline{RD}$: It is a read control pin

Ready : It is a wait state request pin. It is an input pin.

Test : This pin is used to synchronize operations of multiple processor in a system. It is an output pin.

INTR : It is an output pin used for interrupt request.

NMI : It is a non-maskable interrupt pin.

The 8086 microprocessor is capable of addressing one megabyte of memory. Various versions of this processor can operate with clock frequency from 5 MHz to 10 MHz.

SUMMARY

In this chapter, you have learned that

1. The 8085 microprocessor has 8-bit address bus and 8-bit address/data bus.
2. Arithmetic logic unit is the heart of microprocessor. It performs all arithmetic and logical operations.
3. The result of ALU operation stores in accumulator.
4. 8085 microprocessor has six registers B, C, D, E, H and L and two temporary registers W, Z. All these registers are 8 bit length. B-C, D-E and H-L can be used together to hold 16-bit data.
5. 8085 has 3 sets of communication lines — address bus, data bus and control bus.
6. Program counter contains the address of the instruction to be executed next.
7. 8085 microprocessor has five interrupts. TRAP has the highest priority.
8. 8085 microprocessor has 5 flags, S, Z, AC, P and CY. These flags are set or reset according to the result of operation.
9. The flag of microprocessor indicates internal status of ALU.
10. Stack pointer points to the address of top of the stack.
11. Both stack pointer and program counter are 16-bit register.
12. RST 7.5, RST 6.5, RST 5.5 and TRAP are the vectored interrupts. INTR is non-vectored interrupt and require external hardware logic circuit.
13. Data transfers operation does not affect the flags.
14. Interrupt control ask the microprocessor to temporarily stop the current process and the unexpected event will be serviced.
15. DMA makes the direct access between memory and microprocessor to speed up the data transfer.
16. To operate 8085 microprocessor at 3 MHz, a crystal of 6 MHz is used.
17. A microprocessor chip alone is of no use. It is worth with programming instructions only.
18. 8085 microprocessor has three 16-bit registers whereas 8086 contains thirteen 16-bit general purpose registers.
19. Register A cannot be combined with any register to form a register pair.

20. The main function of assembler is to convert the source code program to machine codes.

21. An algorithm is a step-by-step procedure to find out the solution for a specific task.

22. An instruction requires minimum of one fetch cycle and one execute cycle.

23. The first byte of a 3-byte instruction is always opcode.

GLOSSARY

Microprocessor : An integrated circuit which form a CPU of a microcomputer.

Accumulator : 8-bit register in a microprocessor in which the results of most of the arithmetic and logic operations are stored.

ALU (Arithmetic Logic Unit) : This is the area of the microprocessor where various computing functions are performed on data.

Assembler : A computer software that converts assembly language program into machine codes.

Architecture : The specific interconnections of registers and logic blocks.

Flag : A single bit register in microprocessor which has critical importance in decision-making process.

General purpose register : A register that can be used for temporary data storage. It is also known as scratch pad register.

Interrupt : A signal which causes the microprocessor to halt the work it is doing in order to perform a high priority unexpected event.

Instruction : A binary pattern designed inside a microprocessor to perform a specific function.

Instruction Set : A set of instructions allowed in microprocessor. It is specified by the manufacturer of the particular microprocessor.

Instruction Register : A register which holds the instruction being executed currently by the manufacturer.

Instruction Cycle : The sum of fetch cycle and execution cycle.

I/O devices : The devices used for communication between microprocessor and the outside world.

Mnemonic : A symbolic code for each instruction.

Machine language : A programming language that a microprocessor can directly understand. It is in the form of 0s and 1s.

Machine Cycle : A group of T-states or clock-pulses required to complete the specific task.

Opcode (Operation Code) : It is the command code in the instruction that serves for a particular operation.

Operand : The data that is operated on.

Port : The hardware circuit used to transfer data in and out of microprocessor.

Peripheral : The input/output and storage devices of a microprocessor.

Program : A sequence of instructions written for a microprocessor to perform a certain task.

Program Counter : A 16-bit register in microprocessor that contains the address of next instruction to be executed.

Stack : A set of adjoining memory locations used in last-in-first-out (LiFo) configuration.

Stack pointer : A 16-bit register in microprocessor whose contents point to the top of the stack.

Word : A group of bits that a microprocessor processes at a time.

Write : The process of storing information into the memory.

DESCRIPTIVE QUESTIONS

1. Describe a microprocessor based system.
2. Explain timing and control unit of 8085 microprocessor. Which pins are used for this kind of operation?
3. Point out the special features of an accumulator.
4. How is the working of $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$ coordinated?
5. Draw a pin diagram of 8085. Explain the functions of each pin.
6. Mention the hardware interrupts in 8085 microprocessor in the order of their priority and mention about the address branched when the interrupt occurs.

   (*UPSC Engineering Services — 2001*)
7. Explain what happens in 8085 processor based system when the microprocessor receives HOLD signal.

   (*UPSC Engineering Services — 2000*)
8. Write three microprocessor instructions used for the memory location called stack.

   (*I.E.S. Electrical Engg. — 1997*)
9. Differentiate between program counter and stack pointer.
10. How many 16-bit general-purpose registers are available in 8085 microprocessor?
11. Explain the difference between instruction cycle and machine cycle.
12. List all the instructions of branch group of 8085 microprocessor and give brief description of any two.
13. Explain the function of SID and SOD pins.
14. Explain the functions of flags in 8085 microprocessor. How many flags does 8085 have?

    (*UPSC Engineering Services — 2002*)
15. Explain, with example, the use of RST 6, 8085 instruction.

    (*UPSC Engineering Services — 2002*)
16. What are the advantages of 8086 over 8085? What are the limitations of 8086 ?

    (*UPSC Engineering Services — 2002*)
17. Draw the architecture of Intel 8086 and mention the special functions associated with its registers.

    (*UPSC Engineering Services — 2000*)
18. Write three microprocessor instructions used for the memory location called stack.

    (*IES Electrical Engg. — 1977*)
19. Draw diagrams showing signals in minimum and maximum mode of 8086 microprocessor.
20. Explain SIM and RIM instruction of 8085.

    (*UPSC E & T Engg., — 2000*)
21. Draw the internal architecture of the Intel 8085 microprocessor. Identify the parts.

    (*Gauhati University, 2007*)
22. What role do the following perform?
    Program counter and Stack pointer.

    (*Gauhati University, 2007*)
23. What do the following signify?
    AD0-AD7, RST 7.5, HOLD, HLDA, SOD

    (*Gauhati University, 2007*)

24. Draw the flag register. Identify each flag and mention their roles.
(*Gauhati University, 2007*)

25. What do the following ICs perform 8255 and 8257.
(*Gauhati University, 2007*)

26. Write an assembly level language program to add two numbers.
(*Gauhati University, 2007*)

27. Write an assembly level language program to add ten data in consecutive memory locations.
(*Gauhati University, 2007*)

28. Write the importance of stack.
(*Gauhati University, 2007*)

29. Write short notes on

(*a*) RS 232C
(*b*) 8255 as a PPI
(*c*) Microcontroller
(*d*) Interrupt signals
(*e*) Status signals of the 8085
(*f*) Difference between 8085 and 8086 microprocessors

(*Gauhati University, 2007*)

30. Compare serial and parallel address.
(*Mahatma Gandhi University, Dec. 2007*)

31. Write short note on ALU.
(*Nagpur University, 2004*)

32. Explain the following terms

(*i*) Memory Cell
(*ii*) Memory Word
(*iii*) Byte Capacity
(*iv*) Density
(*v*) Address
(*vi*) Read Operation
(*vii*) Write Operation Access time

(*GBTU/MTU, 2005-06*)

TUTORIAL PROBLEMS

1. Write a program in assembly language of 8085 to only complement the upper nibble of accumulator. (*UPSC Engg. Services — 1998*)

2. In the following instruction sequence determine the contents of the HL and DE pairs after the execution of the DAD instruction.

```
LXI B, 2100H
LXI D, 0200H
LXI SP, 2700H
PUSH B
PUSH D
LXI H, 0100H
XTHL
DAD
HLT
```

(*UPSC Engg. Services — 2001*)

3. Give meaning — operand, number of bytes, machine cycle and T-states for the following opcode used in 8085 architecture :

LDA JMP, POP

(*UPSC Engg. Services — 2001*)

4. The following program is running on 8085 microprocessor :

Memory address in HEX	Instruction
2000	LXI SP, 1000
2003	PUSH H
2004	PUSH D
2005	CALL 2050
2008	POP H
2009	HLT

Determine the contents of program counter (PC) and stack pointer (SP) at the completion of execution of the program.

(*GATE E & T — 1992*)

5. An 8085 microprocessor uses a 2 MHz crystal. Find the time taken by it to execute the following delay subroutine, inclusive of the call instruction in the calling program.

```
Calling program        DELAY        PUSH PSLU
...............................                     MVI A, 64H
CALL                   DELAY LOOP   NOP
                                    DCR A
                                    JNZ LOOP
                                    POP PSLU
                                    RET
```

You are given that a CALL instruction takes 18 cycles of the system clock, PUSH requires 12 cycles and a conditional jump takes 10 cycles if the jump is taken and 7 cycles if it is not. All other instructions used above take (3n + 1) clock cycles, where n is the number of access to the memory, inclusive of the opcode fetch. (*GATE E & T — 1997*)

6. What do the following instructions do? What happens if RET is removed from location 2106?

```
2000     Start     MVI A, CO
2002               SIM
2003               CALL 2100
2006               MVI A, 40
2008               SIM
2009               CALL 2107
200C               JMP Start
2100               MVI B, 2A
2102     MOON :    DCR B
2103               JNZ MOON
2106               RET
2107               MVI C, 9F
2109               DCR C
210 A    Sunny :   JNZ Sunny
210 D              RET
```

(*UPSC Engg. Services, 2001*)

7. Consider the execution of the following instructions by a 8085 microprocessor.

LXI H, OIFF H

SHLD, 2050H

Determine the contents of HL pair register and memory location. 2050H and 2051H after the execution?

(*UPSC Engg. Services, 2002*)

8. In a 8085 microprocessor, the following sequence of instruction is executed :

```
STC
CMC
MOV A, B
RAL
MOV B, A
```

What is the function carried out by above program?

(*UPSC Engg. Services, 1998*)

9. Explain what will happen if instructions STC and CMC are removed from program in tutorial problem number 8.

10. Write a 8085 microprocessor program to multiply the data $(05)_H$ by number $(07)_H$.

MULTIPLE CHOICE QUESTIONS

1. The first machine cycle of an instruction is always

(*a*) A memory read cycle (*b*) A fetch cycle

(*c*) An I/O read cycle (*d*) A memory write cycle

(*UPSC Engineering Services, 2003*)

2. In 8085 microprocessor, the value of the most significant bit of the result following the execution of any arithmetic or Boolean instruction is stored in

(*a*) The carry status flag (*b*) The auxiliary carry status flag

(*c*) The sign status flag (*d*) The zero status flag

(*UPSC Engineering Services, 2003*)

3. In a microprocessor when a CPU is interrupted, it

(*a*) Stops executions of instructions

(*b*) Acknowledges interrupt and branches of subroutine

(*c*) Acknowledges interrupts and continues

(*d*) Acknowledges interrupts and waits for the next instruction from the interrupt device.

(*UPSC Engineering Services, 2003*)

4. Consider the following registers :

(*a*) Accumulator and B register (*b*) B and D register

(*c*) C and E register (*d*) H and L register

Which of these 8-bit registers of 8085 microprocessor can be paired together to make a 16-bit register ?

(*UPSC Engineering Services, 2002*)

5. Consider the following instructions of 8085 microprocessor :

(*a*) MOV M,A (*b*) CMP M (*c*) MVI A, FF

Which of these causes the change in status of flag(s)?

(*UPSC Engineering Services, 2002*)

6. The computer program which converts statements written in high language to object code is known as

(*a*) Assembler (*b*) Compiler
(*c*) Disassembler (*d*) Operating System

(*UPSC Engineering Services, 2002*)

7. In the 8086 instruction ADD DX [BX], [CI], the addressing mode of source operand is

(*a*) Register (*b*) Register Indirect (*c*) Based indexed (*d*) Direct

(*UPSC Engineering Services, 2002*)

8. A microprocessor has 24 address lines and 32 data lines. If it uses 10 bits of opcode the size of its memory buffer register is

(*a*) 22 bits (*b*) 24 bits (*c*) 32 bits (*d*) 14 bits

(*UPSC Engineering Services, 2002*)

9. The contents of accumulator in 8085 microprocessor are altered after the execution of the instruction?

(*a*) CMP C (*b*) CPI 3A (*c*) ANI SC (*d*) ORA A

(*UPSC Engineering Services, 2002*)

10. Which one of the following flags is not used for branch operations in a microprocessor?

(*a*) Carry flag (*b*) Auxiliary carry flag (*c*) Overflow flag (*d*) Parity flag

(*UPSC Engineering Services, 2002*)

11. The frequency of driving network connected between pins 1 and 2 of a 8085 chip must be

(*a*) equal to the desired clock frequency (*b*) twice the desired clock frequency
(*c*) Four times the desired clock frequency (*d*) eight times the desired clock frequency

(*UPSC Engineering Services, 2000*)

12. In 8086, if the content of the code segment register is 1FAB and the content of the 1P register is 10A, then the effective memory address is

(*a*) IFBCO (*b*) 304C (*c*) FDB5 (*d*) 20B51

(*UPSC Engineering Services, 2000*)

13. To have the multiprocessing capabilities of the 8086 microprocessor, the pin connected to the ground is

(*a*) $\overline{\text{DEN}}$ (*b*) ALE (*c*) INTR (*d*) $\text{MN}/\overline{\text{MX}}$

(*UPSC Engineering Services, 1999*)

14. An interrupt in which the external devices supplies its address as well as the interrupt request, is known as

(*a*) Vectored Interrupt (*b*) Maskable Interrupt
(*c*) Polled Interrupt (*d*) Non-Maskable Interrupt

(*UPSC Engineering Services, 1998*)

15. In a multiprocessor configuration, sets two processors are connected to the host 8086 processor. The two co-processor instruction

(*a*) must be same (*b*) must overlap
(*c*) must be disjoint (*d*) must be same as the host

(*UPSC Engineering Services, 1997*)

16. For 8085 microprocessor, the instruction RST6 restarts subroutine at address

(*a*) OOH (*b*) 03H (*c*) 30H (*d*) 33H

(*IES Electrical Engg., 2003*)

17. Which logical operation is performed by ALU of 8085 to complement a number?

(*a*) AND (*b*) NOT (*c*) OR (*d*) X-OR

(*IES Electrical Engg., 2002*)

18. The number of output pins of 8085 microprocessor are

(*a*) 40 (*b*) 27 (*c*) 21 (*d*) 19

(*IES Electrical Engg., 2002*)

19. Control bus is used for transmitting and receiving control signals between

(*a*) processor and keyboard (*b*) processor and various devices

(*c*) processor and memory (*d*) input devices and memory

(*IES Electrical Engg., 2000*)

20. A program access ten continuous memory locations. The CPU will take less time if the addressing mode used is

(*a*) direct (*b*) implicit (*c*) register indirect (*d*) immediate

(*IES Electrical Engg., 1999*)

21. The highest priority in 8085 microprocessor system is

(*a*) RST 7.5 (*b*) RST 6.5 (*c*) INTR (*d*) TRAP

(*IES Electrical Engg., 1998*)

ANSWERS

1. (*b*)	**2.** (*a*)	**3.** (*b*)	**4.** (*d*)	**5.** (*b*)	**6.** (*a*)
7. (*d*)	**8.** (*d*)	**9.** (*a*)	**10.** (*b*)	**11.** (*b*)	**12.** (*d*)
13. (*d*)	**14.** (*c*)	**15.** (*a*)	**16.** (*c*)	**17.** (*b*)	**18.** (*b*)
19. (*b*)	**20.** (*d*)	**21.** (*d*)			

14

DIGITAL SYSTEM DESIGN USING VHDL

OUTLINE

Objectives

Upon completion of this chapter, you should be able to

- ❍ know the digital system design process
- ❍ understand the considerations while designing digital system
- ❍ list the software tools used for designing digital systems
- ❍ describe the VHDL design flow

14-1. Introduction

Nowadays digital systems have become part of our life. Almost all our daily needs are served by one or the other digital system. But the design of the digital system or the integrated circuit technology, which is having millions of activities, is a daunting task. As these digital systems have become more complex, detailed design of the systems at the gate and flip-flop levels have become very tedious and time-consuming. To ease these difficulties computer-aided design (CAD) tools are used. Using these tools, the digital system can be designed and debugged at a higher level before conversion to the gate and flip-flop levels. Normally these system designs may involve many designers and may take many man-years to develop a particular product. But during the recent years, as compitition and requirements are growing very rapidly, the necessity to finish the systems at the given budget and timeframe is very much needed. So use of these computer-aided design tools to do the designs is becoming more widespread. Programming of these CAD tools is very easy and the languages used in these tools are more or like our high-level languages such as C and CH. Using these CAD tools, we can automate the entire processes like development, simulation and testing. In the market, there are many development tools available to design the digital system. Over those tools, VHDL and Verilog are the most popular among many companies in the world.

14-2. Digital System Design Process

For the initial design process, a designer develops several design steps before a hardware implementation is obtained. Here most care is taken while developing each step of the digital system design. During each step, the designer checks the result of the last transformation, and if necessary, add more details into it for the next step. To ensure the quality of the product, the design process will be well specified with rigorous checks at each step. Because if the problems are found at later stage, it would be very expensive to correct at later stage. The generalized design and production process of the digital system is shown as a flowchart in Fig. 14-1.

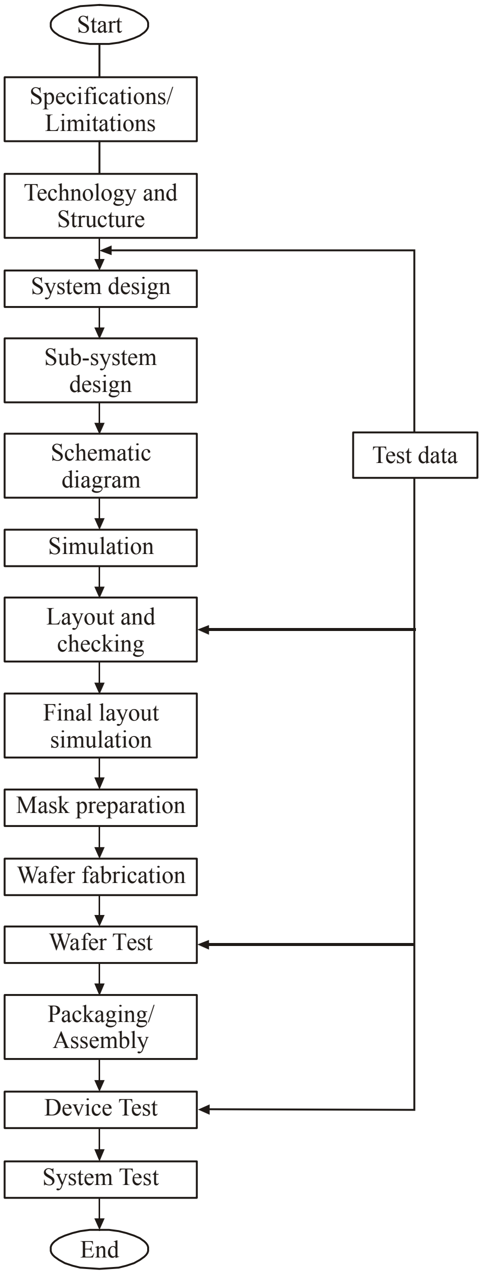


Fig. 14-1. Flowchart of design system.

Initially the design starts with specifications of the system, that consists of functionality of the design, inputs, outputs, size, package type, power consumption, number of devices to be manufactured, its lifetime and so on. This will decide the technology to be used to design, manufacture, testing considerations and so on. Because of the factors mentioned above and for the careful consideration of economic factors, the technology and the structure will be finalized. The technology also will be based on whether the circuit/system is designed using digital, analog or mixed. The overall design is divided into many small sub-blocks for easy development and testing. During the system design, the test scenarios for the blocks are identified and they will be tested thoroughly. During the initial design, drawings of these schematic circuits are finalized and from that the circuit netlists are created. This netlist gives the connection details of the entire schematic designs. After this, a first level of simulation with the schematic design will be done. At the end of the simulation, basic functionality of the circuits are tested. During this simulation, timing calculations are not considered and all the possible combinations of testings are carried out. If the designer finds any problem during this kind of simulation, then the necessary changes on the Schematic will be done to get a correct output. Once the designer finalizes these kinds of testings, then the next stage will be circuit layout design. During the layout process, the schematic drawings of circuits are taken into mask specifications and these will be used in the fabrication process. Since these processes are highly automated, after this step there should not be any further modifications. During the layout process also, there are some rules about layout dimensions to follow. The next stage will be to regenerate the schematic diagrams of the digital circuits. During this phase, final verifications of the circuit will be done. Here additional test data will be inserted into the circuit and all the possible combinations of test cases will be tested. After the post-layout simulation, the layout data will be used to generate the physical masks for the wafer fabrication process. After the final fabrication process, these digital systems will be used for various applications.

14-3. Design Considerations of Digital System

During the design of digital systems, from initial specification levels to final fabrication levels, CAD tools are used. But during the design process, there are some limitations/considerations to follow through. Some of the limitations/considerations are as listed below :

1. Technology and architecture
2. Integration of components
3. Packaging and fabrication
4. Power considerations

Now we will discuss about these limitations one by one in the following pages.

(*i*) *Technology and architecture*

There are different technologies available for the fabrication of ICs. These are based on the different types of transistors that can be formed in the monolithic process. The transistors may be either bipolar junction transistors (BJT) or the metal-oxide-semiconductor field effect transistors (MOSFET). There are different types of processes of each transistor based on power requirements, size and so on. So for a possible digital system design, a suitable technology and architecture has to be selected.

(*ii*) *Integration of components*

After deciding the technology and architecture for the particular system design, there will be a method to integrate the suitable components for our design. There are two methods for the integration purposes, namely, top-down and bottom-up methods. In the top-down approach, the main system is divided into small components for separate design. But in bottom-up approach, it starts from the lowest level of the circuit design like transistor. From that the basic building blocks are made like gates in the digital circuits, amplifiers and comparators and so on. Here each approach has its own advantages and disadvantages too. The top-down method gives a better control of the design. However it is difficult to define the lower level of abstractions. But in bottom-up approach, each level provides a satisfactory solution to that stage's requirement, but as it goes up in the level, there is no control at the end point. So the final design may not be optimized in terms of size or cost. Nowadays most of the systems are made up of the mixture of top-down and bottom-up methods.

(*iii*) *Packaging and fabrication*

Using the automated layout tools, the structure of the ICs is defined. Basically the actual semiconductor area consists of two main parts. Firstly, the central part, referred as the cone, contains the complete functional circuit of the designed one. The input and output signals, control and power supply lines must be taken from the central area to the outside world for connection to the other circuits and instruments. This is done through the bond pads which are placed around the periphery of the die. Fig. 14-2 shows the details of the digital IC and its package details.

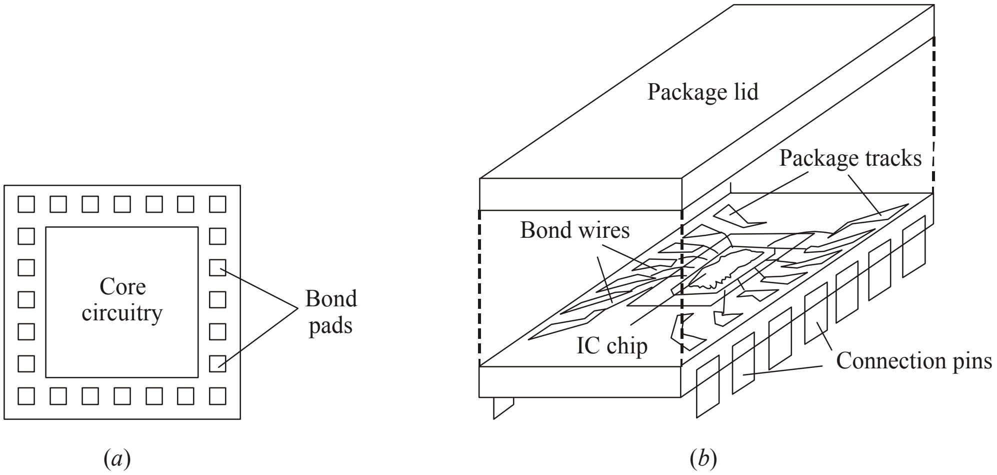


Fig. 14-2. (*a*) Plan of bare IC; (*b*) Assembly and packaging of IC.

(*iv*) *Power considerations*

All the circuit building blocks require a power supply. The number of inputs to the circuit and the voltage values will vary from technology to technology and for different design specifications. The minimum and the most usual supply requirements are two lines, a positive voltage and a ground connection. But some analog circuits may require negative supply also. These power supply related lines should be carefully considered while designing the IC.

14-4. Digital Design Aspects

Nowadays software tools have become the essential part of the digital design. The availability, the use of hardware description languages (HDLs) and the relative circuit simulation and synthesis tools have changed the entire digital system design. Using computer-aided design, various software tools are used to improve the designer's productivity and helped to increase the correctness of the quality designs. Some useful software tools for the software designs are listed below.

(*i*) *Schematic Designs*

It is equivalent to our normal word processor kind of tool for the digital system design. Here we can draw the schematics of the circuit diagrams in the software and this will be useful for easy reference and for any revision. All the libraries related to the hardware designs are available and we can just drag and paste into the software based on our hardware design. It is very easy to find any error while designing the system. But using this software, we can only draw the Schematics and we cannot simulate

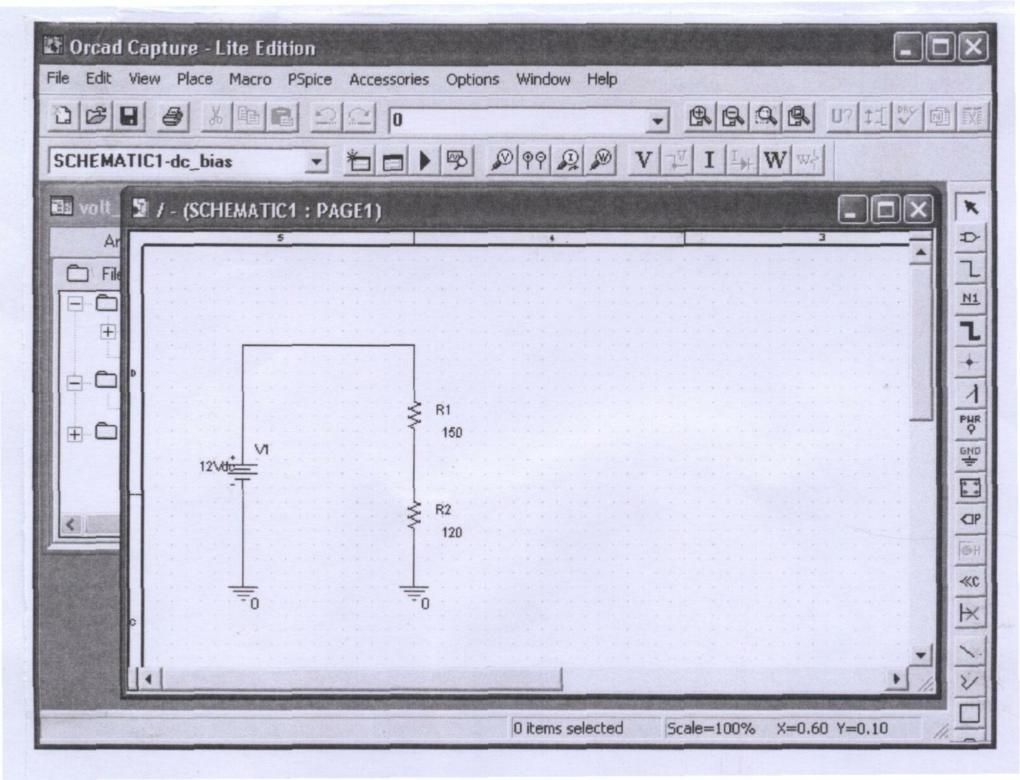
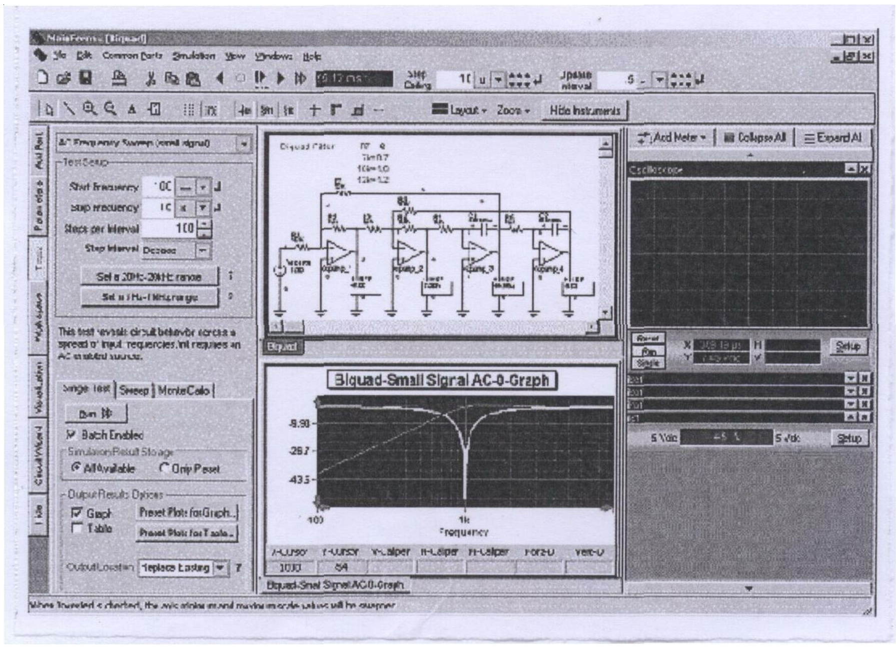


Fig. 14-3. Snapshot of orcad tool.



AC Frequency Sweep (small signal)

Test Setup

Start Frequency 100

Stop frequency 10k

Steps per interval 100

Step interval Decade

This test reveals circuit behavior across a spread of input frequencies and requires an AC enabled source.

☑ Batch Enabled

Simulation Result Storage

● All Available ○ Only Preset

Output Results Options

☑ Graph

□ Table

Output Location Replace Existing

When Threshold is checked, the axis minimum and maximum scale values will be sharpened.

Fig. 14-4. Snapshot of pspice tool.

any actual conditions. Some of the examples of this kind of softwares are pspice and orcad. They are shown in Figs. 14-3 and 14-4.

(*ii*) *Hardware Description Languages (HDLs)*

Hardware description languages are originally developed for circuit modelling. But nowadays it is mostly used for hardware design. They can be used to design anything from individual function modules to large, multichip digital systems. These languages will be similar to C and C++, and easily programmable. VHDL and Verilog are some examples of this software.

(*iii*) *HDL Compilers and Simulators*

A HDL software package consists of several components. In this environment, the designer writes a text-based program. Then the HDL compiler analyses the program for syntax errors. If there is no error during compilation, then the designer may send the program to a synthesis tool that creates a corresponding circuit design using some particular hardware technology. In all these software packages, a simulator will be included for circuit simulation. Normally, before the synthesis, the designer may use the compiler's result into the simulator to verify the behaviour of the design. If the designer wants to change any details, then that statement can be changed using the software before it goes for synthesis.

(*iv*) *Simulators*

The design cycle for a customized single-chip digital integrated circuit is expensive and takes longer time. Once the first chip is made, it is very difficult and impossible to debug it by probing the internal connections to change the gates and interconnections. Normally, the changes must be made in the original design database and a new chip should be manufactured again to induct the new required changes. Since the chip design and development takes many months time to complete the process, the designer wants to test all the behaviours in the first time itself before it goes for mass production. Thus the money and time can be saved. So these simulators will be used to simulate all the possible combinations of the behaviours before making any IC.

(*v*) *Test Benches*

The test benches are used to formalize circuit simulation and testing using the software environments. Here a set of programs for a digital system design are used to automatically exercise its functions to check the functional and timing characteristics of the design. These are used to find any faults with the making of ICs and behaviours. These test programs may be written in high-level languages. Some of popular test benches are made by Agilent, Terradyne and many other companies. Fig. 14-5 shows a test setup of IC testing facilities.

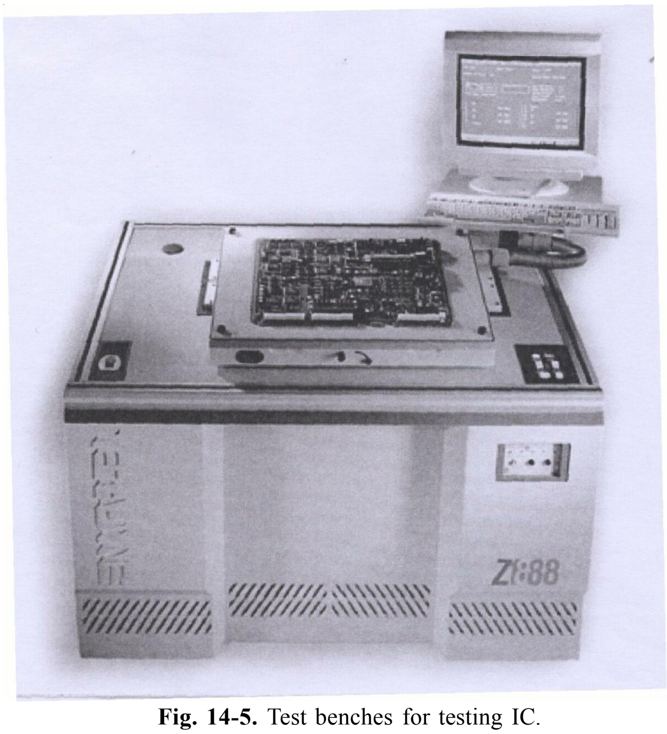


Fig. 14-5. Test benches for testing IC.
(Courtesy: Terradyne Corporation)

(*vi*) *Timing Analyzers and Verifiers*

The timing details are required for any digital design. All the digital circuits take time to produce a new output value in response to an input change. Most of the designer's effort will be spent ensuring that such output changes occurred correctly. So some specialized programs will automate the tedious task of drawing timing diagrams and specifying and verifying the timing relationships between different signals in a complex system. These programs are given by timing analyzers and verifiers.

14-5. VHDL

VHDL (Very high speed integrated circuit hardware description language) has become an important tool in designing and testing the complicated digital systems. Because of its simple and powerful modelling capacity VHDL has got immense popularity from the designers' community. It was developed in the early 1980s for managing design problems that involved in the large digital circuits and multiple team of engineers. VHDL is a software programming language like C and C++. It can be used to develop hardware using abstract high level design methods. VHDL compilers are available as stand-alone programs, and they can be available in so many CAD tools. In this chapter, we will discuss about how to write a simple VHDL program to design simple elements.

14-6. VHDL Design Flow

VHDL design flow has a number of steps to be followed during the design process. So in each step, we have to carry out some tasks. Normally the design process will look like Fig. 14-6.

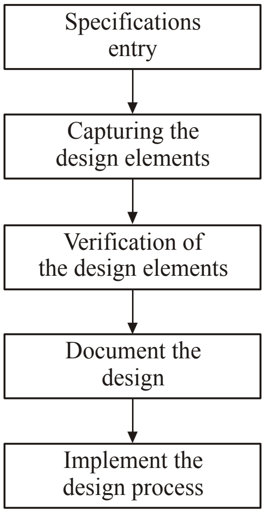

Specifications entry
Capturing the design elements
Verification of the design elements
Document the design
Implement the design process

Fig. 14-6. VHDL design flow.

During the first step, all the information related to the design will be ready and they will be entered into as design statements. Based on any models like top-down-approach design, bottom-up-design, the specifications of designs will be made and goes for revision with the design team. During the capturing mode, all the design elements are entered into the VHDL compiler system for design. In this phase, the designs will be captured either as schematics or as hardware description languages. In the verification mode, all the designs we have made during the design phase will be simulated for any problems. So this simulation will help us to make the design perfect and some basic timing characteristics of the design. In the documentation phase, all the design specifications, programs, simulation details and test data are documented for future uses and for further revisions. In the implementation phase, the design will be made into production of the ICs and other units.

Now we will see how the design is taking place in VHDL software environment.

For any design of digital systems using the VHDL, there are five types of individual design units available. They are entities, architectures, packages, package bodies and configurations. So basically all these five units are clubbed together to form a design library. This is an integral part of any design system. The library is shown in Fig. 14-7.

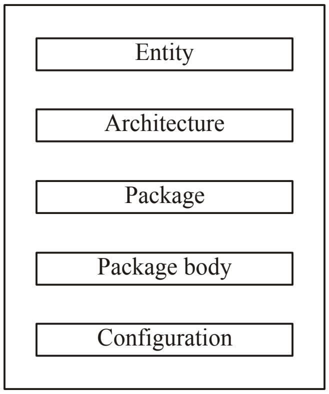


Fig. 14-7. VHDL library system.

Now we will see about each sub-unit in detail.

Entity :

An entity declaration gives us a complete interface of the design circuit. By using the specifications given in this declaration like name, data types and direction of ports, we can connect the circuits into other circuits. Consider a full adder circuit shown in Fig. 14-8.

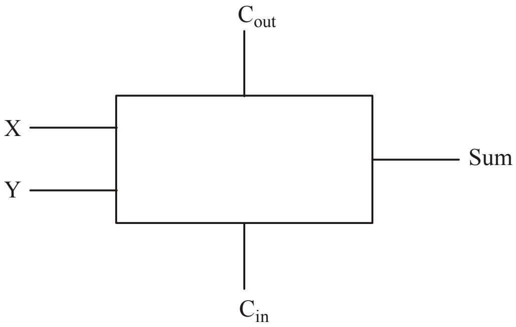


Fig. 14-8. Entity unit example.

For the full adder shown in Fig. 14-8, we can write entity declaration as like,

entity FULL _ ADDER is

port (X, Y, C_{in} : in BIT; C_{out}, Sum : Out BIT); end FULL_ADDER;

So when we write an entity, we have to give a name for the entity and the corresponding port list (input and output ports of the circuit). Normally the entity is denoted by,

entity entity - name is
port (interface — signal — declaration),
end entity-name;

Architecture :

The architecture declaration describes the function of the circuit. Normally this architecture gives the structure of the circuit and its basic functionality. We can create more than one architecture assignments for the given function. Thus, this system will be useful for a project team with many engineers and for simulation of the design unit. An architecture consists of intermediate signals, components used in the circuit, functions and procedures for a circuit. The declaration starts with an ‘architecture’ keyword. The functions and procedures are placed between the ‘begin’ and ‘end’ statements. The statements will be written as,

architecture BEHAVE of FULL_ADDER is begin
Sum < = X xor Y xor C_{in};
C_{out} < = (X and Y) or (X and C_{in}) or (Y and C_{in})
end BEHAVE ;

Normally the architecture is declared by,

```
architecture architecture-name of entity.
        (declarations)
begin
        (architecture body)
end architecture-name;
```

Package :

A package declaration is used to collect commonly used declarations for sharing between different design units. Normally a package will store common storage area to store type declarations, constants and global programs. A package consists of two parts, namely, package declaration and a package body. Package declarations will be having the following statements as its contents. The are as follows,

(*i*) Type and subtype declaration
(*ii*) Global signal declarations
(*iii*) Constant declarations
(*iv*) Function and procedure declarations
(*v*) Attribute specifications
(*vi*) File declarations
(*vii*) Component declarations
(*viii*) Alias declarations
(*ix*) Disconnect specifications
(*x*) Use clauses.

These are all some assignment statements which are used in the declaration of the package body.

Package body :

If a package declaration contains functions or procedures and one or more constants, then a package body is required in addition to the package declaration. A package body must have the same name as package declaration and can be located anywhere in the program. There may be only one package body for one package declaration.

An example of package and package body is shown here

```
begin
                a : = num;
                for i in size down to 1 loop
                if ((a mod 2) = 1) then
                ret (i) : = ‘1’;
                else.
package     package-name is
                package declarations
end package_name,
end
package body package-name is
            package body declarations
end package body package_name;
```

Configurations :

Configuration declaration gives the circuit parts listed in the digital system design. It contains the details like, which architectures bound from which entities and which components to be changed from a circuit during a simulation. Configuration starts with a keyword 'configuration'. The following statements will show how to describe a configuration in the programming environment.

```
Configuration this-build of rcomp is
            for structure
            for COMP 1 :     Compare use entity
                             work. Compare (version 1);
            for ROT 1 :      rotate use entity
                             work. rotate (version 2);
end for;
end this_build;
```

Here COMP1 and ROT1 are two instances and they will use two functions using the entities compare and rotate.

14-7. Compilation and Simulation of VHDL Code

Until now we have seen the codes for designing a system using program codes. After carefully designing the system, now we compile the code for any errors. This compiler will check for program syntax and semantic rules of VHDL. It also checks for the reference libraries mentioned in the code. If everything is correct in the code, then it will generate an intermediate code. The intermediate code is required for the simulator. But before going to the simulator, this intermediate code is sent to the elaborator. In the elaborator, ports are generated for each component, memory storage is allocated for required signals and all the interconnection details between the ports and the components are mentioned. After these operations, the code will be entered to the simulator. There we can see all the timing details of each operation and their functionalities. Basically there will be a output screen

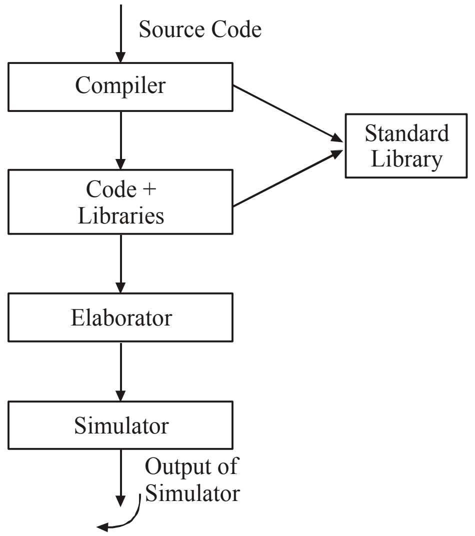


Fig. 14-9. Compilation and Simulation of VHDL code.

that shows all the operations performed by execution of the program. Fig. 14-9 shows the flow of program codes from development to simulation.

14-8. Data Types and Operators of VHDL

Data type & operators	Example
Bit	Value < = '1'
Bit-Vector	Value < = '00 11 0001'
Boolean	Value ≤ True
Integer	Value < = Value + 2
Real	Value = Value/2.3;
Time	Value = 'I' after 3ns;
Character	Char < = 'X'
String	Str < = "STR" & Addr
not	not value
and	Value 1 and Value 2
or	Value 1 or Value 2
Xor	Value 1 xor Value 2
nand	Value 1 nand Value 2
nor	Value 1 nor Value 2
Xnor	Value 1 xnor Value 2

14-9. Some Defined Symbols in VHDL

Symbol	Meaning
+	Addition
–	Subtraction
/	Division
=	Equal
<	Less than
>	Greater than
&	Concantenation
|	Vertical bar
;	Terminator
#	Enclosing based literal
(	Left parenthesis
)	Right parenthesis
.	Dot notation
:	Separates data object-from type
"	Double quote
'	Single quote
**	Exponentiation
=>	Arrow meaning "then"
=>	Arrow meaning "gets"
:=	Variable assignment
/=	In equality
>=	Greater than or equal to
<=	Less than or equal to
<=	Signal assignment
<>	Box
--	Comment

Example 14-1. *Design an entity and architecture statements for a comparator which is shown in Fig. 14-10.*

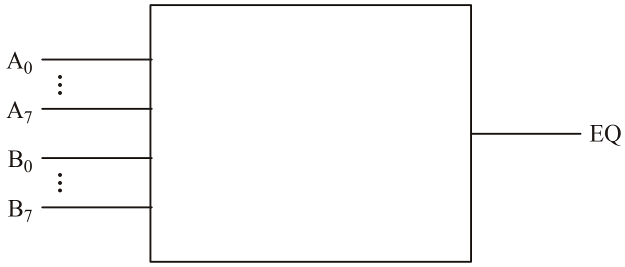


Fig. 14-10.

Solution : The comparator shown in Fig. 14-10 has two sets of inputs and one output (EQ). The operation of this comparator can be written in high-level languages as like,

```
entity COMPARATOR is
        port (A, B : in bit-vector (0 to 7);
            EQ : Out bit);
end COMPARATOR;
architecture COMPARATOR 1 of COMPARATOR is begin
        EQ < = '1' when (A = B) else '0';
end COMPARATOR 1;
```

Example 14-2. *Design a program for a multiplexer which is shown in Fig. 14-11 and having time to input as 1ns.*

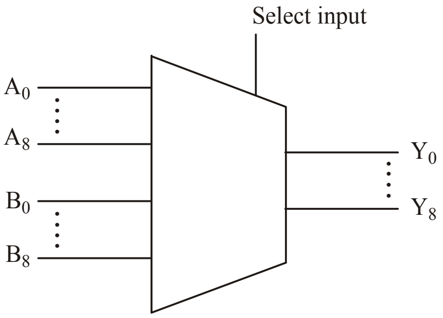


Fig. 14-11.

Solution : This is a 2:1 multiplexer. It has two 8 input registers of A and B. The output comes as Y_0 .... Y_8. It has a single select input as a selector and it has timing as 1ns.

We can write a program for this multiplexer as,

```
entity Mux 8 is
        port (A, B : in BIT_VECTOR (7 down to 0);
        Sel : in BIT : = '0'; Y : Out BIT-VECTOR (7 down to 0);
end
architecture Behave of Mux 8 is
begin
        Y < = A after TPD when Sel = '1' else B after TPD;
end;
```

Example 14-3. *Develop a program code for a zero detector which is shown in Fig. 14-12 and* *has the timing as 1ns for input to output.*

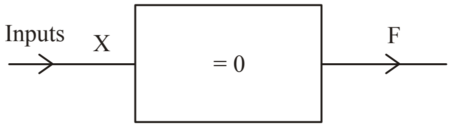


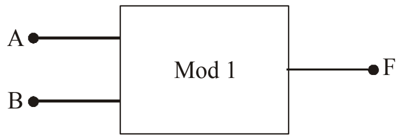
Fig. 14-12.

Solution : This is a variable width zero detector that accepts a bus of any width and will produce a single-bit output of '1' if all input bits are zero.

The program may be written as,

```
entity All zero is
        port (X : BIT-VECTOR; F : out BIT);
end;
architecture Behave of All zero is
begin process (X) begin F < = '1' after TPD;
        for j in X'RANGE loop
              if × (j) = '1' then F < = '0' after TPD;
              end if;
        end loop;
end process;
end;
```

Example 14-4 : *Write the VHDL entity and declaration statements for a module which is shown in Fig. 14-13.*



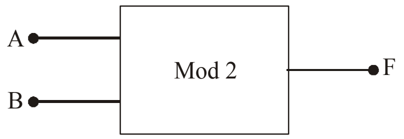
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 14-13.

Solution : Fig. 14-13 has two inputs and one output. The truth table shown above is for a OR gate. So we can write the entity and declaration statements as such,

```
entity Mod 1 is
        port (A, B : in bit; F : out bit) :
end Mod 1;
architecture Logic of Mod 1 is
begin
        F < = A or B;
end Logic;
```

Example 14-5 : *Write a VHDL entity and architecture statements for a module which is shown in Fig. 14-14.*



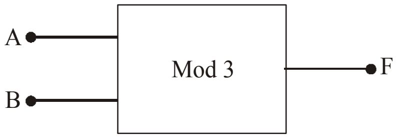
A	B	F
0	0	1
0	1	1
1	0	1
1	1	1

Fig. 14-14.

Solution : The Module 2 which is shown in Fig. 14-14 has two inputs and one output. The truth table denotes that as a nand gate. So we can write the statements as,

```
entity Mod 2 is
        port (A, B : in bit; F : out bit);
end Mod 2;
architecture Logic of Mod 2 is
begin
        F ≤ A nand B;
end Logic;
```

Example 14-6 : *Write a VHDL entity and architecture statements for a module shown in Fig. 14-15.*



A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 14-15.

Solution : The module which is shown in Fig. 14-15 has two inputs and one output. The truth table denotes that module as xnor gate. So we can write the program as,

```
entity Mod 3 is
        port (A, B : in bit; F : out bit);
end Mod 3;
architecture Logic of Mod 3 is
begin
        F < = A xnor B;
end Logic;
```

SUMMARY

1. CAD tools are used very much to design the digital hardware system.

2. VHDL is one of the main important CAD designing tools.

3. Digital design process has many steps to be followed like specifications, subsystem design, simulation, layout checking, wafer fabrication and so on.

4. During every process of subunit system, it has to be tested for its specifications.

5. Digital design using CAD tools minimizes time spent for the design of the system.

6. There are some limitations also during the development of design system.

7. Schematic design software tools are like word processor like tools, used to draw the schematic diagram of the circuit or a digital system.

8. Hardware description languages are used to write a sophisticated program to write any software for a digital system.

9. Simulators are used to simulate the digital system for its accuracy and timing specifications.

10. Test benches are specialized tools to test the digital designs.

11. VHDL is a software programming language like C and C++ and used to develop the hardware using abstract high-level languages.

12. VHDL design phase has five stages, namely, specifications, capturing, verification, documentation and implementation.

GLOSSARY

CAD : Computer Aided Design tools used to design any digital system design.

VHDL : Very high speed integrated circuit hardware description language used to design any digital system and ICs.

Schematic diagram : Representation of digital circuits in more readable form in the computer.

Simulation : Checking the correctness of the design using the compilers available in the computer itself.

Hardware Description Languages (HDLs) : These are developed for circuit modelling. It is like a high-level language used to design any digital system.

HDL Compilers and Simulators : These are used to key in the software programs for programming and used to analyse the results using the computer itself.

Test benches : These are specifically used to test the digital system using programming languages. So test programs will be used to test the functionality of the circuit.

Entity : It is used to denote the complete interface of the design circuit. All the VHDL design units will be having one or more entities.

Architecture : It is one type of design unit used to describe the function of the design. This will be associated with the library.

Package : It is used to collect commonly used declarations for sharing between the design units.

Package body : It will be associated with the package declaration and will be available, if the package has functions and procedures.

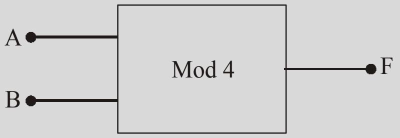
Configuration : It is used to assemble all the design units before the simulation. It gives the details like binding of entities to the architecture, mapping of the components and so on.

DESCRIPTIVE QUESTIONS

1. Explain the basic digital system design process.
2. What are the design considerations of a digital system?
3. What are the tools for basic digital system design?
4. Explain about VHDL design flow.

TUTORIAL PROBLEMS

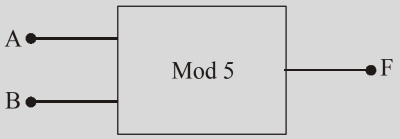
1. Write a VHDL entity and architecture declaration for the module and truth table shown in Fig. 14-16.



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 14-16.

2. Write a VHDL entity and architecture declaration for the module and truth table shown in Fig. 14-17.



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 14-17.

3. Write a VHDL entity and architecture declaration statements for an 8-bit ripple carry adder which is shown in Fig. 14-18.

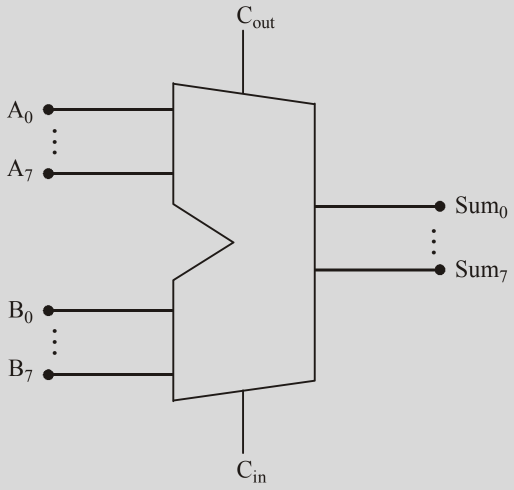


Fig. 14-18.

MULTIPLE CHOICE QUESTIONS

1. Programming of CAD tools are based on
   (*a*) Machine language (*b*) Assembly language
   (*c*) High-level languages (*d*) All of the above
2. Hardware description languages (HDLs) are used
   (*a*) to design a digital system (*b*) to write a program
   (*c*) for modelling of the system (*d*) for simulating the program output
   (*e*) All of the above
3. The simulators are used to
   (*a*) find the problems in the IC manufacturing
   (*b*) simulate the actual working scenarios of IC before making it.
   (*c*) test the written programs.
   (*d*) an IDE for program development.
4. Number of design units in VHDL is
   (*a*) 4 (*b*) 3 (*c*) 2 (*d*) 5
5. VHDL is used to develop the
   (*a*) analog circuit systems (*b*) digital systems
   (*c*) Both of the above (*d*) None of the above

ANSWERS

1. (*c*) **2.** (*e*) **3.** (*c*) **4.** (*d*) **5.** (*b*)

Appendix A

DATA SHEETS OF SELECTED DIGITAL INTEGRATED CIRCUITS

SN5400, SN54LS00, SN54S00, SN7400, SN74LS00, SN74S00 QUADRUPLE 2-INPUT POSITIVE-NAND GATES

DECEMBER 1983-REVISED MARCH 1986

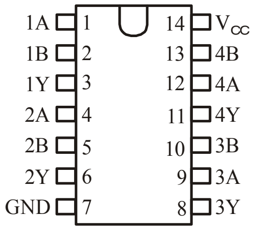
- **Package Options Include Plastic “Small Outline” Package, Ceramic Chip Carriers and Flat Packages. and Plastic and Ceramic DIPs**
- **Dependable Texas Instruments Quality and Reliability**

Description

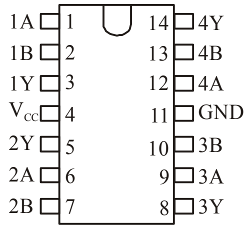
These devices contain four independent 2-input-NAND gates.

The SN5400, SN54LS00, and SN54S00 are characterized for operation over the full military temperature range of – 55º C to 125º C. The SN700, SN74LS00 and SN74S00 are characterized for operation from 0 ºC to 70 ºC.

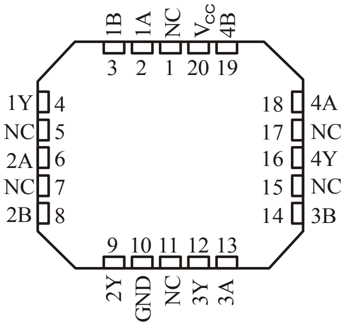
SN5400. . . J PACKAGE
SN54LS00, SN54S00. . . J OR W PACKAGE
SN74S00, . N PACKAGE
SN74LS00, SN74S00. . . D OR N PACKAGE
(TOP VIEW)



SN5400. . . W PACKAGE
(TOP VIEW)



SN54LS00, SN54S00 . . . FK PACKAGE
(TOP VIEW)

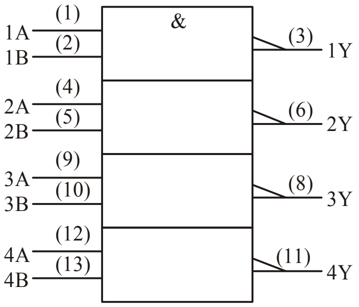


NC No internal connection

FUNCTION TABLE (each gate)

INPUTS		*OUTPUT*
A	*B*	*Y*
H	H	L
L	X	H
X	L	H

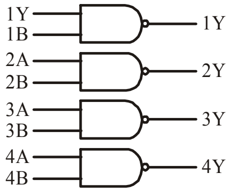
logic symbol †



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12

Pin numbers shown are for D, J, and N packages.

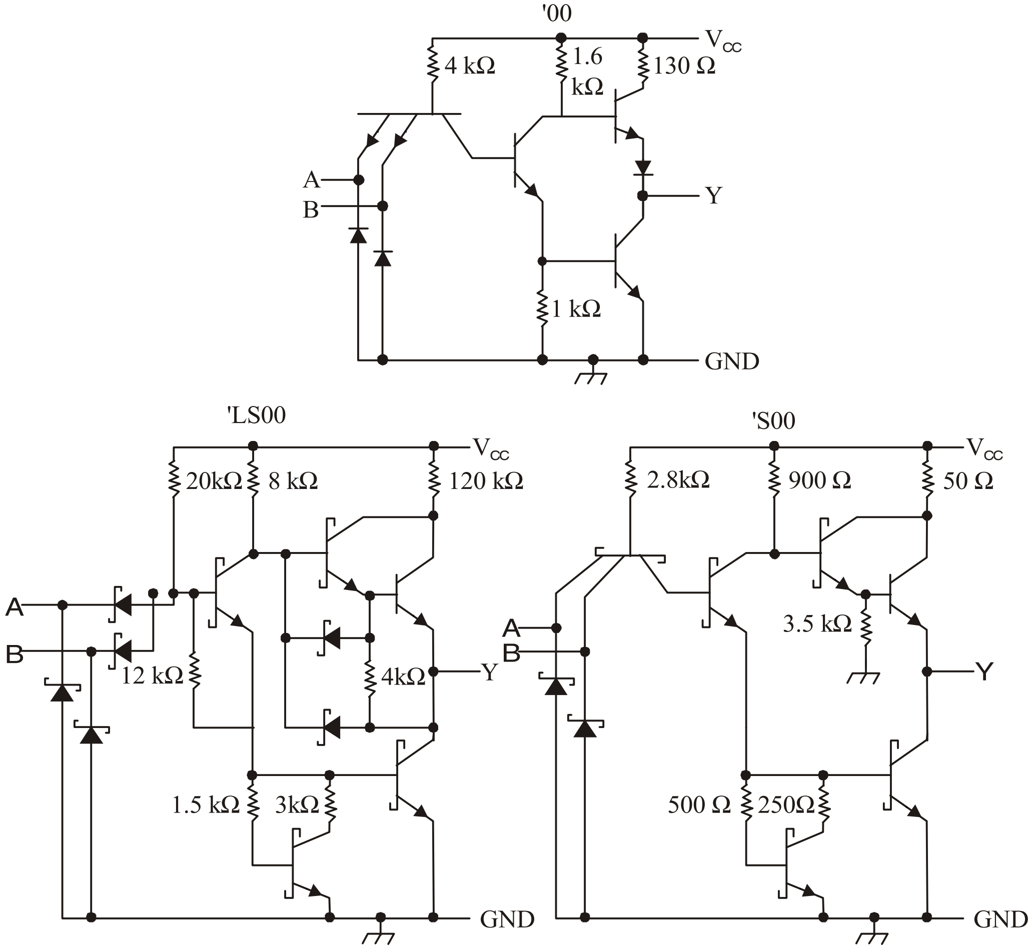
logic diagram (positive logic)



$Y \quad \overline{A \quad B}$ or $Y \quad \overline{A} + \overline{B}$

SN5400, SN54LS00, SN54S00,
SN7400, SN74LS00, SN74S00
QUADRUPLE 2-INPUT POSITIVE-NAND GATES

schematics (each gate)



Resistor values shown are nominal.

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1)	7 V
Input voltage: '00, 'S00	5.5 V
'LS00	7 V
Operating free-air temperature range: SN54'	– 55º C to 125º C
SN74'	0º C to 70º C
Storage temperature range	– 65º C to 150º C

NOTE 1: Voltage values are with respect to network ground terminal.

SN5400, SN7400
QUADRUPLE 2-INPUT POSITIVE-NAND GATES

recommended operating conditions

	SN5400			SN7400			UNIT
	MIN	*NOM*	*MAX*	*MIN*	*NOM*	*MAX*	
V_{CC} supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH} High-level input voltage	2			2			V
V_{IL} Low-level input voltage			0.8			0.8	V
I_{OH} High-level output current			– 0.4			– 0.4	mA
I_{OL} Low-level output current			16			16	mA
T_A Operating free-air temperature	– 55		125	0		70	ºC

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARA-METER	*TEST CONDITIONS* †	SN5400			SN700			*UNIT*
		MIN	*TYP* ‡	*MAX*	*MIN*	*TYP* ‡	*MAX*	
V_{IK}	V_{CC} = MIN. I_I = – 12 mA			– 1.5			– 1.5	V
V_{OH}	V_{CC} = MIN. V_{IL} = 0.8 V, I_{OH} = – 0.4 mA	2.4	3.4		2.4	3.4		V
V_{OL}	V_{CC} = MIN, V_{IH} = 2 V, I_{OL} = 16 mA		0.2	0.4		0.2	0.4	V
I_I	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
I_{IH}	V_{CC} = MAX, V_I = 2.4 V			40			40	µA
I_{IL}	V_{CC} = MAX, V_I = 0.4 V			– 1.6			– 1.6	mA
I_{OS} §	V_{CC} = MAX	– 20		– 55	– 18		– 55	mA
I_{CCH}	V_{CC} = MAX, V_I = 0 V	4	8			4	8	mA
I_{CCL}	V_{CC} = MAX, V_I = 4.5 V		12	22		12	22	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡ All typical values are at VCC = 5 V, T_A = 25º C.

§ Not more than one output should be shorted at a time.

switching characteristics, V_{CC} = 5 V, T_A = 25 ºC

PARAMETER	*FROM (INPUT)*	*TO (OUTPUT)*	*TEST CONDITIONS*	*MIN*	*TYP*	*MAX*	*UNIT*
t_{PLH}	A or B	Y	R_L = 400 Ω, C_L = 15 pF		11	22	ns
t_{PHL}					7	15	ns

SN54LS00, SN74LS00
QUADRUPLE 2-INPUT POSITIVE-NAND GATES

recommended operating conditions

	SN54LS00			SN74LS00			UNIT
	MIN	*NOM*	*MAX*	*MIN*	*NOM*	*MAX*	
V_{CC} supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH} High-level input voltage	2			2			V
V_{IL} Low-level input voltage			0.7			0.8	V
I_{OH} High-level output current			– 0.4			– 0.4	mA
I_{OL} Low-level output current			4			8	mA
T_A Operating free-air temperature	– 55		125	0		70	ºC

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARA-METER	*TEST CONDITIONS* †	SN54LS00			SN74LS00			*UNIT*
		MIN	*TYP* ‡	*MAX*	*MIN*	*TYP* ‡	*MAX*	
V_{IK}	V_{CC} = MIN. I_I = – 18 mA			– 1.5			– 1.5	V
V_{OH}	V_{CC} = MIN. V_{IL} = MAX, I_{OH} = – 0.4 mA	2.5	3.4		2.7	3.4		V
V_{OL}	V_{CC} = MIN, V_{IH} = 2 V, I_{OL} = 4 mA		0.25	0.4		0.25	0.4	V
	V_{CC} = MIN, V_{IH} = 2 V, I_{OL} = 8 mA					0.35	0.5	
I_I	V_{CC} = MAX, V_I = 7 V			0.1			0.1	mA
I_{IH}	V_{CC} = MAX, V_I = 2.7 V			20			20	µA
I_{IL}	V_{CC} = MAX, V_I = 0.4 V			– 0.4			– 0.4	mA
I_{OS} §	V_{CC} = MAX	– 20		– 100	– 20		– 100	mA
I_{CCH}	V_{CC} = MAX, V_I = 0 V		0.8	1.6		0.8	1.6	mA
I_{CCL}	V_{CC} = MAX, V_I = 4.5 V		2.4	4.4		2.4	4.4	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡ All typical values are at V_{CC} = 5 V, T_A = 25º C.

§ Not more than one output should be shorted at a time, and the duration of the short-circuit should not exceed one second.

switching characteristics, V_{CC} = 5 V, T_A = 25º C

PARAMETER	*FROM* (*INPUT*)	*TO* (*OUTPUT*)	*TEST CONDITIONS*	*MIN*	*TYP*	*MAX*	*UNIT*
t_{PLH}	A or B	Y	R_L = 2 kΩ, C_L = 15 pF		9	15	ns
t_{PHL}					10	15	ns

SN54F00, SN74F00
QUADRUPLE 2-INPUT POSITIVE-NAND GATES

SDFS035A – MARCH 1987 – REVISED OCTOBER 1993

- **Package Options Include Plastic Small-Outline Packages, Ceramic chip Carriers and Standard Plastic and Ceramic 300-mll DIPs**

Description

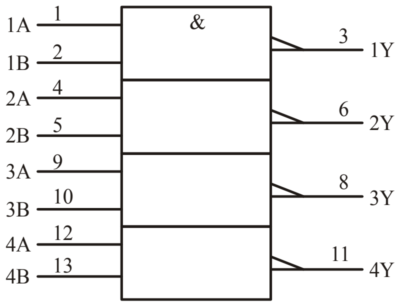
These devices contain four independent 2-input NAND gates. They perform the Boolean function $Y = \overline{A \cdot B}$ or $Y = \overline{A}+\overline{B}$ in posive logic.

The SN54F00 is characterized for operation over the full military temperature range of – 55º C to 125º C. The SN74F00 is characterized for operation from 0º C to 70º C.

FUNCTION TABLE (each gate)

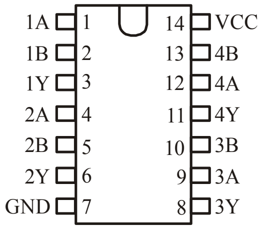
INPUTS		*OUTPUT*
A	***B***	***Y***
H	H	L
L	X	H
X	L	H

logic symbol†

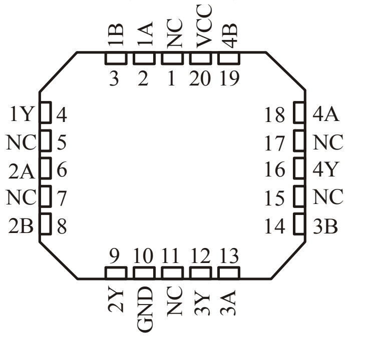


†This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

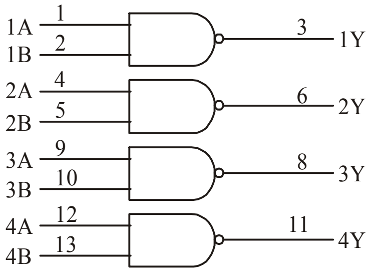
SN5400. . . J PACKAGE
SN54LS00, SN54S00. . . J OR W PACKAGE
SN74S00, . N PACKAGE
SN74LS00, SN74S00. . . D OR N PACKAGE
(TOP VIEW)



SN54LS00, SN54S00 . . . FK PACKAGE
(TOP VIEW)



logic diagram (positive logic)



Pin numbers shown are for D, J, and N packages.

SN54F00, SN74F00
QUADRUPLE 2-INPUT POSITIVE-NAND GATES

SDFS035A – MARCH 1987 – REVISED OCTOBER 1993

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage range, V_{CC}	– 0.5 V to 7 V
Input voltage range, V_I (see Note 1)	– 1.2 V to 7 V
Input current range	– 30 mA to 5 mA
Voltage range applied to any output in the high state	– 0.5° C to 125° C
Current into any output in the low state	40 mA
Operating free-air temperature range: SN54F00	– 55° C to 125° C
SN74F00	0° C to 70° C
Storage temperature range	– 65° C to 150° C

† Stresses beyond those listed under “absolute maximum ratings” may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under “recommended operating conditions” is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1. The input voltage ratings may be exceeded provided the input current ratings are observed.

recommended operating conditions

	SN54F00			SN74F00			UNIT
	MIN	*NOM*	*MAX*	*MIN*	*NOM*	*MAX*	
V_{CC} supply voltage	4.5	5	5.5	4.5	5	5.5	V
V_{IH} High-level input voltage	2			2			V
V_{IL} Low-level input voltage			0.8			0.8	V
I_{IK} Input clamp current			– 18			– 18	mA
I_{OH} High-level output current			– 1			– 1	mA
I_{OL} Low-level output current			20			20	mA
T_A Operating free-air temperature	– 55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARA-METER	*TEST CONDITIONS*	SN54F00			SN74F00			*UNIT*
		MIN	*TYP* ‡	*MAX*	*MIN*	*TYP* ‡	*MAX*	
V_{IK}	V_{CC} = 4.5V, I_I = – 18 mA			– 1.2			– 1.2	V
V_{OH}	V_{CC} = 4.5 V, I_{OH} = – 1 mA	2.5	3.4		2.5	3.4		V
	V_{CC} = 4.75 V, I_{OH} = – 1 mA				2.7			
V_{OL}	V_{CC} = 4.5 V, I_{OL} = 20 mA		0.3	0.5		0.3	0.5	V
I_I	V_{CC} = 5.5 V, V_I = 7 V			0.1			0.1	mA
I_{IH}	V_{CC} = 5.5 V, V_I = 2.7 V			20			20	μA
I_{IL}	V_{CC} = 5.5 V, V_I = 0.5 V			– 0.6			– 0.6	mA

PARAMETER	*TEST CONDITIONS*	SN54F00			SN74F00			*UNIT*
		MIN	*TYP* ‡	*MAX*	*MIN*	*TYP* ‡	*MAX*	
I_{OS} §	V_{CC} = 5.5 V, V_O = 0	– 60		– 150	– 60		– 150	mA
I_{CCH}	V_{CC} = 5.5 V, V_I = 0		1.9	2.8		1.9	2.8	mA
I_{CCL}	V_{CC} = 5.5 V, V_I = 4.5 V		6.8	10.2		6.8	10.2	mA

‡ All typical values are at V_{CC} = 5 V, T_A = 25° C.

§ Not more than one output should be shorted at a time, and the duration of the short circuit should not exceed one second.

switching characteristics (see Note 2)

PARAMETER	*FROM* (*INPUT*)	*TO* (*OUTPUT*)	V_{CC} = 5 V, C_L = 50 pF, R_L = 500 Ω, T_A = 25° C			V_{CC} = 4.5 V to 5.5 V, C_L = 50 pF, R_L = 500 Ω , T_A = MIN to MAX†				*UNIT*
			'F00			SN54F00		SN74F00		
			MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
t_{PLH}	A or B	Y	1.6	3.3	5	2	7	1.6	6	ns
t_{PHL}			1	2.8	4.3	1.5	6.5	1	5.3	

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

Note 2: Load circuits and waveforms are shown in Section 1.

TEXAS INSTRUMENTS

CD54HC04, CD54HCT04
CD74HC04, CD74HCT04

Data shet acquired from Harris Semiconductor
SCHS117

August 1997

High Speed CMOS Logic Hex Inverter

Features

- **Buffered Inputs**
- **Typical Propagation Delay: 6ns at V_{CC} = 5 V, C_L = 15 pF, T_A = 25° C**
- **Fanout (Over Temperature Range)**
  - **Standard Outputs..................... 10 LSTTL Loads**
  - **Bus Driver Outputs .... 15 LSTTL Loads**
- **Wide Operating Temperature Range .... – 55° C to 125° C**
- **Balanced Propagation Delay and Transition Times**
- **Significant Power Reduction Compared to LSTTL Logic ICs**
- **HC Types**
  - **2 V to 6 V Operation**
  - **High Noise Immunity: N_{IL} = 30%, N_{IH} = 30% of V_{CC} at V_{CC} = 5 V**
- **HCT Types**
  - **4.5 V to 5.5 V Operation**
  - **Direct LSTTL Input Logic Compatibility, V_{IL} – 0.8 V (Max), V_{IH} = 2V (Min)**
  - **CMOS Input Compatibility, $I_I \leq 1\mu A$ at V_{OL}, V_{OH}**

Description

The Harris CD54HC04, CD54HCT04, CD74HC04 and CD74HCT04 logic gates utilize silicon gate CMOS technology to achieve operating speeds similar to LSTTL gates with the low power consumption of standard CMOS integrated circuits. All devices have the ability to drive 10 LSTTL loads. The 74HCT logic family is functionally pin compatible with the standard 74LS logic family.

Ordering Information

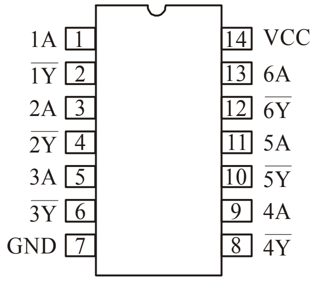
PART NUMBER	*TEMP. RANGE (°C)*	*PACKAGE*	*PKG. NO.*
CD74HC04E	– 55 to 125	14 Ld PDIP	E 14.3
CD74HCT04E	– 55 to 125	14 Ld PDIP	E 14.3
CD74HC04M	– 55 to 125	14 Ld SOIC	M 14.15
CD74HCT04M	– 55 to 125	14Ld SOIC	M 14.15
CD54HC04F	– 55 to 125	14 Ld CERDIP	F 14.3
CD54HCT04F	– 55 to 125	14 Ld CERDIP	F 14.3
CD54HC04W	– 55 to 125	Wafer	
CD54HCT04W	– 55 to 125	Wafer	
CD54HC04H	– 55 to 125	Die	
CD54HCT04H	– 55 to 125	Die	

NOTE:

[1] When ordering, use the entire part number. Add the suffix 96 to obtain the variant in the tape and reel.

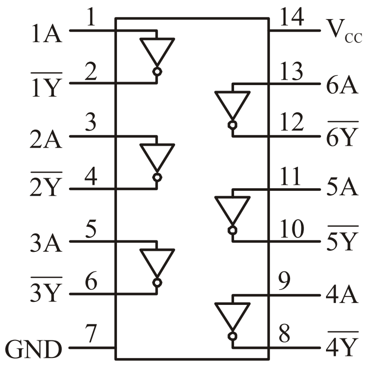
Pinout

CD54HC04, CD54HCT04, CD74HC04, CD74HCT04
(PDIP, CERDIP, SOIC)
(TOP VIEW)



CD54HC04, CD54HCT04, CD74HC04, CD74HCT04

Functional Diagram

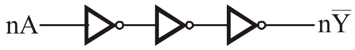


TRUTH TABLE

INPUTS	
nA	nY
L	H
H	L

NOTE: H = High Voltage Level, L = Low Voltage Level

Logic Symbol



CD54HC04, CD54HCT04, CD74HC04, CD74HCT04

Absolute Maximum Ratings

DC Supply Voltage, V_{CC}.................... – 0.5 to 7 V

DC Input Diode Current, I_{IK}

For V_I < – 0.5 V or V_I >V_{CC} + 0.5 V................ ± 20mA

DC Output Diode Current, I_{OK}

For V_O < – 0.5 V or V_O > V_{CC} + 0.5 V................ ± 20 mA

DC Output Source or Sink Current per Output Pin, I_O

For V_O > – 0.5 V or V_O < V_{CC} + 0.5 V................ ± 25 mA

DC V_{CC} or Ground Current, I_{CC} or I_{GND} .... ± 50 mA

Operating Conditions

Temperature Range (T_A)............... – 55° C to 125° C

Supply Voltage Range, V_{CC}

HC Types.. 2V to 6V

HCT Types................................4.5 V to 5.5 V

DC Input or Output Voltage, V_I, V_O......................... 0 V to V_{CC}

Input Rise and Fall time

2 V.. 1000ns (Max)

4.5 V.. 500ns (Max)

6 V.. 400ns (Max)

Thermal Information

Thermal Resistance (Typical, Note 2)	θ_{JA}	θ_{JC}
	(ºC/W)	(ºC/W)
PDIP Package.............................	100	N/A
CERDIP Package.......................	130	55
SOIC Package.............................	180	N/A

Maximum Junction Temperature (Hermetic Package or Die).................. 175° C

Maximum Junction Temperature (Plastic Package).................. 150° C

Maximum Storage Temperature Range..................... – 65º C to 150º C

Maximum Lead Temperature (Soldering 10s)................ 300º C

(SOIC - Lead Tips Only)

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

2. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

DC Electrical Specifications

		TEST CONDITIONS			*25º C*			*– 40º C TO + 85º C*		*– 55ºC TO 125ºC*		
PARAMETER	*SYMBOL*	*$V_I(V)$*	*I_o (mA)*	*V_{CC} (V)*	*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNITS*
HC TYPES												
High Level Input Voltage	V_{IH}	–	–	2	1.5	–	–	1.5	–	1.5	–	V
				4.5	3.15	–	–	3.15	–	3.15	–	V
				6	4.2	–	–	4.2	–	4.2	–	V
Low Level Input Voltage	V_{IL}	–	–	2	–	–	0.5	–	0.5	–	0.5	V
				4.5	–	–	1.35	–	1.35	–	1.35	V
				6	–	–	1.8	–	1.8	–	1.8	V

CD54HC04, CD54HCT04, CD74HC04, CD74HCT04

DC Electrical Specification (Continued)

		TEST CONDITIONS			*25° C*			*– 40° C TO + 85° C*		*– 55°C TO 125°C*		
PARAMETER	*SYMBOL*	*V_I(V)*	*I_o (mA)*	*V_{CC} (V)*	*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNITS*
High Level Output Voltage CMOS Loads	V_{OH}	V_{IH} or V_{IL}	– 0.02	2	1.9	–	–	1.9	–	1.9	–	V
			– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
			– 0.02	6	5.9	–	–	5.9	–	5.9	–	V
High Level Output Voltage TTL Loads			–	–	–	–	–	–	–	–	–	V
			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
			– 5.2	6	5.48	–	–	5.34	–	5.2	–	V
Low Level Output Voltage CMOS Loads	V_{OL}	V_{IH} or V_{IL}	0.02	2	–	–	0.1	–	0.1	–	0.1	V
			0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
			0.02	6	–	–	0.1	–	0.1	–	0.1	V
Low Level Output Voltage TTL Loads			–	–	–	–	–	–	–	–	–	V
			4	4.5	–	–	0.26	–	0.33	–	0.4	V
			5.2	6	–	–	0.26	–	0.33	–	0.4	V
Input Leakage Current	I_I	V_{CC} or GND	–	6	–	–	± 0.1	–	± 1	–	± 1	μA
Quiescent Device Current	I_{CC}	V_{CC} or GND	0	6	–	–	2	–	20	–	40	μA
HCT TYPES												
High Level Input Voltage	V_{IH}	–	–	4.5 to 5.5	2	–	–	2	–	2	–	V
Low Level Input Voltage	V_{IL}	–	–	4.5 to 5.5	–	–	0.8	–	0.8	–	0.8	V
High Level Output Voltage CMOS Loads	V_{OH}	V_{IH} or V_{IL}	– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
High Level Output Voltage TTL Loads			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
Low Level Output Voltage CMOS Loads	V_{OL}	V_{IH} or V_{IL}	0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
LOW Level Output Voltage TTL Loads			4	4.5	–	–	0.26	–	0.33	–	0.4	V
Input Leakage Current	I_I	V_{CC} and GND	0	5.5	–	–	± 0.1	–	± 1	–	± 1	μA
Quiescent Device Current	I_{CC}	V_{CC} or GND	0	5.5	–	–	2	–	20	–	40	μA
Additional Quiescent Device Current Per Input Pin: 1 Unit Load (Note)	ΔI_{CC}	V_{CC} – 2.1	–	4.5 to 5.5	–	100	360	–	450	–	490	μA

NOTE: For dual-supply systems theorectical worst case (V_I = 2.4 V, V_{CC} = 5.5 V) specification is 1.8 mA.

CD54HC04, CD54HCT04, CD74HC04, CD74HCT04

HCT Inputs Loading Table

INPUT	*UNIT LOADS*
nB	1.2

NOTE: Unit Load is ΔI_{CC} limit specified in DC Electrical.
Specifications table, e.g. 360µA max at 25° C.

Switching Specifications Input T_r, t_f = 6ns

PARAMETER	*SYMBOL*	*TEST CONDITIONS*	V_{CC} *(V)*	*25° C*			*– 40° C TO + 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
HC TYPES											
Propagation Delay, Input to Output (Fig. 1)	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	85	–	105	–	130	ns
			4.5	–	–	7	–	21	–	67	ns
			6	–	–	14	–	18	–	22	ns
Propagation Delay, Data Input to Output Y	t_{PLH}, t_{PHL}	C_L = 15pF	5	–	6	–	–	–	–	–	ns
Transition Times (Fig.1)	t_{TLH}, t_{THL}	C_L = 50pF	2	–	–	75	–	95	18	110	ns
			4.5	–	–	15	–	19	–	22	ns
			6	–	–	13	–	16	–	19	ns
Input Capacitance	C_I	–	–	–	–	10	–	10	–	10	pF
Power Dissipation Capacitance (Notes 3, 4)	C_{PD}	–	5	–	21	–	–	–	–	–	pF
HCT TYPES											
Propagation Delay, Input to Output (Figure 2)	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	–	19	–	24	–	29	ns
Propagation Delay, Data Input to Output Y	t_{PLH}, t_{PHL}	C_L = 50pF	5	–	7	–	–	–	–	–	ns
Transition Times (Fig. 2)	t_{TLH}, t_{THL}	C_L = 50pF	4.5	–	–	15	–	19	–	22	ns
Input Capacitance	C_I	–	–	–	–	10	–	10	–	10	pF
Power Dissipation Capacitance (Notes 3, 4)	C_{PD}	–	5	–	24	–	–	–	–	–	pF

NOTES:

3. C_{PD} is used to determine the dynamic power consumption, per gate.
4. $P_D = V_{CC}2\ f_i\ (C_{PD} + C_L)$ where f_i = input frequency, C_L = output load capacitance, V_{CC} = supply voltage.

CD54HC04, CD54HCT04, CD74HC04, CD74HCT04

Test Circuits and Waveforms

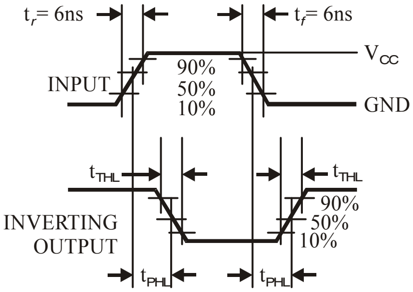


FIG. 1. HC TRANSITION TIMES AND PROPAGATION DELAY TIMES, COMBINATION LOGIC

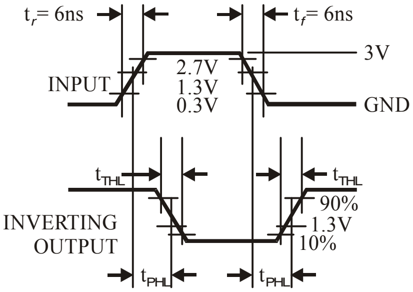


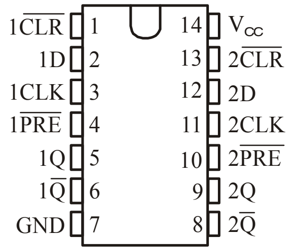
FIG. 2. HCT TRANSITION TIMES AND PROPAGATION DELAY TIMES, COMBINATION LOGIC

SN54AHC00, SN74AHC74 DUAL POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOPS WITH CLEAR AND PRESET

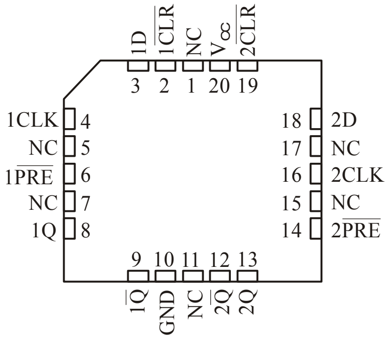
SCLS255E – DECEMBER 1995 – REVISED NOVEMBER 1998

- **Operating Range 2-V to 5.5-V V_{CC}**
- ***EPIC*™ (Enhanced-Performance Implanted CMOS) Process**
- **Latch-Up Performance Exceeds 250 mA Per JESD 17**
- **ESD Protection Exceeds 2000 V Per MIL-STD-883, Method 3015; Exceeds 200 V Using Machine Model (C = 200 pF, R = 0)**
- **Package Options Include Plastic Small-Outline (D), Shrink Small-Outline (DB), Thin Very Small-Outline (DGV), Thin Shrink Small-Outline (PW), and Ceramic Flat (W) Packages, Ceramic Chip Carriers (FK), and Standard Plastic (N) and Ceramic (J) DIPs**

SN54AHC74. . . J OR W PACKAGE
SN74AHC74. . .D, DB, DGV, N, OR PW PACKAGE
(TOP VIEW)



SN54AHC74 . . . FK PACKAGE
(TOP VIEW)



NC – No internal connection

Description

The ‘AHC74 devices are dual positive-edge-triggered D-type flip-flops.

A low level at the preset $\left(\overline{PRE}\right)$ or clear $\left(\overline{CLR}\right)$ inputs sets or resets the outputs, regardless of the levels of the other inputs. When $\overline{PRE}$ and $\overline{CLR}$ are inactive (high), data at the data (D) input meeting the setup time requirements is transferred to the outputs on the positive-going edge of the clock pulse. Clock triggering occurs at a voltage level and is not directly related to the rise time of the clock pulse. Following the hold-time interval, data at the D input can be changed without affecting the levels at the outputs.

The SN54AHC74 is characterized for operation over the full military temperature range of – 55º C to 125º C. The SN74AHC74 is characterized for operation from – 40º C to 85º C.

FUNCTION TABLE

INPUTS				*OUTPUTS*	
$\overline{PRE}$	$\overline{CLR}$	*CLK*	*D*	*Q*	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

† This configuration is nonstable; that is, it does not persist when $\overline{PRE}$ or $\overline{CLR}$ returns to its inactive (high) level.

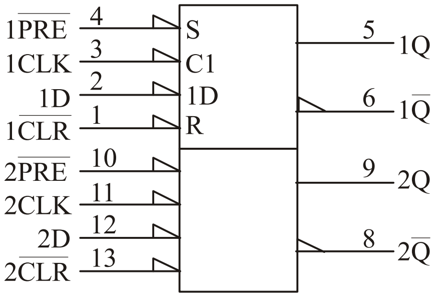
Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

EPIC is a trademark of Texas Instruments Incorporated.

SN54AHC74, SN74AHC741
DUAL POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOPS
WITH CLEAR AND PRESET

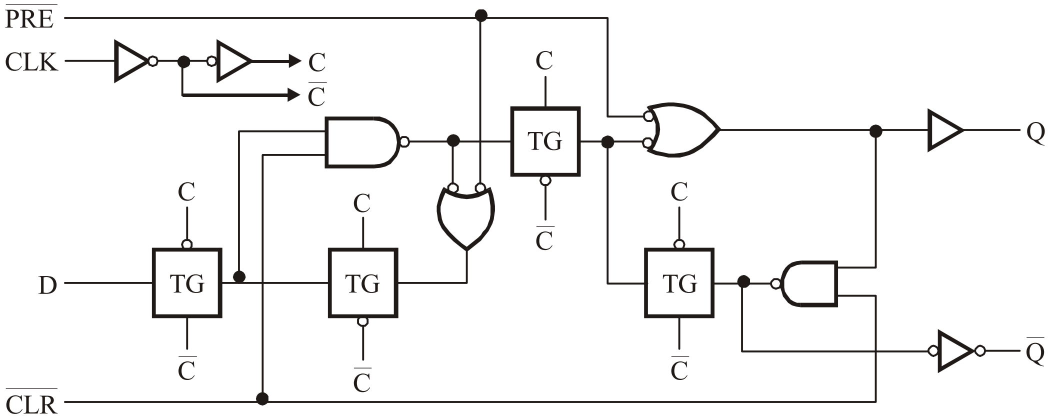
SCLS255E – DECEMBER 1995 – REVISED NOVEMBER 1998

logic symbol†



†This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.
Pin numbers shown are for the D, DB, DGV, J, N, PW, and W packages.

logic diagram, each flip-flop (positive logic)



SN54AHC74, SN74AHC74
DUAL POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOPS
WITH CLEAR AND PRESET
SDLS255E – DECEMBER 1995 – REVISED NOVEMBER 1998

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage range, V_{CC}		– 0.5 V to 7 V
Input voltage range, V_I (see Note 1)		– 0.5 V to 7 V
Output voltage range, V_O (see Note 1)		– 0.5 V to V_{CC} + 0.5 V
Input clamp current, I_{Ik} ($V_I < 0$)		– 20 mA
Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$)		± 20 mA
Continuous output current, I_O ($V_O = 0$ to V_{CC})		± 25 mA
Continuous current through V_{CC} or GND		± 50 mA
Package thermal impedance, θ_{JA} (see Note 2):	D package	127° C/W
	DB package	158° C/W
	DGV package	182° C/W
	N package	78° C/W
	PW package	170° C/W
Storage temperature range, T_{stg}		– 65° C to 150° C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. The input and output voltage ratings may be exceeded if the input and output current ratings are observed.
2. The package thermal impedance is calculated in accordance with JESD 51, except for through-hole packages, which use a trace length of zero.

recommended operating conditions (see Note 3)

			SN54AHC74		SN74AHC74		*UNIT*
			MIN	*MAX*	*MIN*	*MAX*	
V_{CC}	supply voltage		2	5.5	2	5.5	V
V_{IH}	High-level input voltage	V_{CC} = 2 V	1.5		1.5		V
		V_{CC} = 3 V	2.1		2.1		
		V_{CC} = 5.5 V	3.85		3.85		
V_{IL}	Low-level input voltage	V_{CC} = 2 V		0.5		0.5	V
		V_{CC} = 3 V		0.9		0.9	
		V_{CC} = 5.5 V		1.65		1.65	
V_I	Input voltage		0	5.5	0	5.5	V
V_O	Output voltage		0	V_{CC}	0	V_{CC}	V

recommended operating conditions (Continued)

			SN54AHC74		SN74AHC74		*UNIT*
			MIN	*MAX*	*MIN*	*MAX*	
I_{OH}	High-level output current	V_{CC} = 2 V		– 50		– 50	µA
		V_{CC} = 3.3 V ± 0.3 V		– 4		– 4	mA
		V_{CC} = 5 V ± 0.5 V		– 8		– 8	
I_{OL}	Low-level output current	V_{CC} = 2 V		50		50	µA
		V_{CC} = 3.3 V ± 0.3 V		4		4	mA
		V_{CC} = 5 V ± 0.5 V		8		8	
$\Delta t / \Delta v$	Input transition rise or fall rate	V_{CC} = 3.3 V ± 0.3 V		100		100	ns/V
		V_{CC} = 5 V ± 0.5 V		20		20	
T_A	Operating free-air temperature		– 55	125	– 40	85	°C

NOTE 3: All unused inputs of the device must be held at V_{CC} or GND to ensure proper device operation. Refer to the TI application report, *Implications of Slow or Floating CMOS Inputs,* leterature number SCBA004.

SN54AHC74, SN74AHC74
DUAL POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOPS
WITH CLEAR AND PRESET

SCLRS255E – DECEMBER 1995 – REVISED NOVEMBER 1998

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		*TEST CONDITIONS*	V_{CC}	T_A = 25° C			SN54AHC74		SN74AHC74		*UNIT*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
V_{OH}		I_{OH} = – 50 μA	2 V	1.9	2		1.9		1.9		V
			3 V	2.9	3		2.9		2.9		
			4.5 V	4.4	4.5		4.4		4.4		
		I_{OH} = – 4 mA	3 V	2.58			2.48		2.48		
		I_{OH} = – 8 mA	4.5 V	3.94			3.8		3.8		
V_{OL}		I_{OL} = 50 μA	2 V			0.1		0.1		0.1	V
			3 V			0.1		0.1		0.1	
			4.5 V			0.1		0.1		0.1	
		I_{OL} = 4 mA	3 V			0.36		0.5		0.44	
		I_{OL} = 8 mA	4.5 V			0.36		0.5		0.44	
I_I	Data inputs	$V_I = V_{CC}$ or GND	5.5 V			± 0.1		± 1		± 1	μA
	Control inputs					± 0.1		± 1		± 1	
I_{CC}		$V_I = V_{CC}$ or GND, $I_O = 0$	5.5 V			2		20		20	μA
C_i		$V_I = V_{CC}$ or GND	5 V		2	10				10	pF

timing requirements over recommended operating free-air temperature range, V_{CC} = 3.3 V ± 0.3 V (unless otherwise noted) (see Figure 1)

			T_A = 25° C		SN54AHC74		SN74AHC74		
			MIN	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNIT*
t_W	Pulse duration	$\overline{\text{PRE}}$ or $\overline{\text{CLR}}$ low	6		7		7		ns
		CLK	6		7		7		
t_{su}	Setup time before CLK ↑	Data	6		7		7		ns
		$\overline{\text{PRE}}$ or $\overline{\text{CLR}}$ inactive	5		5		5		
t_h	Hold time, data after CLK ↑		0.5		0.5		0.5		ns

timing requirements over recommended operating free-air temperature range, V_{CC} = 5 V ± 0.5 V (unless otherwise noted) (see Figure 1)

			T_A = 25° C		SN54AHC74		SN74AHC74		
			MIN	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNIT*
t_W	Pulse duration	$\overline{\text{PRE}}$ or $\overline{\text{CLR}}$ low	5		5		5		ns
		CLK	5		5		5		
t_{su}	Setup time before CLK ↑	Data	5		5		5		ns
		$\overline{\text{PRE}}$ or $\overline{\text{CLR}}$ inactive	3		3		3		
t_h	Hold time, data after CLK ↑		0.5		0.5		0.5		ns

switching characteristics over recommended operating free-air temperature range, V_{CC} = 3.3 ± 0.3 V (unless otherwise noted) (see Figure 1)

PARAMETER	*FROM (INPUT)*	*TO (OUTPUT)*	*LOAD CAPACITANCE*	SN54AHC74					*UNIT*
				T_A = 25° C			*MIN*	*MAX*	
				MIN	*TYP*	*MAX*			
f_{max}			C_L = 15 pF*	80	125		70		MHz
			C_L = 50 pF	50	75		45		
t_{PLH}*	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 15 pF		7.6	12.3	1	14.5	ns
t_{PLH}*					7.6	12.3	1	14.5	
t_{PLH}*	CLK	Q or $\overline{Q}$	C_L = 15 pF		6.7	11.9	1	14	ns
t_{PHL}*					6.7	11.9	1	14	
t_{PLH}	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 50 pF		10.1	15.8	1	18	ns
t_{PHL}					10.1	15.8	1	18	
t_{PLH}	CLK	Q or $\overline{Q}$	C_L = 50 pF		9.2	15.4	1	17.5	ns
t_{PHL}					9.2	15.4	1	17.5	

* On products compliant to MIL-PRF-38535, this parameter is not production tested.

switching characteristics over recommended operating free-air temperature range, V_{CC} = 3.3 V ± 0.3 V (unless otherwise noted) (see Figure 1)

PARAMETER	*FROM (INPUT)*	*TO (OUTPUT)*	*LOAD CAPACITANCE*	SN74AHC74					*UNIT*
				T_A = 25° C			*MIN*	*MAX*	
				MIN	*TYP*	*MAX*			
f_{max}			C_L = 15 pF	80	125		70		MHz
			C_L = 50 pF	50	75		45		
t_{PLH}	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 15 pF		7.6	12.3	1	14.5	ns
t_{PHL}					7.6	12.3	1	14.5	
t_{PLH}	CLK	Q or $\overline{Q}$	C_L = 15 pF		6.7	11.9	1	14	ns
t_{PHL}					6.7	11.9	1	14	
t_{PLH}	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 50 pF		10.1	15.8	1	18	ns
t_{PHL}					10.1	15.8	1	18	
t_{PLH}	CLK	Q or $\overline{Q}$	C_L = 50 pF		9.2	15.4	1	17.5	ns
t_{PHL}					9.2	15.4	1	17.5	

SN54AHC74, SN74AHC74
DUAL POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOPS
WITH CLEAR AND PRESET

SCLS255E – DECEMBER 1995 – REVISED NOVEMBER 1998

switching characteristics over recommended operating free-air temperature range, V_{CC} = 5 V ± 0.5 V (unless otherwise noted) (see Figure 1)

PARAMETER	*FROM* (*INPUT*)	*TO* (*OUTPUT*)	*LOAD CAPACITANCE*	SN54AHC74					*UNIT*
				T_A = 25° C			*MIN*	*MAX*	
				MIN	*TYP*	*MAX*			
f_{max}			C_L = 15 pF*	130	170		110		MHz
			C_L = 50 pF	90	115		75		
t_{PLH}*	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 15 pF		4.8	7.7	1	9	ns
t_{PHL}*					4.8	7.7	1	9	
t_{PLH}*	CLK	Q or $\overline{Q}$	C_L = 15 pF		4.6	7.3	1	8.5	ns
t_{PHL}*					4.6	7.3	1	8.5	
t_{PLH}	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 50 pF		6.3	9.7	1	11	ns
t_{PHL}					6.3	9.7	1	11	
t_{PLH}	CLK	Q or $\overline{Q}$	C_L = 50 pF		6.1	9.3	1	10.5	ns
t_{PHL}					6.1	9.3	1	10.5	

* On products compliant to MIL-PRF-38535, this parameter is not production tested.

switching characteristics over recommended operating free-air temperature range, V_{CC} = 5 V ± 0.5 V (unless otherwise noted) (see Figure 1)

PARAMETER	*FROM* (*INPUT*)	*TO* (*OUTPUT*)	*LOAD CAPACITANCE*	SN54AHC74					*UNIT*
				T_A = 25° C			*MIN*	*MAX*	
				MIN	*TYP*	*MAX*			
f_{max}			C_L = 15 pF*	130	170		110		MHz
			C_L = 50 pF	90	115		75		
t_{PLH}	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 15 pF		4.8	7.7	1	9	ns
t_{PHL}					4.8	7.7	1	9	
t_{PLH}	CLK	Q or $\overline{Q}$	C_L = 15 pF		4.6	7.3	1	8.5	ns
t_{PHL}					4.6	7.3	1	8.5	
t_{PLH}	$\overline{PRE}$ or $\overline{CLR}$	Q or $\overline{Q}$	C_L = 50 pF		6.3	9.7	1	11	ns
t_{PHL}					6.3	9.7	1	11	
t_{PLH}	CLK	Q or $\overline{Q}$	C_L = 50 pF		6.1	9.3	1	10.5	ns
t_{PHL}					6.1	9.3	1	10.5	

noise characteristics, V_{CC} = 5 V, C_L = 50 pF, T_A = 25º C (see Note 4)

PARAMETER		SN74AHC74		*UNIT*
		MIN	*MAX*	
$V_{OL\,(P)}$	Ouiet output, maximum dynamic V_{OL}		0.8	V
$V_{OL\,(V)}$	Quiet output, minimum dynamic V_{OL}		– 0.8	V
$V_{OH\,(V)}$	Quiet output, minimum dynamic V_{OH}	4.7		V
$V_{IH\,(D)}$	High-level dynamic input voltage	3.5		V
$V_{IL\,(D)}$	Low-level dynamic input voltage		1.5	V

NOTE 4: Characteristics are for surface-mount packages only.

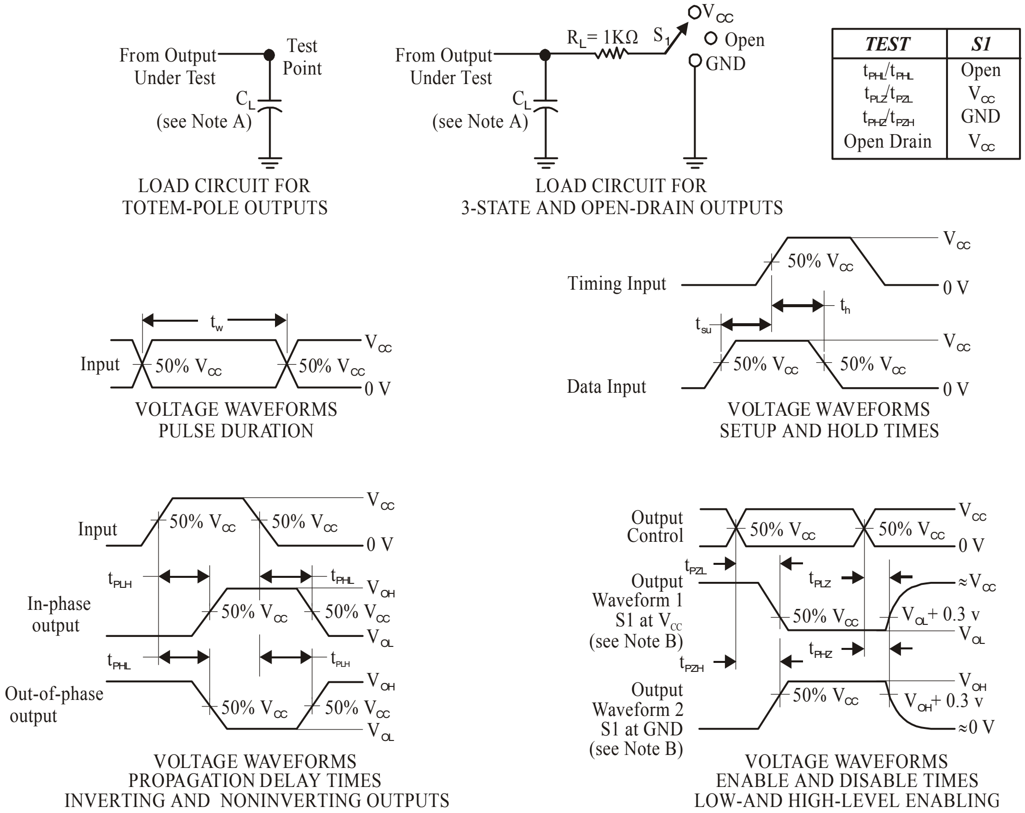
operating characteristics, V_{CC} = 5 V, T_A = 25º C

PARAMETER		*TEST CONDITIONS*	*TYP*	*UNIT*
C_{pd}	Power dissipation capacitance	No load, f = 1 MHz	32	pF

SN54AHC74, SN74AHC74
DUAL POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOPS
WITH CLEAR AND PRESET

SCLS225E – DECEMBER 1995 – REVISED NOVEMBER 1998

PARAMETER MEASURENT INFORMATION



TEST	*S1*
t_{PHL}/t_{PHL}	Open
t_{PLZ}/t_{PZL}	V_{CC}
t_{PHZ}/t_{PZH}	GND
Open Drain	V_{CC}

LOAD CIRCUIT FOR
TOTEM-POLE OUTPUTS

LOAD CIRCUIT FOR
3-STATE AND OPEN-DRAIN OUTPUTS

VOLTAGE WAVEFORMS
PULSE DURATION

VOLTAGE WAVEFORMS
SETUP AND HOLD TIMES

VOLTAGE WAVEFORMS
PROPAGATION DELAY TIMES
INVERTING AND NONINVERTING OUTPUTS

VOLTAGE WAVEFORMS
ENABLE AND DISABLE TIMES
LOW-AND HIGH-LEVEL ENABLING

NOTES:

A. C_L includes probe and jig capacitance.

B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control.

C. All input pulses are supplied by generators having the following characteristics: PRR $\leq$ 1 MHz, Z_O = 50 Ω, $t_r \leq 3$ ns, $t_f \leq 3$ ns.

D. The outputs are measured one at a time with one input transition per measurement.

Figure 1. Load Circuit and Voltage Waveforms

SN54HC138, SN74HC138
3-LINE TO 8-LINE DECODERS/DEMULITIPLEXERS

SCLS107C – DECEMBER 1982 – REVISED MAY 1997

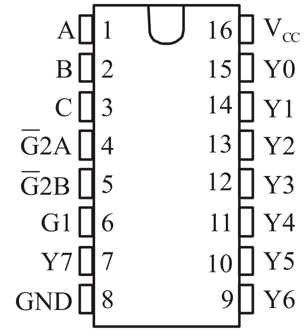
- **Designed Specifically for High-Speed Memory Decoders and Data Transmission Systems**
- **Incorporate Three Enable Inputs to Simplify Cascading and/or Data Reception**
- **Package Options Include Plastic Small-Outline (D), Thin Shrink, Small-Outline (PW), and Ceramic Flat (W) Packages, Ceramic Chip Carriers (FK), and Standard Plastic (N) and Ceramic (J) 300-mil DIPs**

Description

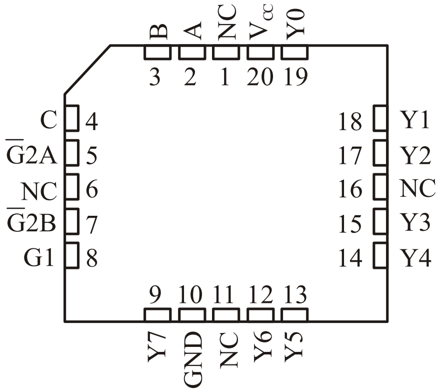
The ’HC138 are designed to be used in high-performance memory-decoding or data-routing applications requiring very short propagation delay times. In high-performance memory systems, these decoders can be used to minimize the effects of system decoding. When employed with high-speed memories utilizing a fast enable circuit, the delay times of these decoders and the enable time of the memory are usually less than the typical access time of the memory. This means that the effective system delay introduced by the decoders is negligible.

The conditions at the binary-select inputs at the three enable inputs select one of eight output lines. Two active-low and one active-high enable inputs reduce the need for external gates or inverters when expanding. A 24-line decoder can be implemented without external inverters and a 32-line decoder requires only one inverter. An enable input can be used as a data input for demultiplexing applications.

SN54HC138. . . J OR W PACKAGE
SN74HC138. . .D, N, OR PW PACKAGE
(TOP VIEW)



SN54HC138 . . . FK PACKAGE
(TOP VIEW)



NC – No internal connection

The SN54HC138 is characterized for operation over the full military temperature range of – 55° C to 125° C. The SN74HC138 is characterized for operation from – 40° C to 85° C.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and dislaimers thereto appears at the end of this data sheet.

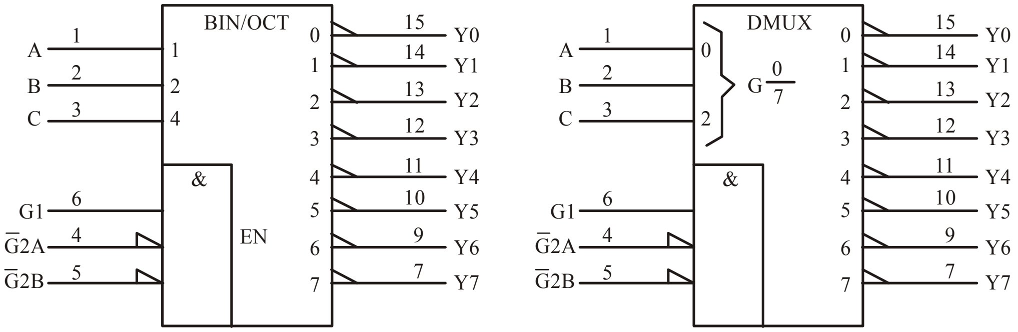
SN54HC138, SN74HC138
3-LINE TO 8-LINE DECODERS/DEMULTIPLEXERS

SCLS107C – DECEMBER 1982 – REVISED MAY 1997

FUNCTION TABLE

INPUTS						*OUTPUTS*							
ENABLE			*SELECT*										
***G*1**	$\overline{G2}$ *A*	$\overline{\mathbf{G2}}$ *B*	***C***	***B***	***A***	***Y*0**	***Y*1**	***Y*2**	***Y*3**	***Y*4**	***Y*5**	***Y*6**	***Y*7**
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

logic symbols (alternatives) †

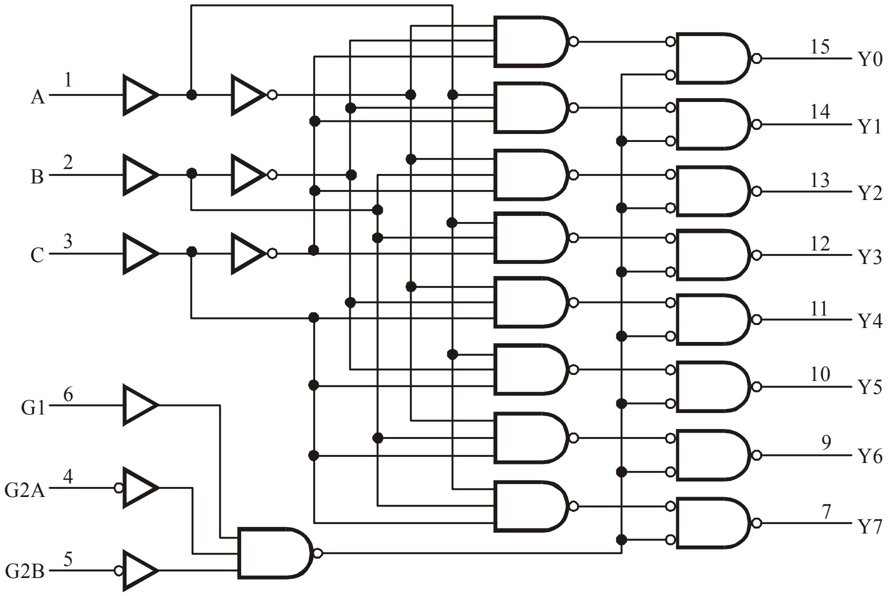


† These symbols are in accordance with ANSI/IEEE std 91-1984 and IEC Publication 617-12.

Pin numbers shown are for the D, J, N, PW, and W packages.

SN54HC138, SN74HC138
3-LINE TO 8-LINE DECODERS/DEMULTIPLEXERS

logic diagram (positive logic)



Pin numbers shown are for the D, J, N, PW, and W packages.

absolute maximum ratings over operating free-air temperature range†

Supply voltage range, V_{CC}	– 0.5 V to 7 V
Input clamp current, I_{Ik} ($V_I < 0$ or $V_I > V_{CC}$) (see Note 1)	± 20 mA
Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) (see Note 1)	± 20 mA
Continuous output current, I_O ($V_O = 0$ to V_{CC})	± 25 mA
Continuous current through V_{CC} or GND	± 50 mA
Package thermal impedance, θ_{JA} (see Note 2): D package	113º C/W
N package	78º C/W
PW package	149º C/W
Storage temperature range, T_{stg}	– 65º C to 150º C

† Stresses beyond those listed under “absolute maximum ratings” may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under “recommended operating conditions” is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1. The input and output voltage ratings may be exceeded if the input and output current ratings are observed.

2. The package thermal impedance is calculated in accordance with JESD 51, except for through-hole packages, which use a trace length of zero.

SN54HC138, SN74HC138
3-LINE TO 8-LINE DECODERS/DEMULTIPLEXERS

SCLS107C – DECEMBER 1982 – REVISED MAY 1997

recommended operating conditions

			SN54HC138			SN74HC138			UNIT
			MIN	*NOM*	*MAX*	*MIN*	*NOM*	*MAX*	
V_{CC}	supply voltage		2	5	6	2	5	6	V
V_{IH}	High-level input voltage	$V_{CC} = 2$ V	1.5			1.5			V
		$V_{CC} = 4.5$ V	3.15			3.15			
		$V_{CC} = 6$ V	4.2			4.2			
V_{IL}	Low-level input voltage	$V_{CC} = 2$ V	0		0.5	0		0.5	V
		$V_{CC} = 4.5$ V	0		1.35	0		1.35	
		$V_{CC} = 6$ V	0		1.8	0		1.8	
V_I	Input voltage		0		V_{CC}	0		V_{CC}	V
V_O	Output voltage		0		V_{CC}	0		V_{CC}	V
t_t	Input transition (rise and fall) time	$V_{CC} = 2$ V	0		1000	0		1000	ns
		$V_{CC} = 4.5$	0		500	0		500	
		$V_{CC} = 6$ V	0		400	0		400	
T_A	Operating free-air temperature		– 55		125	– 40		85	ºC

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	*TEST CONDITIONS*		V_{CC}	$T_A = 25º$ C			SN54HC138		SN74HC138		*UNIT*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
V_{OH}	$V_I = V_{IH}$ or V_{IL}	$I_{OH} = – 20$ μA	2 V	1.9	1.998		1.9		1.9		V
			4.5 V	4.4	4.499		4.4		4.4		
			6 V	5.9	5.999		5.9		5.9		
		$I_{OH} = – 4$ mA	4.5 V	3.98	4.3		3.7		3.84		
		$I_{OH} = – 5.2$ mA	6 V	5.48	5.8		5.2		5.34		
V_{OL}	$V_I = V_{IH}$ or V_{IL}	$I_{OL} = 20$ μA	2 V		0.002	0.1		0.1		0.1	V
			4.5 V		0.001	0.1		0.1		0.1	
			6 V		0.001	0.1		0.1		0.1	
		$I_{OL} = 4$ mA	4.5 V		0.17	0.26		0.4		0.33	
		$I_{OL} = 5.2$ mA	6 V		0.15	0.26		0.4		0.33	
I_I	$V_I = V_{CC}$ or 0		6 V		± 0.1	± 100		± 1000		± 1000	nA
I_{CC}	$V_I = V_{CC}$ or 0, $I_O = 0$		6 V			8		160		80	μA
C_i			2 V to 6 V		3	10		10		10	pF

SN54HC138, SN74HC138
3-LINE TO 8-LINE DECODERS/DEMULTIPLEXERS

SCLS107C - DECEMBER 1982 - REVISED MAY 1997

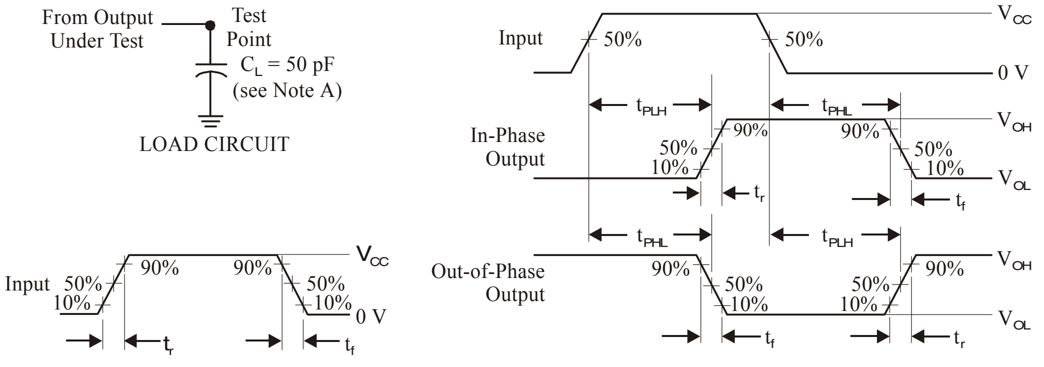
Switching characteristics over recommended operating free-air temperature range, C_L = 50 pF (unless otherwise noted) (see Figure 1)

PARAMETER	*FROM (INPUT)*	*TO (OUTPUT)*	V_{CC}	T_A = 25° C			SN54HC138		SN74HC138		*UNIT*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
t_{pd}	A, B or C	Any Y	2V		67	180		270		225	ns
			4.5 V		18	36		54		45	
			6 V		15	31		46		38	
	Enable	Any Y	2 V		66	155		235		195	
			4.5 V		18	31		47		39	
			6 V		15	26		40		33	
t_t		Any	2 V		38	75		110		95	ns
			4.5 V		8	15		22		19	
			6 V		6	13		19		16	

operating characteristics, T_A = 25° C

PARAMETER		*TEST CONDITIONS*	*TYP*	*UNIT*
c_{pd}	Power dissipation capacitance	No load	85	pF

PARAMETER MEASUREMENT INFORMATION



LOAD CIRCUIT

VOLTAGE WAVEFORM
INPUT RISE AND FALL TIMES

VOLTAGE WAVEFORMS
PROPAGATION DELAY AND OUTPUT TRANSITION TIMES

NOTE:

A. C_L includes probe and test-fixture capacitance.

B. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by generators having the following characteristics: PRR ≤ 1 MHz, Z_O = 50 Ω, t_r = 6 ns, t_f = 6 ns.

C. The outputs are measured one at a time with one input transition per measurement.

D. t_{PLH} and t_{PHL} are the same as tpd.

Figure 1. Load Circuit and Voltage Waveforms

TEXAS
INSTRUMENTS

CD74HC151, CD74HCT151

Data sheet acquired from Harris Semiconductor
SCHS150

September 1997

High Speed CMOS Logic 8-Input Multiplexer

Features

- **Complementary Data Outputs**
- **Buffered Inputs and Outputs**
- **Fanout (Over Temperature Range)**
  - **Standard Outputs..................... 10 LSTTL Loads**
  - **Bus Driver Outputs .... 15 LSTTL Loads**
- **Wide Operating Temperature Range ... – 55° C to 125° C**
- **Balanced Propagation Delay and Transition Times**
- **Significant Power Reduction Compared to LSTTL Logic ICs**
- **Alternate Source is Philips/Signetics**
- **HC Types**
  - **2 V to 6 V Operation**
  - **High Noise Immunity: N_{IL} = 30%, N_{IH} = 30% of V_{CC} at V_{CC} = 5 V**
- **HCT Types**
  - **4.5 V to 5.5 V Operation**
  - **Direct LSTTL Input Logic Compatibility, V_{IL} = 0.8 V (Max), V_{IH} = 2V (Min)**
  - **CMOS Input Compatibility, $I_I \leq$ 1μA at V_{OL}, V_{OH}**

Description

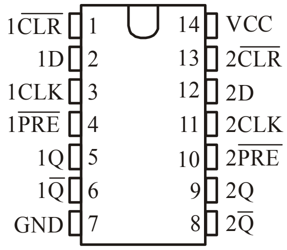
The Harris CD54HC151 and CD74HCT151 are single 8-channel digital multiplexers having three binary control inputs, S0, S1 and S2 and an active low enable (E) input. The three binary signals select 1 to 8 channels. Outputs are both inverting $(\overline{Y})$ and non-inverting (Y).

Ordering Information

PART NUMBER	*TEMP. RANGE (°C)*	*PACKAGE*	*PKG. NO.*

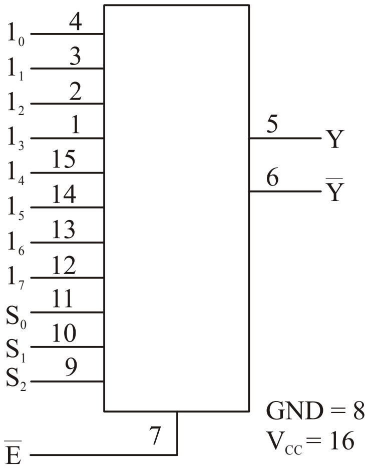
Pinout

SN54AHC74. . . J OR W PACKAGE
SN74AHC74. . .D, DB, DGV, N, OR PW PACKAGE
(TOP VIEW)



CD74HC151, CD74HCT151

Functional Diagram



TRUTH TABLE

SELECT INPUTS			***DATA INPUTS***								***ENABLE***	***OUTPUT***	
S2	**S1**	**S0**	**I0**	**I$\overline{1}$**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**$\overline{E}$**	**$\overline{Y}$**	**Y**
X	X	X	X	X	X	X	X	X	X	X	H	H	L
L	L	L	L	X	X	X	X	X	X	X	L	H	L
L	L	L	H	X	X	X	X	X	X	X	L	L	H
L	L	H	X	L	X	X	X	X	X	X	L	H	L
L	L	H	X	H	X	X	X	X	X	X	L	L	H
L	H	L	X	X	L	X	X	X	X	X	L	H	L
L	H	L	X	X	H	X	X	X	X	X	L	L	H
L	H	H	X	X	X	L	X	X	X	X	L	H	L
L	H	H	X	X	X	H	X	X	X	X	L	L	H
H	L	L	X	X	X	X	L	X	X	X	L	H	L
H	L	L	X	X	X	X	H	X	X	X	L	L	H
H	L	H	X	X	X	X	X	L	X	X	L	H	L
H	L	H	X	X	X	X	X	H	X	X	L	L	H
H	H	L	X	X	X	X	X	X	L	X	L	H	L
H	H	L	X	X	X	X	X	X	H	X	L	L	H
H	H	H	X	X	X	X	X	X	X	L	L	H	L
H	H	H	X	X	X	X	X	X	X	H	L	L	H

NOTE : H = High Voltage Level, L = Low Voltage Level, X = Don't Care

CD74HC151, CD74HCT151

Absolute Maximum Ratings

DC Supply Voltage, V_{CC}.................... – 0.5 to 7 V

DC Input Diode Current, I_{IK}

For $V_I < – 0.5$ V or $V_I > V_{CC} + 0.5$ V................. ± 20mA

DC Output Diode Current, I_{OK}

For $V_O < – 0.5$ V or $V_O > V_{CC} + 0.5$ V................ ± 20 mA

DC Output Source or Sink Current per Output Pin, I_O

For $V_O > – 0.5$ V or $V_O < V_{CC} + 0.5$ V................ ± 25 mA

DC V_{CC} or Ground Current, I_{CC} or I_{GND} .... ± 50 mA

Operating Conditions

Temperature Range (T_A)............... – 55º C to 125º C

Supply Voltage Range, V_{CC}

HC Types.. 2V to 6V

HCT Types................................4.5 V to 5.5 V

DC Input or Output Voltage, V_I, V_O)......................... 0 V to V_{CC}

Input Rise and Fall time

2 V.. 1000ns (Max)

4.5 V.. 500ns (Max)

6 V... 400 ns (Max)

Thermal Information

Thermal Resistance (Typical, Note 3) θ_{JA} (ºC/W)

PDIP Package............................. 100

CERDIP Package....................... 115

SOIC Package............................. 150º C

Maximum Junction Temperature .... 150º C

Maximum Storage Temperature Range.......... .... – 65º C to 150º C

Maximum Lead Temperature (Soldering 10s)... 300º C (SOIC - Lead Tips Only)

CAUTION: Stresses above those listed in “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

3. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

DC Electrical Specifications

PARAMETER	*SYMBOL*	*TEST CONDITIONS*		V_{CC} *(V)*	*25º C*			*– 40º C TO 85º C*		*– 55ºC TO 125ºC*		*UNITS*
		V_I*(V)*	I_o *(mA)*		*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
HC TYPES												
High Level Input Voltage	V_{IH}	–	–	2	1.5	–	–	1.5	–	1.5	–	V
				4.5	3.15	–	–	3.15	–	3.15	–	V
				6	4.2	–	–	4.2	–	4.2	–	V
Low Level Input Voltage	V_{IL}	–	–	2	–	–	0.5	–	0.5	–	0.5	V
				4.5	–	–	1.35	–	1.35	–	1.35	V
				6	–	–	1.8	–	1.8	–	1.8	V

CD74HC151, CD74HCT151

DC Electrical Specifications (Cont.)

PARAMETER	SYMBOL	TEST CONDITIONS			25° C			– 40° C TO + 85° C		– 55°C TO 125°C		UNITS
		$V_I(V)$	I_o (mA)	V_{CC} (V)	MIN	TYP	MAX	MIN	MAX	MIN	MAX	
HC TYPES												
High Level Output	V_{OH}	V_{IH}	– 0.02	2	1.9	–	–	1.9	–	1.9	–	V
Voltage		or V_{IL}	– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
CMOS Loads			– 0.02	6	5.9	–	–	5.9	–	5.9	–	V
High Level Output			–	–	–	–	–	–	–	–	–	V
Voltage			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
TTL Loads			– 5.2	6	5.48	–	–	5.34	–	5.2	–	V
Low Level Output	V_{OL}	V_{IH}	0.02	2	–	–	0.1	–	0.1	–	0.1	V
Voltage		or V_{IL}	0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
CMOS Loads			0.02	6	–	–	0.1	–	0.1	–	0.1	V
Low Level Output			–	–	–	–	–	–	–	–	–	V
Voltage			4	4.5	–	–	0.26	–	0.33	–	0.4	V
TTL Loads			5.2	6	–	–	0.26	–	0.33	–	0.4	V
Input Leakage	I_I	V_{CC}	–	6	–	–	± 0.1	–	± 1	–	± 1	µA
Current		or GND										
Quiescent Device	I_{CC}	V_{CC}	0	6	–	–	8	–	80	–	160	µA
Current		or GND										
HCT TYPES												
High Level Input	V_{IH}	–	–	4.5	2	–	–	2	–	2	–	V
Voltage				to 5.5								
Low Level Input	V_{IL}	–	–	4.5	–	–	0.8	–	0.8	–	0.8	V
Voltage				to 5.5								
High Level Output	V_{OH}	V_{IH}	– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
Voltage		or V_{IL}										
CMOS Loads												
High Level Output			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
Voltage												
TTL Loads												
Low Level Output	V_{OL}	V_{IH}	0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
Volt. CMOS Loads		or V_{IL}										
LOW Level Output			4	4.5	–	–	0.26	–	0.33	–	0.4	V
Voltage												
TTL Loads												
Input Leakage	I_I	V_{CC}	0	5.5	–	–	± 0.1	–	± 1	–	± 1	µA
Current		and										
		GND										
Quiescent Device	I_{CC}	V_{CC}	0	5.5	–	–	8	–	80	–	160	µA
Current		or GND										
Additional Quiescent	ΔI_{CC}	V_{CC}	–	4.5	–	100	360	–	450	–	490	µA
Device Current Per		– 2.1		to 5.5								
Input Pin: 1 Unit Load												

NOTE: For dual-supply systems theoretical worst case (V_I = 2.4 V, V_{CC} = 5.5 V) specification is 1.8 mA.

CD74HC151, CD74HCT151

HCT Input Loading Table

INPUT	*UNIT LOADS*
Select	1.5
Data	0.45
Enable	0.3

NOTE: Unit Load is ΔI_{CC} limit specified in DC Electrical Table, e.g., 360µA max at 25° C.

Switching Specifications Input t_r, t_f = 6ns

PARAMETER	*SYMBOL*	*TEST CONDITIONS*	V_{CC} *(V)*	*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
HC TYPES											
Propagation Delay (Fig.1) Any Data Input to Y	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	170	–	215	–	255	ns
			4.5	–	–	34	–	43	–	51	ns
		C_L = 15pF	5	–	14	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	29	–	37	–	43	ns
*Any Data Input to $\overline{Y}$	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	185	–	230	–	280	ns
			4.5	–	–	37	–	46	–	56	ns
		C_L = 15pF	5	–	15	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	31	–	39	–	48	ns
Any Select to Y	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	185	–	230	–	280	ns
			4.5	–	–	37	–	46	–	56	ns
		C_L = 15pF	5	–	15	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	31	–	39	–	48	ns
Any Select to $\overline{Y}$	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	205	–	255	–	310	ns
			4.5	–	–	41	–	51	–	62	ns
		C_L = 15pF	5	–	17	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	35	–	43	–	53	ns
Enable to Y	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	140	–	175	–	210	ns
			4.5	–	–	28	–	35	–	42	ns
		C_L = 15pF	5	–	11	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	24	–	30	–	36	ns

CD74HC151, CD74HCT151

Swtiching Specifications Input t_r, t_f = 6ns **(Continued)**

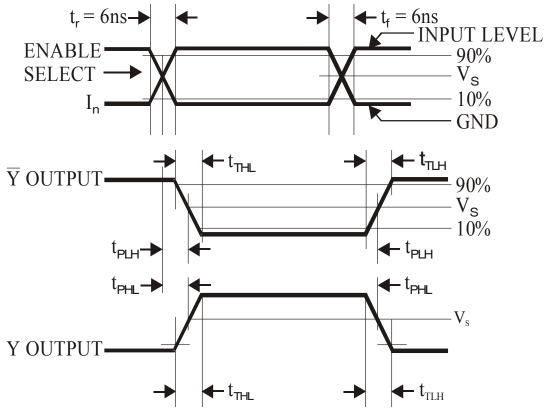
PARAMETER	*SYMBOL*	*TEST CONDITIONS*	*V_{CC} (V)*	*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
Enable to $\overline{Y}$	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	–	145	–	180	–	220	ns
			4.5	–	–	29	–	36	–	44	ns
		C_L = 15pF	5	–	12	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	25	–	31	–	38	ns
Output Transition Time	t_{TLH}, t_{THL}	C_L = 50pF	2	–	–	75	–	95	–	110	ns
			4.5	–	–	15	–	19	–	22	ns
			6	–	–	13	–	10	–	10	ns
Input Capacitance	C_{IN}	–	–	–	–	10	–	10	–	10	pF
Power Dissipation Capacitance (Notes 4, 5)	C_{PD}	–	5	–	59	–	–	–	–	–	pF
HCT TYPES											
Propagation Delay,	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	–	39	–	48	–	57	ns
Any Data Input (Fig. 2)		C_L = 15pF	5	–	16	–	–		–	–	ns
Any Data Input to $\overline{Y}$	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	–	36	–	45	–	54	ns
		C_L = 15pF	5	–	15	–	–	–	–	–	ns
Any Select to Y	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–		41	–	51	–	62	ns
		C_L = 15pF	5	–	17	–	–	–	–	–	ns
Any Select to $\overline{Y}$	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	–	43	–	54	–	65	ns
		C_L = 15pF	5	–	18	–	–	–	–	–	ns
Enable to Y	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	–	29	–	36	–	44	ns
		C_L = 15pF	5	–	12	–	–	–	–	–	ns
Enable to $\overline{Y}$	C_L = 50pF	C_L = 50pF	4.5	–	–	36	–	46	–	54	ns
	C_L = 15pF	C_L = 15pF	5	15	–	–	–	–	–	–	ns
Output Transition Time	t_{TLH}, t_{THL}	C_L = 50pF	4.5	–	–	15	–	19	–	22	ns
Input Capacitance	C_{IN}	–	–	–	–	10	–	10	–	10	pF
Power Dissipation Capacitance (Notes 4, 5)	C_{PD}	–	5		58	–	–	–	–	–	pF

NOTES:

4. C_{PD} is used to determine the dynamic power consumption, per gate.
5. $P_D = V_{CC}^2 f_i (C_{PD} + C_L)$ where f_I = input frequency, C_L = output load capacitance, V_{CC} = supply voltage.

CD74HC151, CD74HCT151

Test Circuit and Waveform



TEXAS INSTRUMENTS

CD74HC161,CD74HCT161 CD74HC163, CD74HCT163

Data sheet acquired from Harris Semiconductor
SCHS154

February 1998

High Speed CMOS Logic Presettable Counters

Features

- **CD74HC161, CD74HCT161 4-Bit Binary Counter, Asynchronous Reset**
- **CD74HC163, CD74HCT163 4-Bit Binary Counter, Synchronous Reset**
- **Synchronous Counting and Loading**
- **Two Count Enable Inputs for n-Bit Cascading**
- **Look-Ahead Carry for High-Speed Counting**
- **Fanout (Over Temperature Range)**
  - **Standard Outputs.. 10 LSTTL Loads**
  - **Bus Driver Outputs .... 15 LSTTL Loads**
- **Wide Operating Temperature Range................ – 55º C to 125º C**
- **Balanced Propagation Delay and Transition Times**
- **Significant Power Reduction Compared to LSTTL Logic ICs**
- **HC Types**
  - **2 V to 6 V Operation**
  - **High Noise Immunity: N_{IL} = 30%, N_{IH} = 30% of V_{CC} at V_{CC} = 5 V**
- **HCT Types**
  - **4.5 V to 5.5 V Operation**
  - **Direct LSTTL Input Logic Compatibility, V_{IL} = 0.8 V (Max), V_{IH} = 2 V (Min)**
  - **CMOS Input Compatibility, $I_l \le 1\mu A$ at V_{OL}, V_{OH}**

Ordering Information

PART NUMBER	*TEMP. RANGE (ºC)*	*PACKAGE*	*PKG. NO.*
CD74HC161E	– 55 TO 125	16 Ld PDIP	E 16.3
CD74HC161M	– 55 to 125	16 Ld SOIC	E 16.15
CD74HC163E	– 55 to 125	16 Ld PDIP	M 16.3
CD74HC163M	– 55 to 125	16 Ld SOIC	M 16.15
CD74HCT161E	– 55 to 125	16 Ld PDIP	E 16.3
CD74HCT161M	– 55 to 125	16 Ld SOIC	M 16.15
CD74HCT163E	– 55 to 125	16 Ld PDIP	E16.3
CD74HCT163E	– 55 to 125	16 Ld SOIC	M16.15

NOTE:

1. When ordering, use the entire part number. Add the suffix 96 to obtain the variant in the tape and reel.
2. Wafer and die for this part number is available which meets all electrical specifications. Please contact your local sales office or Harris customer service for ordering information.

Description

The Harris CD74HC161, CD74HCT161, CD74HC163 and CD74HCT163 are presettable synchronous counters that feature look-ahead carry logic for use in high-speed counting applications. The CD74HC161 and CD74HCT161 are asynchronous reset decade and binary counters, respectively; the CD74H163 and CD74C163 devices decade and binary counters, respectively and are reset synchronously with the clock. Counting and parallel presetting are both accomplished synchronously with the negative-to-positive transition of the clock.

A low level on the synchronous parallel enable input, SPE, disables counting operation and allows data at the P0 to P3 inputs to be loaded into the counter (provided that the setup and hold requirements for SPE are met).

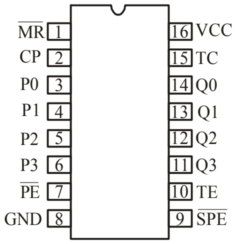
All counters are reset with a low level on the Master Reset input, MR. In the CD74HC163 and CD74HCT163 counters (synchronous reset types), the requirements for setup and hold time with respect to the clock must be met. Two count enables, PE and TE, in each counter are provided for n-bit cascading. In all counters reset action occurs regardless of the level of the SPE , PE and TE inputs (and the clock input, CP. in the CD74HC161 and CD74HCT161 types).

If a decade counter is preset to an illegal state or assumes an illegal state when power is applied, it will return to the normal sequence in one count as shown in state diagram.

The look-ahead carry feature simplifies serial cascading of the counters. Both count enable inputs (PE and TE) must be high to count. The TE input is gated with the Q outputs of all four stages so that at the maximum count the terminal count (TC) output goes high for one clock period. This TC pulse is used to enable the next cascaded stage.

Pinout

CD74HC161, CD74HCT161, CD74HC163,CD74HCT163
(PDIP, SOIC)
(TOP VIEW)



CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

Absolute Maximum Ratings

DC Supply Voltage, V_{CC}.................... – 0.5 V to 7 V

DC Input Diode Current, I_{IK}

For $V_I < – 0.5$ V or $V_I > V_{CC} + 0.5$ V................. ± 20mA

DC Output Diode Current, I_{OK}

For $V_O < – 0.5$ V or $V_O > V_{CC} + 0.5$ V................ ± 20 mA

DC Drain Current, per Output, IO

For – 0.5 V < VO < VCC + 0.5 V ....± 25 mA

DC Output Source or Sink Current per Output Pin, I_O

For $V_O > – 0.5$ V or $V_O < V_{CC} + 0.5$ V............ ± 25 mA

DC V_{CC} or Ground Current, I_{CC}.................. ± 50 mA

Operating Conditions

Temperature Range, T_A............... – 55º C to 125º C

Supply Voltage Range, V_{CC}

HC Types.. 2V to 6V

HCT Types................................4.5 V to 5.5 V

DC Input or Output Voltage, V_I, V_O........ 0 V to V_{CC}

Input Rise and Fall time

2 V.. 1000ns (Max)

4.5 V.. 500ns (Max)

6 V.. 400 ns (Max)

Thermal Information

Thermal Resistance (Typical, Note 6) θ_{JA} (ºC/W)

PDIP Package.............................	90
SOIC Package...........................	160º C

Maximum Junction Temperature .... 150º C

Maximum Storage Temperature Range.......... .... – 65º C to 150º C

Maximum Lead Temperature (Soldering 10s).. 300º C (SOIC - Lead Tips Only)

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

6. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

DC Electrical Specifications

PARAMETER	SYMBOL	TEST CONDITIONS		V_{CC} (V)	25º C			– 40º C TO 85º C		– 55ºC TO 125ºC		UNITS
		V_I(V)	I_o (mA)		MIN	TYP	MAX	MIN	MAX	MIN	MAX	
HC TYPES												
High Level Input Voltage	V_{IH}	–	–	2	1.5	–	–	1.5	–	1.5	–	V
				4.5	3.15	–	–	3.15	–	3.15	–	V
				6	4.2	–	–	4.2	–	4.2	–	V
Low Level Input Voltage	V_{IL}	–	–	2	–	–	0.5	–	0.5	–	0.5	V
				4.5	–	–	1.35	–	1.35	–	1.35	V
				6	–	–	1.8	–	1.8	–	1.8	V

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

DC Electrical Specifications (Contd.)

		TEST CONDITIONS			*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		
PARAMETER	*SYMBOL*	*V_I(V)*	*I_o (mA)*	*V_{CC} (V)*	*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNITS*
HC TYPES												
High Level Output Voltage CMOS Loads	V_{OH}	V_{IH} or V_{IL}	– 0.02	2	1.9	–	–	1.9	–	1.9	–	V
			– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
			– 0.02	6	5.9	–	–	5.9	–	5.9	–	V
High Level Output Voltage TTL Loads			–	–	–	–	–	–	–	–	–	V
			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
			– 5.2	6	5.48	–	–	5.34	–	5.2	–	V
Low Level Output Voltage CMOS Loads	V_{OL}	V_{IH} or V_{IL}	0.02	2	–	–	0.1	–	0.1	–	0.1	V
			0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
			0.02	6	–	–	0.1	–	0.1	–	0.1	V
Low Level Output Voltage TTL Loads			–	–	–	–	–	–	–	–	–	V
			4	4.5	–	–	0.26	–	0.33	–	0.4	V
			5.2	6	–	–	0.26	–	0.33	–	0.4	V
Input Leakage Current	I_I	V_{CC} or GND	–	6	–	–	± 0.1	–	± 1	–	± 1	µA
Quiescent Device Current	I_{CC}	V_{CC} or GND	0	6	–	–	8	–	80	–	160	µA
HCT TYPES												
High Level Input Voltage	V_{IH}	–	–	4.5 to 5.5	2	–	–	2	–	2	–	V
Low Level Input Voltage	V_{IL}	–	–	4.5 to 5.5	–	–	0.8	–	0.8	–	0.8	V
High Level Output Voltage CMOS Loads	V_{OH}	V_{IH} or V_{IL}	– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
High Level Output Voltage TTL Loads			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
Low Level Output Voltage CMOS Loads	V_{OL}	V_{IH} or V_{IL}	0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
LOW Level Output Voltage TTL Loads			4	4.5	–	–	0.26	–	0.33	–	0.4	V
Input Leakage Current	I_I	V_{CC} and GND	0	5.5	–	–	± 0.1	–	± 1	–	± 1	µA
Quiescent Device Current	I_{CC}	V_{CC} or GND	0	5.5	–	–	8	–	80	–	160	µA
Additonal Quiescent Device Current Per Load Input Pin: 1 Unit	ΔI_{CC} (Note)	V_{CC} – 2.1	–	4.5 to 5.5	–	100	360	–	450	–	490	µA

NOTE: For dual-supply systems theoretical worst case (V_I = 2.4 V, V_{CC} = 5.5 V) specification is 1.8 mA.

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

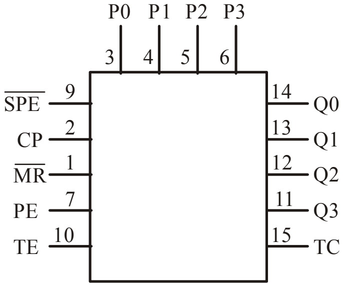
HCT Inputs Loading Table

INPUT	*UNIT LOADS*
P0 – P3	0.25
PE	0.65
CP	1.05
$\overline{\text{MR}}$	0.8
$\overline{\text{SPE}}$	0.5
TE	1.05

NOTE: Unit Load is ΔI_{CC} limit specified in DC Electrical Table, e.g., 360µA max at 25º C.

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

Functional Diagram



MODE SELECT - FUNCTION TABLE FOR CD74HC/HCT161

	INPUTS						*OUTPUTS*	
OPERATING MODE	$\overline{MR}$	***CP***	***PE***	***TE***	$\overline{SPE}$	P_n	Q_n	***TC***
Reset (Clear)	L	X	X	X	X	X	L	L
Parallel Load	H	↑	X	X	l	l	L	L
	H	↑	X	X	l	h	H	(Note 3)
Count	H	↑	h	h	h (Note 5)	X	Count	(Note 3)
Inhibit	H	X	l (Note 4)	X	h (Note 5)	X	q_n	(Note 3)
	H	X	X	l (Note 4)	h (Note 5)	X	q_n	L

MODE SELECT - FUNCTION TABLE FOR CD74HC/HCT163

	INPUTS						*OUTPUTS*	
OPERATING MODE	$\overline{MR}$	***CP***	***PE***	***TE***	$\overline{SPE}$	P_n	Q_n	***TC***
Reset (Clear)	l	↑	X	X	X	X	L	L
Parallel Load	h (Note 5)	↑	X	X	l	l	L	L
	h (Note 5)	↑	X	X	l	h	H	(Note 3)
Count	h (Note 5)	↑	h	h	h (Note 5)	X	Count	(Note 3)
Inhibit	h (Note 5)	X	l (Note 4)	X	h (Note 5)	X	q_n	(Note 3)
	h (Note 5)	X	X	l (Note 4)	h (Note 5)	X	q_n	L

NOTE: H = High voltage level steady state; L = Low voltage level steady state; h = High voltage level one setup time prior to the Low-to-High clock transition; l = Low voltage level one setup time prior to the Low-to-High clock transition; X = Don't Care; q = Lower case letters indicate the state of the referenced output prior to the Low-to-High clock transition; ↑ = Low-to-High clock transition.

3. The TC output is High when TE is High and the counter is at Terminal Count (HHH:H for CD74HC/HCT161 and CD74HC/HCT163).
4. The High-to-Low transition of PE or TE on the CD74HC/HCT161 and the CD74HC/HCT163 should only occur while CP is HIGH for conventional operation.
5. The Low-to-High transition of $\overline{SPE}$ on the CD74HC/HCT161 and $\overline{SPE}$ or $\overline{MR}$ on the CD74HC/HCT163 should only occur while CP is HIGH for conventional operation.

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

Prerequisite For Switching Specifications

				25° C			*– 40° C TO 85° C*		*– 55°C TO 125°C*		
PARAMETER	*SYMBOL*	*TEST COND.*	*V_{CC} (V)*	*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNITS*
HC TYPES											
Maximum CP Frequency (Note 7)	f_{MAX}	–	2	6	–	–	5	–	4	–	MHz
			4.5	30	–	–	24	–	20	–	MHz
			6	35	––	28	–	24	–	MHz	
CP Width (LOW)	$t_{W (L)}$	–	2	80	–	–	100	–	120	–	ns
			4.5	16	–	–	20	–	24	–	ns
			6	14	–	–	17	–	20	–	ns
$\overline{MR}$ Pulse Width (161)	t_W	–	2	100	–	–	125	–	150	–	ns
			4.5	20	–	–	25	–	30	–	ns
			6	17	–	–	21	–	26	–	ns
Setup Time, Pn to CP	t_{SU}	–	2	60	–	–	75	–	90	–	ns
			4.5	12	–	–	15	–	18	–	ns
			6	10	–	–	13	–	15	–	ns
Setup Time, PE or TE to CP	t_{SU}	–	2	50	–	–	65	–	75	–	ns
			4.5	10	–	–	13	–	15	–	ns
			6	9	–	–	11	–	13	–	ns
Setup Time, $\overline{SPE}$ to CP	t_{SU}	–	2	60	–	–	75	–	90	–	ns
			4.5	12	–	–	15	–	18	–	ns
			6	10	–	–	13	–	15	–	ns
Setup Time, $\overline{MR}$ to CP (163)	t_{SU}	–	2	65	–	–	80	–	100	–	ns
			4.5	13	–	–	16	–	20	–	ns
			6	11	–	–	14	–	17	–	ns
Hold Time, PN to CP	t_H	–	2	3	–	–	3	–	3	–	ns
			4.5	3	–	–	3	–	3	–	ns
			6	3	–	–	3	–	3	–	ns
Hold Time, TE or PE to CP	t_H		2	0	–	–	0	–	0	–	ns
			4.5	0	–	–	0	–	0	–	ns
			6	0	–	–	0	–	0	–	ns
Hold Time, $\overline{SPE}$ to CP	t_H	–	2	0	–	–	0	–	0	–	ns
			4.5	0	–	–	0	–	0	–	ns
			6	0	–	–	0	–	0	–	sn
Recovery Time, $\overline{MR}$ to CP (161)	t_{REC}	–	2	75	–	–	95	–	110	–	ns
			4.5	15	–	–	19	–	22	–	ns
			6	13	–	–	16	–	19	–	ns

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

Prerequisite For Switching Specifications (Continued)

PARAMETER	*SYMBOL*	*TEST COND.*	V_{CC} *(V)*	*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
HCT TYPES											
Maximum CP Frequency	f_{MAX}	–	4.5	30	–	–	24	–	20	–	MHz
CP Width (Low) (Note 7)	$t_{W\,(L)}$	–	4.5	16	–	–	20	–	24	–	ns
$\overline{MR}$ Pulse Width (161)	t_W	–	4.5	20	–	–	25	–	30	–	ns
Setup Time, Pn to CP	t_{SU}	–	4.5	10	–	–	13	–	15	–	ns
Setup Time, PE or TE to CP	t_{SU}	–	4.5	13	–	–	16	–	20	–	ns
Setup Time, $\overline{SPE}$ to CP	t_{SU}	–	4.5	12	–	–	15	–	18	–	ns
Setup Time, $\overline{MR}$ to CP (163)	t_{SU}	–	4.5	13	–	–	16	–	20	–	ns
Hold Time, PN to CP	t_H	–	4.5	5	–	–	5	–	5	–	ns
Hold Time, TE or PE to CP	t_H	–	4.5	3	–	–	3	–	3	–	ns
Hold Time, $\overline{SPE}$ to CP	t_H	–	4.5	3	–	–	3	–	3	–	ns
Recovery Time, $\overline{MR}$ to CP (161)	t_{REC}	–	4.5	15	–	–	19	–	22	–	ns

NOTE:

7. Applies to non-cascaded operation only. With cascaded counters clock to terminal count propagation delays, count enables (PE or TE)-to-clock setup times, and count enables (PE or TE)-to-clock hold times determine maximum clock frequency. For example with these HC devices:

$$f_{max}\ (\text{CP}) = \frac{1}{\text{CP-to-TC prop. delay} + \text{TE-to-CP setup} + \text{TE-to-CP Hold}} = \frac{1}{37+10+0} = 21\,\text{MHz}\,(\text{min})$$

Switching Specifications C_L = 50pF, Input t_r, t_f = 60ns

PARAMETER	*SYMBOL*	*TEST CONDITIONS*	V_{CC} *(V)*	*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
HC TYPES											
Propagation Delay CP to TC	t_{PHL}, t_{PLH}	C_L = 50pF	2	–	–	185	–	230	–	280	ns
			4.5	–	–	37	–	46	–	56	ns
		C_L = 15pF	5	–	15	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	31	–	39	–	48	ns

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

Switching Specifications C_L = 50pF, Input t_r, t_f = 6ns **(Continued)**

PARAMETER	*SYMBOL*	*TEST CONDITIONS*	V_{CC} *(V)*	*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
CP toQn	t_{PHL}, t_{PLH}	C_L = 50pF	2	–	–	185	–	230	–	280	ns
			4.5	–	–	37	–	46	–	56	ns
		C_L = 15pF	5	–	15	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	31	–	39	–	48	ns
TE to TC	t_{PHL}, t_{PLH}	C_L = 50pF	2	–	–	120	–	150	–	180	ns
			4.5	–	–	24	–	30	–	36	ns
		C_L = 15pF	5	–	9	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	20	–	26	–	31	ns
$\overline{MR}$ to Qn (161)	t_{PHL}	C_L = 50pF	2	–	–	210	–	265	–	315	ns
			4.5	–	–	42	–	53	–	63	ns
		C_L = 15pF	5	–	18	–	–	–	–	–	ns
		C_L = 50pF	6	–	–	36	–	45	–	54	ns
$\overline{MR}$ to TC (161)	t_{PHL}	C_L = 50pF	2	–	–	210	–	265	–	315	ns
			4.5	–	–	42	–	53	–	63	ns
		C_L = 50pF	6	–	–	36	–	45	–	54	ns
Output Transition Time	t_{THL} t_{TLH}	C_L = 50pF	2	–	–	75	–	95	–	110	ns
			4.5	–	–	15	–	19	–	22	ns
			6	–	–	13	–	16	–	19	ns
Power Dissipation Capacitance (Notes 8, 9)	C_{PD}	–	5	–	60	–	–	–	–	–	pF
Input Capacitance	C_{IN}	C_L = 50pF	–	10	–	10	–	10	–	10	pF
HCT TYPES											
Propagation Delay CP to TC	t_{PHL}, t_{PLH}	C_L = 50pF	4.5	–	–	42	–	53	–	63	ns
		C_L = 15pF	5	–	18	–	–		–	–	ns
CP to Qn	t_{PHL}, t_{PLH}	C_L = 50pF	4.5	–	–	39	–	49	–	59	ns
		C_L = 15pF	5	–	16	–	–	–	–	–	ns
TE to TC	t_{PHL}, t_{PLH}	C_L = 50pF	4.5	–	–	32	–	40	–	48	ns
		C_L = 15pF	5	–	13	–	–	–	–	–	ns

CD74HC161, CD74HCT161, CD74HC163, CD74HCT163

Switching Specifications C_L = 50pF, Input t_r, t_f = 6ns (Continued)

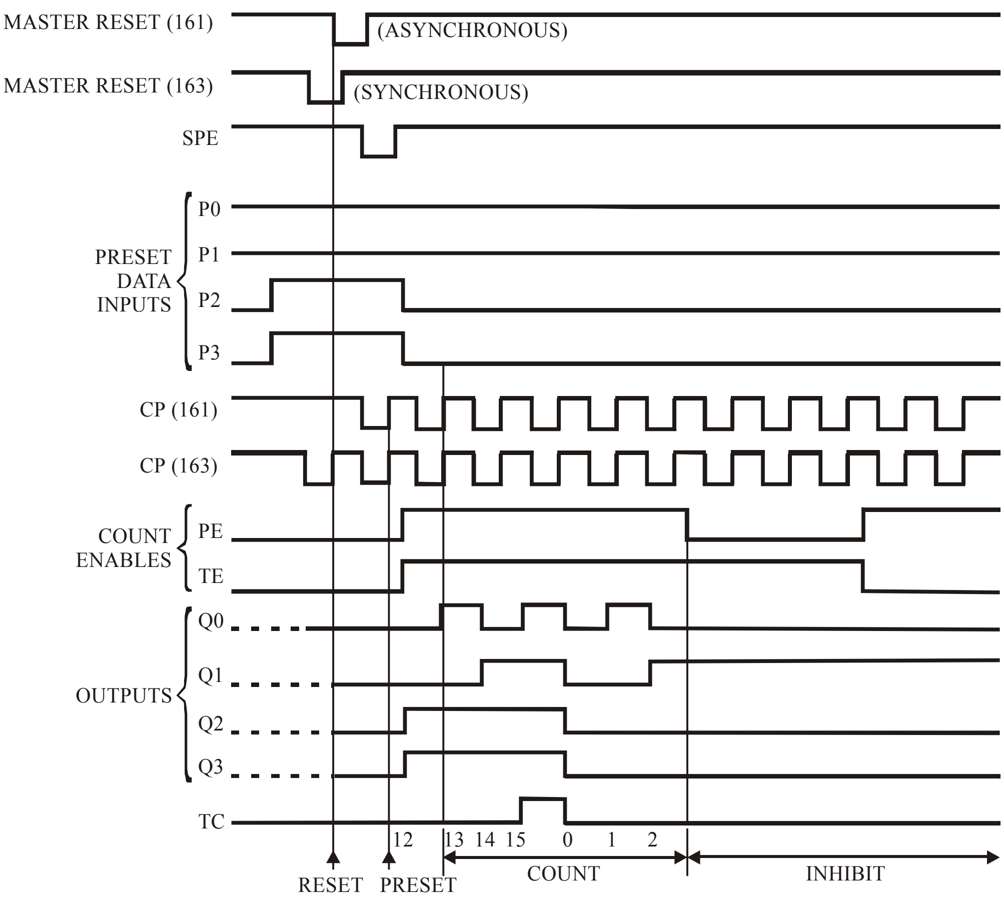
PARAMETER	*SYMBOL*	*TEST CONDITIONS*	V_{CC} *(V)*	*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
				MIN	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
$\overline{MR}$ to Qn (161)	t_{PHL}	C_L = 50pF	4.5	–	–	50	–	63	–	75	ns
		C_L = 15pF	5	–	21	–	–	–	–	–	ns
$\overline{MR}$ to TC (161)	t_{PHL}	C_L = 50pF	4.5	–	–	50	–	63	–	75	ns
Output Transition Time	t_{THL} t_{TLH}	C_L = 50pF	4.5	–	–	15	–	19	–	22	ns
Power Dissipation Capacitance (Notes 6, 7)	C_{PD}	–	5	–	63	–	–	–	–	–	pF
Input Capacitance	C_{IN}	C_L = 50pF	–	10	–	10	–	10	–	10	pF

NOTES:

8. C_{PD} is used to determine the dynamic power consumption, per package.
9. $P_D = C_{PD} V_{cc}^2 f_i\ 2\ (C_L V_{CC}^2 f_0)$ where f_l = Input Frequency, f_0 = Output Frequency, C_L = Output Load Capacitance, V_{CC} = Supply Voltage.

CD74HC161, CD74HCT161, CD64HC163, CD74HCT163

Timing Diagram



Sequence Illustrated on waveforms:

1. Reset outputs to zero.
2. Preset to binary twelve.
3. Count to thirteen, fourteen, fifteen, zero, one and two.
4. Inhibit.

TEXAS INSTRUMENTS

CD74HC165, CD74HCT165

Data sheet acquired from Harris Semiconductor
SCHS156
February 1998

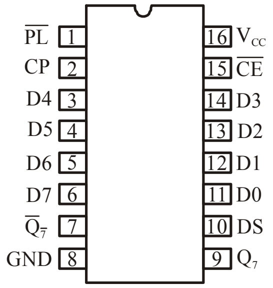
High Speed CMOS Logic
8-Bit Parallel-In/Serial-Out Shift Register

Features

- **Buffered Inputs**
- **Asynchronous Parallel Load**
- **Complementary Outputs**
- **Typical f_{MAX} = 60MHz at V_{CC} = 5 V, C_L = 15 pF, T_A = 25º C**
- **Fanout (Over Temperature Range)**
  - ◆ **Standard Outputs..................... 10 LSTTL Loads**
  - ◆ **Bus Driver Outputs .... 15 LSTTL Loads**
- **Wide Operating Temperature Range .... – 55º C to 125º C**
- **Balanced Propagation Delay and Transition Times**
- **Significant Power Reduction Compared to LSTTL Logic ICs**
- **HC Types**
  - ◆ **2 V to 6 V Operation**
  - ◆ **High Noise Immunity: N_{IL} = 30%, N_{IH} = 30% of V_{CC} at V_{CC} = 5 V**
- **HCT Types**
  - ◆ **4.5 V to 5.5 V Operation**
  - ◆ **Direct LSTTL Input Logic Compatibility, V_{IL} = 0.8 V (Max), V_{IH} = 2V (Min)**
  - ◆ **CMOS Input Compatibility, $I_l \leq$ 1µA at V_{OL}, V_{OH}**

Pinout

CD74HC165, CD74HCT165
(PDIP, SOIC)
TOP VIEW



CD74HC165, CD74HCT165

Description

The Harris CD74HC165 and CD74HCT165 are 8-bit parallel or serial-in shift registers with complementary serial outputs (Q7 and $\overline{Q_7}$) available from the last stage. When the parallel load (PL) input is LOW, parallel data from the D0 to D7 inputs are loaded into the register asynchronously. When the $\overline{PL}$ is HIGH, data enters the register serially at the DS input and shifts one place to the right $(Q_0 \rightarrow Q_1 \rightarrow Q_2, \text{etc.})$ with each positive-going clock transition. This feature allow parallel-to-serial converter expansion by typing the Q_7 output to the DS input of the succeeding device.

For predictable operation the LOW-to-HIGH transition of $\overline{CE}$ should only take place while CP is HIGH. Also, CP and d $\overline{CE}$ should be LOW before the LOW-to-HIGH transition of PL to prevent shifting the data when $\overline{PL}$ goes HIGH.

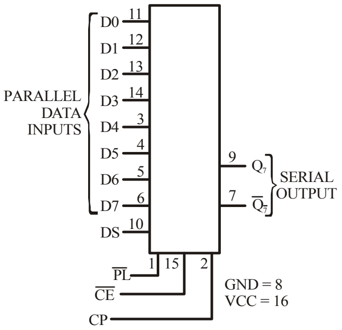
Ordering Information

PART NUMBER	***TEMP. RANGE (°C)***	***PACKAGE***	***PKG. NO.***
CD74HC165E	– 55 to 125	16 Ld PDIP	E 16.3
CD74HCT165E	– 55 to 125	16 Ld PDIP	E 16.3
CD74HC165M	– 55 to 125	16 Ld SOIC	M 16.15
CD74HCT165M	– 55 to 125	16 Ld SOIC	M 16.15

NOTE:

1. When ordering, use the entire part number. Add the suffix 96 to obtain the variant in the tape and reel.
2. Wafer and die is available which meets all electrical specifications. Please contact your local sales office or Harris customer service for ordering information.

Functional Diagram



TRUTH TABLE

OPERATING MODE	***INPUTS***					***Q_n REGISTER***		***OUTPUTS***	
	$\overline{PL}$	***$\overline{CE}$***	***CP***	***DS***	***D_0-D_7***	***Q_0***	***Q_1-Q_6***	***Q_7***	***$\overline{Q_7}$***
Parallel Load	L	X	X	X	L	L	L-L	L	H
	L	X	X	X	H	H	H-H	H	L
Serial Shift	H	L	↑	l	X	L	Q_0 - q_5	q_6	$\overline{q_6}$
	H	L	↑	h	X	H	q_0 - q_5	q_6	$\overline{q_6}$
Hold Do Nothing	H	H	X	X	X	q_0	q_1 - q_6	q_7	$\overline{q_7}$

NOTE:

H = High Voltage Level
h = High Voltage Level One Set-up Time Prior To The Low-to-high Clock Transition
l = Low Voltage Level One Set-up Time Prior To The Low-to-high Clock Transition
L = Low Voltage Level
X = Don't Care
↑ = Transition from Low to High Level
q_n = Lower Case Letters indicate the State Of the Reference Output Clock Transition

CD74HC165, CD74HCT165

Absolute Maximum Ratings

DC Supply Voltage, V_{CC}.................... – 0.5 V to 7 V

DC Input Diode Current, I_{IK}

For $V_I < -0.5$ V or $V_I > V_{CC} + 0.5$ V...... ± 20mA

DC Output Diode Current, I_{OK}

For $V_O < -0.5$ V or $V_O > V_{CC} + 0.5$ V................ ± 20 mA

DC Drain Current per Output, I_O

For $V_O < -0.5$ V $V_O > V_{CC} + 0.5$ V ....± 25 mA

DC Output Source or Sink Current per Output Pin, I_O

For $V_O > -0.5$ V or $V_O < V_{CC} + 0.5$ V................ ± 25 mA

DC V_{CC} or Ground Current, I_{CC} or I_{GND} .....± 50 mA

Operating Conditions

Temperature Range (T_A)............... – 55° C to 125° C

Supply Voltage Range, V_{CC}

HC Types.. 2V to 6V

HCT Types................................4.5 V to 5.5 V

DC Input or Output Voltage, V_I, V_O)....... 0 V to V_{CC}

Input Rise and Fall Time

2 V.. 1000ns (Max)

4.5 V... 500ns (Max)

6 V.. 400 ns (Max)

Thermal Information

Thermal Resistance (Typical, Note 3)	θ_{JA} (ºC/W)
PDIP Package.............................	90
SOIC Package...........................	115

Maximum Junction Temperature .... 150º C

Maximum Storage Temperature Range.......... .... – 65º C to 150º C

Maximum Lead Temperature (Soldering 10s).. 300º C

(SOIC - Lead Tips Only)

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

3. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

DC Electrical Specifications

PARAMETER	*SYMBOL*	*TEST CONDITIONS*			*25º C*			*– 40º C TO 85º C*		*– 55ºC TO 125ºC*		*UNITS*
		$V_I(V)$	*I_o (mA)*	*V_{CC} (V)*	*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
HC TYPES												
High Level Input Voltage	V_{IH}	–	–	2	1.5	–	–	1.5	–	1.5	–	V
				4.5	3.15	–	–	3.15	–	3.15	–	V
				6	4.2	–	–	4.2	–	4.2	–	V
Low Level Input Voltage	V_{IL}	–	–	2	–	–	0.5	–	0.5	–	0.5	V
				4.5	–	–	1.35	–	1.35	–	1.35	V
				6	–	–	1.8	–	1.8	–	1.8	V

CD74HC165, CD74HCT165

DC Electrical Specifications (Contd.)

		TEST CONDITIONS			*25° C*			*– 40° C TO 85° C*		*– 55°C TO 125°C*		
PARAMETER	*SYMBOL*	*V_I(V)*	*I_o (mA)*	*V_{CC} (V)*	*MIN*	*TYP*	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	*UNITS*
High Level Output Voltage CMOS Loads	V_{OH}	V_{IH} or V_{IL}	– 0.02	2	1.9	–	–	1.9	–	1.9	–	V
			– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
			– 0.02	6	5.9	–	–	5.9	–	5.9	–	V
High Level Output Voltage TTL Loads			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
			– 5.2	6	5.48	–	–	5.34	–	5.2	–	V
Low Level Output Voltage CMOS Loads	V_{OL}	V_{IH} or V_{IL}	0.02	2	–	–	0.1	–	0.1	–	0.1	V
			0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
			0.02	6	–	–	0.1	–	0.1	–	0.1	V
Low Level Output Voltage TTL Loads			4	4.5	–	–	0.26	–	0.33	–	0.4	V
			5.2	6	–	–	0.26	–	0.33	–	0.4	V
Input Leakage Current	I_I	V_{CC} or GND	–	6	–	–	± 0.1	–	± 1	–	± 1	μA
Quiescent Device Current	I_{CC}	V_{CC} or GND	0	6	–	–	8	–	80	–	160	μA
HCT TYPES												
High Level Input Voltage	V_{IH}	–	–	4.5 to 5.5	2	–	–	2	–	2	–	V
Low Level Input Voltage	V_{IL}	–	–	4.5 to 5.5	–	–	0.8	–	0.8	–	0.8	V
High Level Output Voltage CMOS Loads	V_{OH}	V_{IH} or V_{IL}	– 0.02	4.5	4.4	–	–	4.4	–	4.4	–	V
High Level Output Voltage TTL Loads			– 4	4.5	3.98	–	–	3.84	–	3.7	–	V
Low Level Output Voltage CMOS Loads	V_{OL}	V_{IH} or V_{IL}	0.02	4.5	–	–	0.1	–	0.1	–	0.1	V
LOW Level Output Voltage TTL Loads			4	4.5	–	–	0.26	–	0.33	–	0.4	V
Input Leakage Current	I_I	V_{CC} and GND	0	5.5	–	–	± 0.1	–	± 1	–	± 1	μA
Quiescent Device Current	I_{CC}	V_{CC} or GND	0	5.5	–	–	8	–	80	–	160	μA
Additional Quiescent Device Current Per Input Pin: 1 Unit Load	ΔI_{CC} (Note)	V_{CC} – 2.1	–	4.5 to 5.5	–	100	360	–	450	–	490	μA

NOTE: 4. For dual-supply systems theoretical worst case (V_I = 2.4 V, V_{CC} = 5.5 V) specification is 1.8mA.

HCT Input Loading Table

INPUT	*UNIT LOADS*
DS, D0 to D7	0.35
CP, $\overline{PL}$	0.65

NOTE: Unit Load is ΔI_{CC} limit specified in DC Electrical specifications Table, e.g., 360µA max at 25º C.

CD74HC165, CD74HCT165

Prerequisite For Switching Specifications

PARAMETER	SYMBOL	V_{CC} (V)	25° C		– 40° C TO 85° C		– 55ºC TO 125ºC		UNITS
			MIN	MAX	MIN	MAX	MIN	MAX	
HC TYPES									
CP Pulse Width	f_{WL}, t_{WH}	2	80	–	100	–	120	–	ns
		4.5	16	–	20	–	24	–	ns
		6	14	–	17	–	20	–	ns
$\overline{PL}$ Pulse Width	t_{WL}	2	80	–	100	–	120	–	ns
		4.5	16	–	20	–	24	–	ns
		6	14	–	17	–	20	–	ns
Set- up Time	t_{SU}	2	80	–	100	–	120	–	ns
DS to CP		4.5	16	–	20	–	24	–	ns
		6	14	–	17	–	20	–	ns
$\overline{CE}$ to CP	$t_{SU(L)}$	2	80	–	100	–	120	–	ns
		4.5	16	–	20	–	24	–	ns
		6	14	–	17	–	20	–	ns
D0-D7 to $\overline{PL}$	t_{SU}	2	80	–	100	–	120	–	ns
		4.5	16	–	20	–	24	–	ns
		6	14	–	17	–	20	–	ns
Hold Time	t_H	2	35	–	45	–	55	–	ns
DS to CP or $\overline{CE}$		4.5	7	–	9	–	11	–	ns
		6	6	–	8	–	9	–	ns
$\overline{CE}$ to CP	t_H	2	0	–	0	–	0	–	na
		4.5	0	–	0	–	0	–	ns
		6	0	–	0	–	0	–	ns
Recovery Time	t_{REC}	2	100	–	125	–	150	–	ns
$\overline{PL}$ to CP		4.5	20	–	25	–	30	–	ns
		6	17	–	21	–	26	–	ns
Maximum Clock Pulse	f_{MAX}	2	6	–	5	–	4		MHz
Frequency		4.5	30	–	24	–	20	–	Mhz
		6	35	–	28	–	24	–	MHz
HCT TYPES									
CP Pulse Width	t_{WL}, t_{WH}	4.5	18	–	23	–	27	–	ns
$\overline{PL}$ Pulse Width	t_{WL}	4.5	20	–	25	–	30	–	ns
Set-up Time DS to CP	t_{SU}	4.5	20	–	25	–	30	–	ns

CD74HC165, CD74HCT165

Prerequisite For Switching Specifications (Continued)

PARAMETER	*SYMBOL*	V_{CC} *(V)*	*25° C*		*– 40° C TO 85° C*		*– 55°C TO 125°C*		*UNITS*
			MIN	*MAX*	*MIN*	*MAX*	*MIN*	*MAX*	
$\overline{CE}$ to CP	$t_{SU(L)}$	4.5	20	–	25	–	30	–	ns
D0-D7 to $\overline{PL}$	t_{SU}	6	20	–	25	–	30	–	ns
Hold Time DS to CP or $\overline{CE}$	t_H	4.5	7	–	9	–	11	–	ns
$\overline{CE}$ to CP	t_S, t_H	4.5	0	–	0	–	0	–	ns
Recovery Time $\overline{PL}$ to CP	t_{REC}	4.5	20	–	25	–	30	–	ns
Maximum Clock Pulse Frequency	f_{MAX}	4.5	27	–	22	–	18	–	MHz

CD74HC165, CD74HCT165

Switching Specifications Input t_r, t_f = 6ns

				25° C		*– 40° C TO 85° C*	*– 55°C TO 125°C*	
PARAMETER	*SYMBOL*	*TEST CONDITIONS*	V_{CC} *(V)*	*TYP*	*MAX*	*MAX*	*MAX*	*UNITS*
HC TYPES								
Propagation Delay	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	165	205	250	ns
CP or $\overline{CE}$ to Q_7 or Q_7			4.5	–	33	41	50	ns
		C_L = 15pF	5	13	–	–	–	ns
		C_L = 50pF	6	–	28	35	43	ns
$\overline{PL}$ to Q_7 or $\overline{Q}_7$	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	175	220	265	ns
			4.5	–	35	44	53	ns
		C_L = 15pF	5	14	–	–	–	ns
		C_L = 50pF	6	–	30	37	45	ns
D_7 to Q_7 or $\overline{Q}_7$	t_{PLH}, t_{PHL}	C_L = 50pF	2	–	150	190	225	ns
			4.5	–	30	38	45	ns
		C_L = 15pF	5	12	–	–	–	ns
		C_L = 50pF	6	–	26	33	38	ns
Output Transition Times	t_{TLH}, t_{THL}	C_L = 50pF	2	–	75	95	110	ns
			4.5	–	15	19	22	ns
			6	–	13	16	19	ns
Input Capacitance	C_{IN}	–	–	–	10	10	10	pF
Power Dissipation Capacitance (Notes 5, 6)	C_{PD}	–	5	17	–	–	–	pF
HCT TYPES								
Propagation Delay, CP or $\overline{CE}$ to Q_7 or $\overline{Q}_7$	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	40	50	60	ns
		C_L = 15pF	5	17	–	–	–	ns
$\overline{PL}$ to Q_7 or $\overline{Q}_7$	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	40	50	60	ns
		C_L = 15pF	5	17	–	–	–	ns
D_7 to Q_7 or $\overline{Q}_7$	t_{PLH}, t_{PHL}	C_L = 50pF	4.5	–	35	44	53	ns
		C_L = 15pF	5	14	–	–	–	ns
Output Transition Times	t_{TLH}, t_{THL}	C_L = 50pF	4.5	–	15	19	22	ns
Input Capacitance	C_{IN}	C_L = 50pF	–	–	10	10	10	pF
Power Dissipation capasitance (Notes 5, 6)	C_{PD}	–	5	24	–	–	–	pF

NOTES:

5. C_{PD} is used to determine the dynamic power consumption, per package.
6. $P_D = V_{CC}^2 f_i + \Sigma \; (C_L V_{CC}^2 + f_O)$ where f_i = Input Frequency, f_O = Output Frequency, C_L = Output Load Capacitance, V_{CC} = Supply Voltage.

Test Circuits and Waveforms

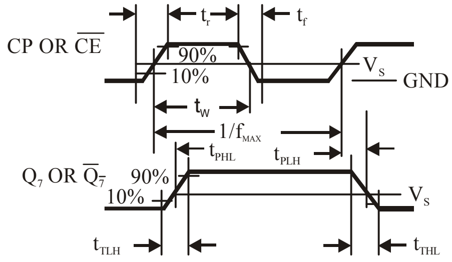


FIGURE 3. SERIAL-SHIFT MODE

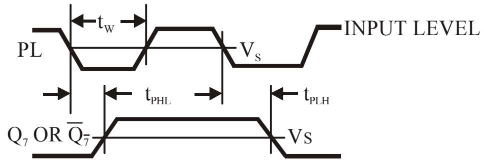


FIGURE 4. PARALLEL-LOAD MODE

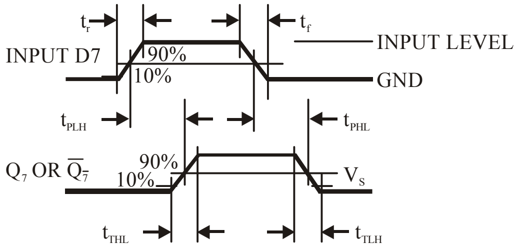


FIGURE 5. PARALLEL-LOAD MODE

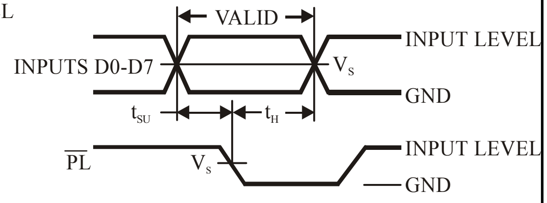


FIGURE 6. PARALLEL-LOAD MODE

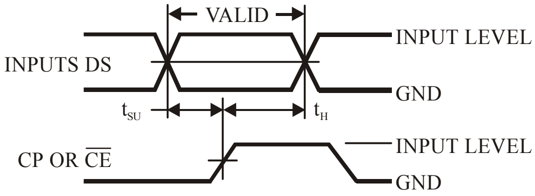


FIGURE 7. SERIAL-SHIFT MODE

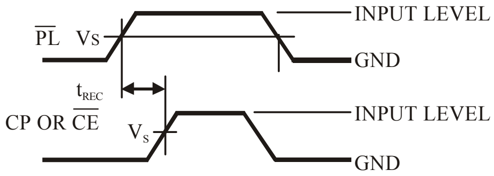


FIGURE 8. SERIAL-SHIFT MODE

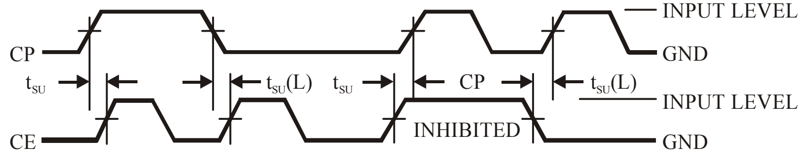


FIGURE 9. SERIAL-SHIFT, CLOCK-INHIBIT MODE

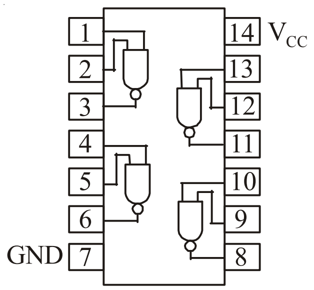
Appendix B

List of 7400TTL Series Integrated Circuits (includes standard TTL, LS, ALS, F and HCMOS ICs)

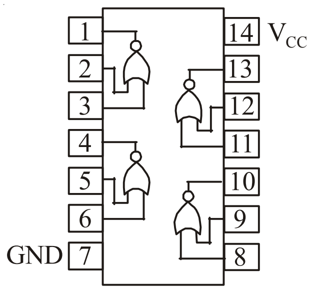
7400	Quad 2-input NAND gate
7402	Quad 2-input NOR gate
7404	Hex inverter
7408	Quad 2-input AND gate
7411	Triple 3-input AND gate
7414	Hex inverter Schmitt trigger
7421	Dual 4-input AND gate
7427	Triple 3-input NOR gate
7430	8-input NAND gate
7432	Quad 2-input OR gate
7442	BCD-to-decimal decoder
7447	BCD-to-7 segment decoder/driver
7474	Dual D-type flip-flop
7475	Quad bistable latch
7476	Dual J-K flip-flop
7483	4-bit full-adder
7485	4-bit magnitude comparator
7486	Quad 2-input exclusive-OR gate
7490	Decade counter
7492	Divide-by-twelve counter
7493	4-bit ripple counter
74106	Dual J-K falling edge triggered flip-flop
74112	Dual J-K rising edge triggered flip-flop
74121	Monortable Multivibrator
74123	Dual retriggerable monostable multivibrator
74138	1-of-8 decoder/multiplexer
74147	10-line-to-4-line priority encoder
72148	8-input priority encoder
74150	16-input multiplexer
74151	8-input multiplexer
74154	1-of-16 decoder/demultiplexers
74160	4-bit binary counter
74164	8-bit serial-in parallel-out shift register
74165	8-bit serial/parallel-in serial-out shift register
74184	BCD-to-binary converter
74185	Binary-to-BCD converter
74190	Presettable BCD decade up/down counter
74191	Presettable 4-bit binary up/down counter
74192	Presettable BCD decade up/down counter
74193	Presettable 4-bit binary up/down counter
74238	1-of-8 demultiplexer

Appendix C

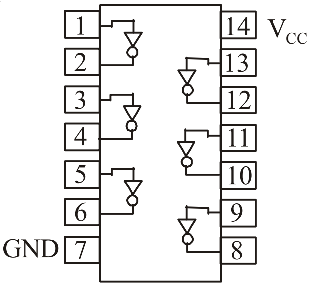
TTL PIN CONFIGURATIONS



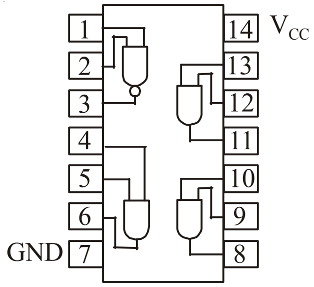
7400



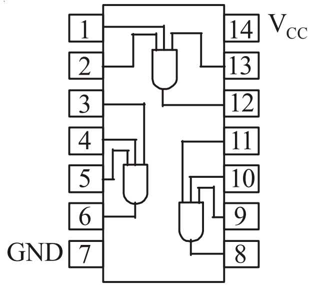
7402



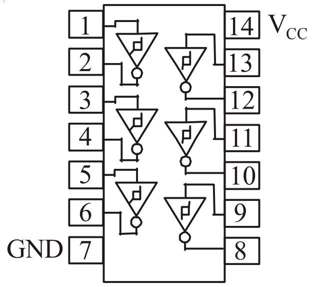
7404



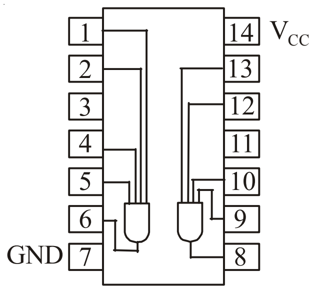
7408



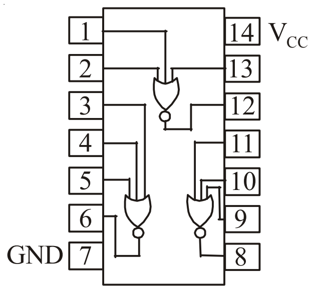
7411



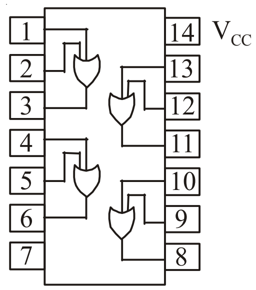
7414



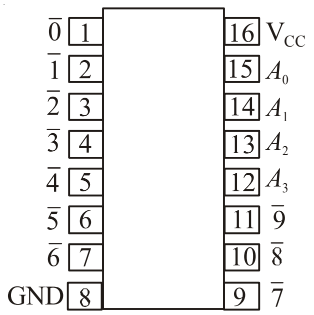
7421



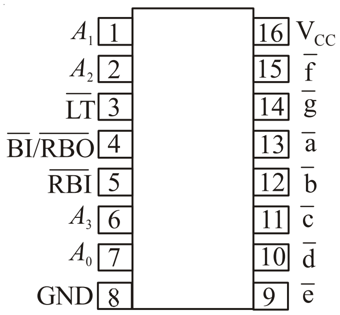
7427



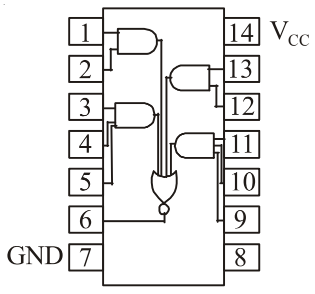
7432



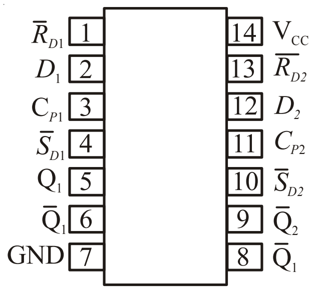
7442



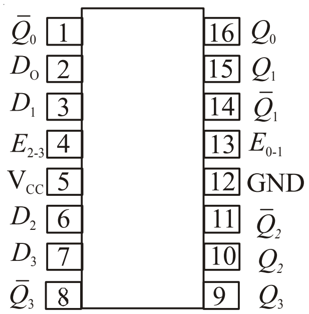
7447



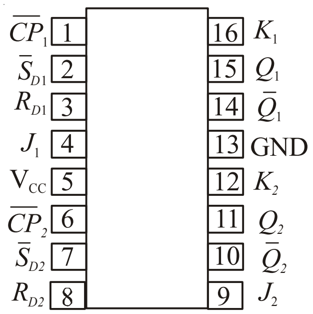
7454



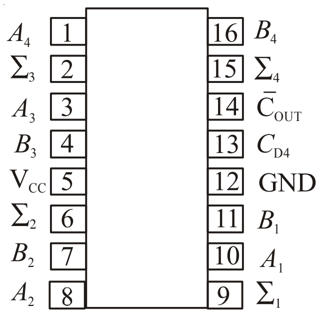
7474



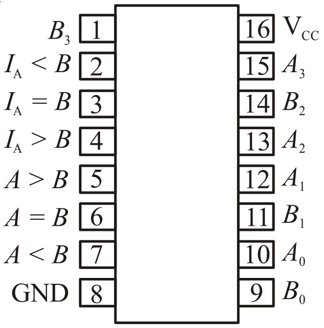
7475



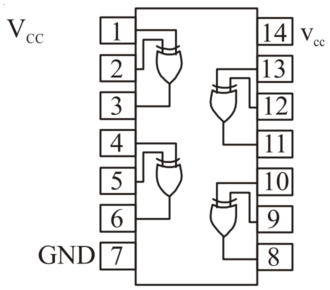
7476



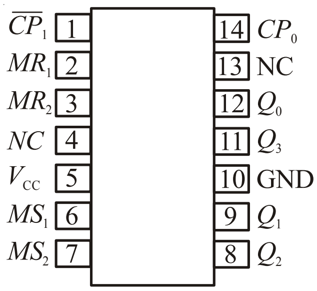
7483



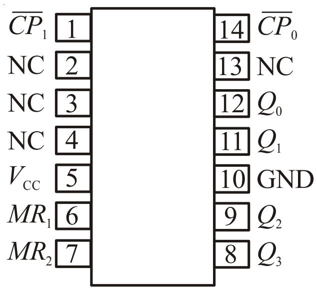
7485



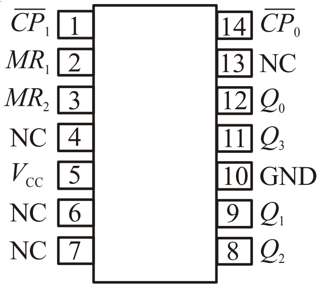
7486



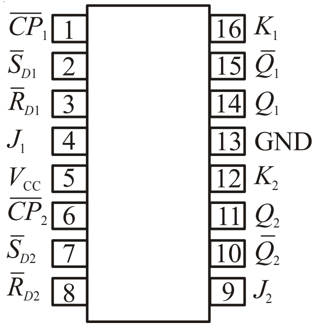
7490



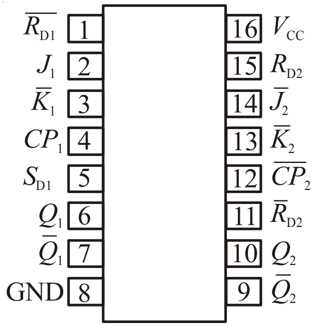
7492



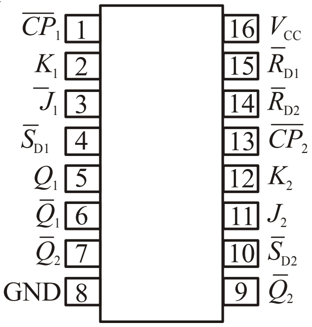
7493



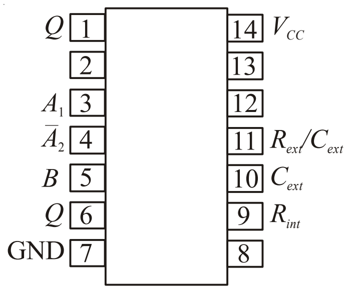
74106



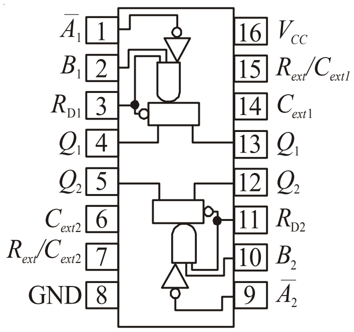
74109



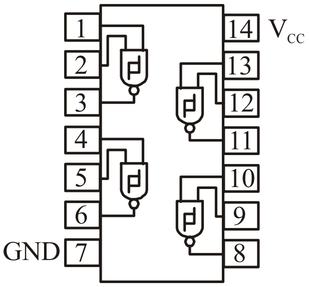
74112



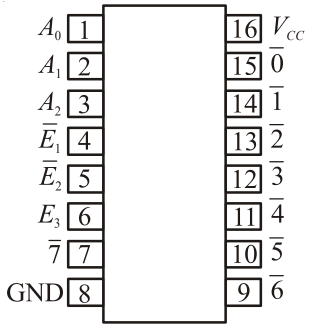
74121



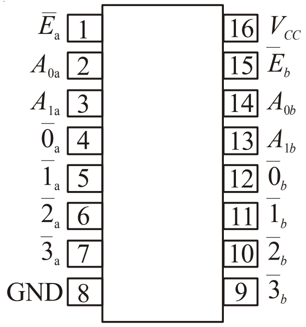
74123



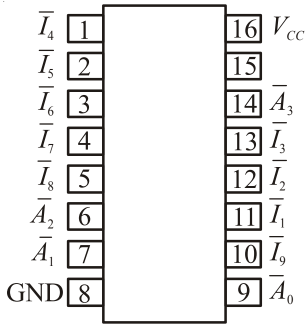
74132



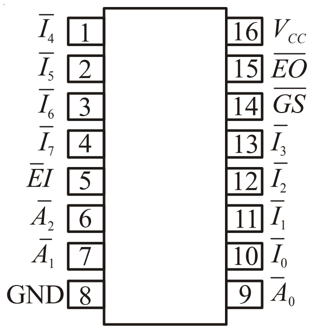
74138

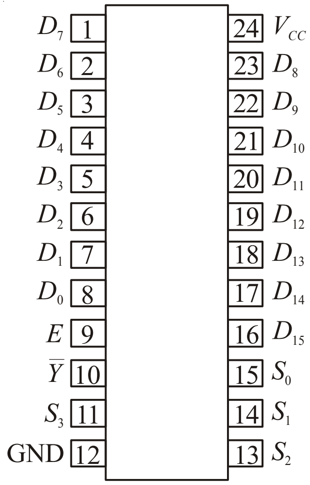


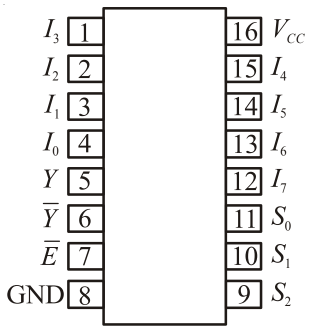
74139



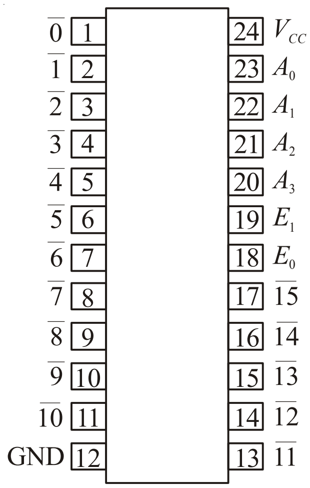
74147



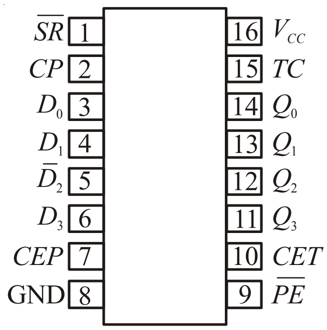




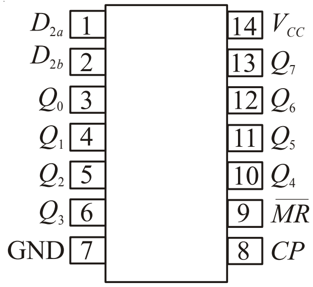
74148 74150 74151



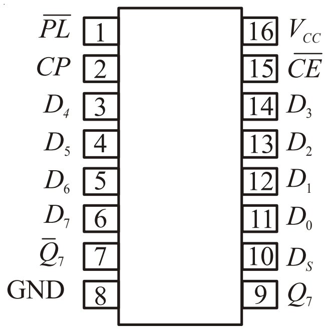
74154



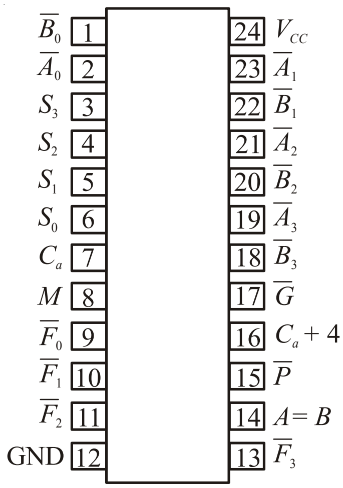
74163



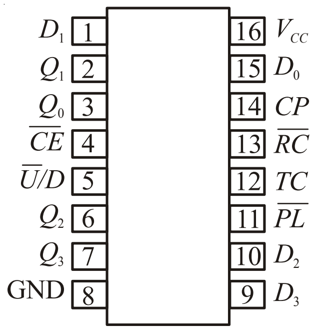
74164



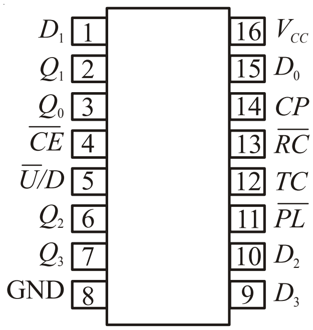
74165



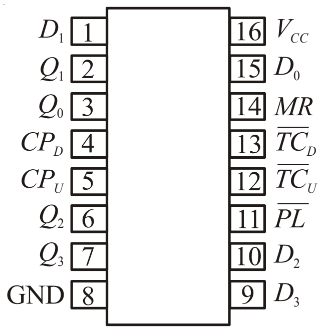
74181



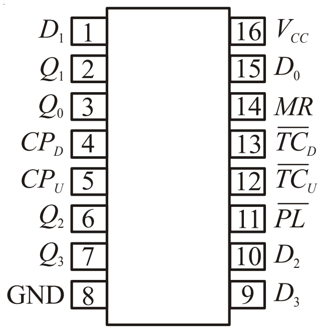
74190



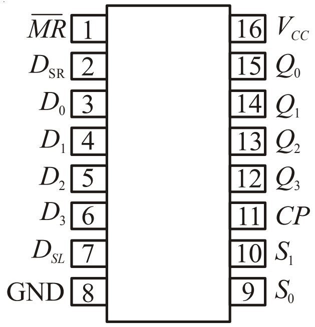
74191



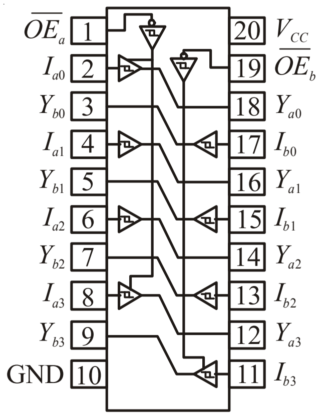
74192



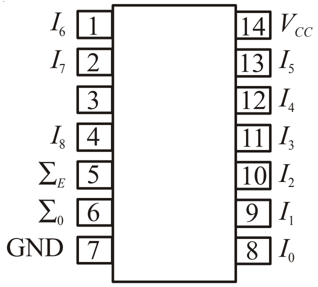
74193



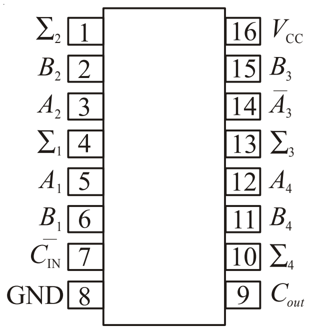
74194



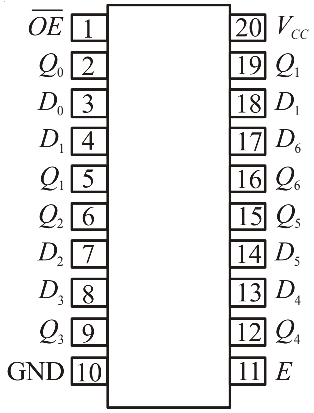
74244



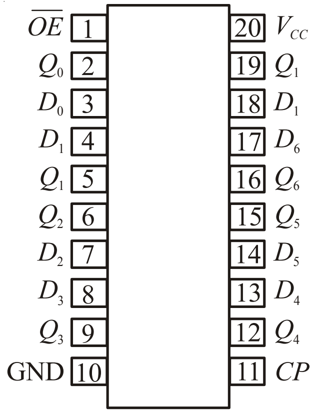
74280



74283



74373



74374

Appendix D

CONVERSIONS AMONG DIFFERENT NUMBER SYSTEMS

Decimal	*Binary*	*Octal*	*Hex*	*BCD (8421)*	*Decimal*	*Binary*	*Octal*	*Hex*	*BCD (8421)*
0	0	0	0	0	32	100000	40	20	110010
1	1	1	1	1	33	100001	41	21	110011
2	10	2	2	10	34	100010	42	22	110100
3	11	3	3	11	35	100011	43	23	110101
4	100	4	4	100	36	100100	44	24	110110
5	101	5	5	101	37	100101	45	25	110111
6	110	6	6	110	38	100110	46	26	111000
7	111	7	7	111	39	100111	47	27	111001
8	1000	10	8	1000	40	101000	50	28	1000000
9	1001	11	9	1001	41	101001	51	29	1000001
10	1010	12	A	10000	42	101010	52	2A	1000010
11	1011	13	B	10001	43	101011	53	2B	1000011
12	1100	14	C	10010	44	101100	54	2C	1000100
13	1101	15	D	10011	45	101101	55	2D	1000101
14	1110	16	E	10100	46	101110	56	2E	1000110
15	1111	17	F	10101	47	101111	57	2F	1000111
16	10000	20	10	10110	48	110000	60	30	1001000
17	10001	21	11	10111	49	110001	61	31	1001001
18	10010	22	12	11000	50	110010	62	32	1010000
19	10011	23	13	11001	51	110011	63	33	1010001
20	10100	24	14	100000	52	110100	64	34	1010010
21	10101	25	15	100001	53	110101	65	35	1010011
22	10110	26	16	100010	54	110110	66	36	1010100
23	10111	27	17	100011	55	110111	67	37	1010101
24	11000	30	18	100100	56	111000	70	38	1010110
25	11001	31	19	100101	57	111001	71	39	1010111
26	11010	32	1A	100110	58	111010	72	3A	1011000
27	11011	33	1B	100111	59	111011	73	3B	1011001
28	11100	34	1C	101000	60	111100	74	3C	1100000
29	11101	35	1D	101001	61	111101	75	3D	11000001
30	11110	36	1E	110000	62	111110	76	3E	1100010
31	11111	37	1F	110001	63	111111	77	3F	1100011

Decimal	*Binary*	*Octal*	*Hex*	*BCD (8421)*	*Decimal*	*Binary*	*Octal*	*Hex*	*BCD (8421)*
64	1000000	100	40	1100100	83	1010011	123	53	10000011
65	1000001	101	41	1100101	84	1010100	124	54	10000100
66	1000010	102	42	1100110	85	1010101	125	55	10000101
67	1000011	103	43	1100111	86	1010110	126	56	10000110
68	1000100	104	44	1101000	87	1010111	127	57	10000111
69	1000101	105	45	1101001	88	1011000	130	58	10001000
70	1000110	106	46	1110000	89	1011001	131	59	10001001
71	1000111	107	47	1110001	90	1011010	132	5A	10010000
72	1001000	110	48	1110010	91	1011011	133	5B	10010001
73	1001001	111	49	1110011	92	1011100	134	5C	10010010
74	1001010	112	4A	1110100	93	1011101	135	5D	10010011
75	1001011	113	4B	1110101	94	1011110	136	5E	10010100
76	1001100	114	4C	1110110	95	1011111	137	5F	10010101
77	1001101	115	4D	1110111	96	1100000	140	60	10010110
78	1001110	116	4E	1111000	97	1100001	141	61	10010111
79	1001111	117	4F	11111001	98	1100010	142	62	10011000
80	1010000	120	50	10000000	99	1100011	143	63	10011001
81	1010001	121	51	10000001	100	1100100	144	64	100000000
82	1010010	122	52	10000010	101	1100101	145	65	100000001

Appendix E

Table E$_1$ Standard Values of Commercially Available Resistors

Ohms					***Kilohms***		***Megohms***	
(Ω)					**(kΩ)**		**(MΩ)**	
0.10	**1.0**	**10**	**100**	**1000**	**10**	**100**	**1.0**	**10.0**
0.11	1.1	11	110	1100	11	110	1.1	11.0
0.12	**1.2**	**12**	**120**	**1200**	**12**	**120**	**1.2**	**12.0**
0.13	1.3	13	130	1300	13	130	1.3	13.0
0.15	**1.5**	**15**	**150**	**1500**	**15**	**150**	**1.5**	**15.0**
0.16	1.6	16	160	1600	16	160	1.6	16.0
0.18	**1.8**	**18**	**180**	**1800**	**18**	**180**	**1.8**	**18.0**
0.20	2.0	20	200	2000	20	200	2.0	20.0
0.22	**2.2**	**22**	**220**	**2200**	**22**	**220**	**2.2**	**22.0**
0.24	2.4	24	240	2400	24	240	2.4	
0.27	**2.7**	**27**	**270**	**2700**	**27**	**270**	**2.7**	
0.30	3.0	30	300	3000	30	300	3.0	
0.33	**3.3**	**33**	**330**	**3300**	**33**	**330**	**3.3**	
0.36	3.6	36	360	3600	36	360	3.6	
0.39	**3.9**	**39**	**390**	**3900**	**39**	**390**	**3.9**	
0.43	4.3	43	430	4300	43	430	4.3	
0.47	**4.7**	**47**	**470**	**4700**	**47**	**470**	**4.7**	
0.51	5.1	51	510	5100	51	510	5.1	
0.56	**5.6**	**56**	**560**	**5600**	**56**	**560**	**5.6**	
0.62	6.2	62	620	6200	62	620	6.2	
0.68	**6.8**	**68**	**680**	**6800**	**68**	**680**	**6.8**	
0.75	7.5	75	750	7500	75	750	7.5	
0.82	**8.2**	**82**	**820**	**8200**	**82**	**820**	**8.2**	
0.91	9.1	91	910	9100	91	910	9.1	

Table E$_2$ Typical Capacitor Component Values

pF				**μ*F***				
10	100	1000	10000	0.10	1.0	10	100	1000
12	120	1200						
15	150	1500	15000	0.15	1.5	18	180	1800
22	220	2200	22000	0.22	2.2	22	220	2200
27	270	2700						
33	330	3300	33000	0.33	3.3	33	330	3300
39	390	3900						
47	470	4700	47000	0.47	4.7	47	470	4700
56	560	5600						
68	680	6800	68000	0.68	6.8			
82	820	8200						