FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGERIA

CENTRE FOR OPEN DISTANCE AND e-LEARNING (CODeL)

INTRODUCTION TO DIGITAL DESIGN AND MICROPROCESSOR

(CPT 322)

COURSE DEVELOPMENT TEAM

CPT 322

INTRODUCTION TO DIGITAL DESIGN AND MICROPROCESSOR

Course Developer/Writers
Dr. S. A. ADEPOJU
Department of Computer Science
Federal University of Technology, Minna, Nigeria.

Programme Coordinator
Mrs O. A. Abisoye
Computer Science Department
Federal University of Technology, Minna, Nigeria.

Instructional Designers
Prof. Gambari, Amosa Isiaka
Mr. Falode, Oluwole Caleb
Centre for Open Distance and e-Learning,
Federal University of Technology, Minna, Nigeria.

Editor
Chinenye Priscilla Uzochukwu
Centre for Open Distance and e-Learning,
Federal University of Technology, Minna, Nigeria.

Director
Prof. J. O. Odigure
Centre for Open Distance and e-Learning,
Federal University of Technology, Minna, Nigeria

INTRODUCTION

CPT 322 Introduction to Digital Design and Microprocessor is a 3 credit unit course for students studying towards acquiring a Bachelor of Science in Computer Science and other related disciplines. The course is divided into 4 modules and 13 study units. It will first introduce the history of digital computers. Then number system with emphasis on binary numbers will be discussed. Thereafter, logic circuit, Boolean algebra, K-map, combinational circuit and sequential circuits will be discussed.

The course guide therefore gives you an overview of what the course; CPT 322 is all about, the textbooks and other materials to be referenced, what you expect to know in each unit, and how to work through the course material.

What you will learn in this Course

The overall aim of this course, CPT 322 is to introduce you to basic concepts of digital electronic and microprocessor in order to enable you to understand the basic elements of logic circuits used in building microprocessor used in today's digital computer

Course Aim

This course aims to introduce students to the basics, concepts and design of digital circuits and microprocessor. It is believed the knowledge will enable the reader understand the basic logic circuits used to design the processors in use today and hence know how to build a smaller and faster ones with minimal cost and stress

Course Objectives

It is important to note that each unit has specific objectives. Students should study them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. Youshould always look at the unit objectives after completing a unit. In this way, you can besure that you have done what is required of you by the end of the unit.

However, below are overall objectives of this course. On completing this course, you should be able to:

□now the history of digital computers.
Carry out binary arithmetic operations
Understand different codes used in computer
Describe logic circuits and gates, know their symbols and truth table
Simplify Boolean expressions
Know standard form and canonical expression
Know min term and max terms

Draw Karnaugth (K) map
Distinguish between combinational and sequential circuit.
Discuss the various types of combinational circuits and sequential circuit and their operations

Working through this Course

To complete this course, you are required to study all the units, the recommended text books, and other relevant materials. Each unit contains some self assessment exercises and tutor marked assignments, and at some point in this course, you are required to submit the tutor marked assignments. There is also a final examination at the end of this course. Stated below are the components of this course and what you have to do.

Course Materials

The major components of the course are:

1. Course Guide

2. Study Units

3. Text Books

4. Assignment File

5. Presentation Schedule

Study Units

There are 13 study units and 4 modules in this course. They are:

Module 1: Introduction To Computer & Number System

UNIT 1 History of Digital Computer

UNIT 2 Number system

UNIT 3 Binary Codes and Arithmetic

Module 2: Logic Gates

UNIT 1 Basic Logic

UNIT 2 Boolean Algebra

UNIT 3 KarnaughMaps

UNIT 4 Standard forms, Min term and Max term

Module 3: Combinational Logic

UNIT 1 Adder and Subtractor

UNIT 2 Multiplexer and demultiplexer

UNIT 3 Decoder, Encoder and Comparator

Module 4: Sequential Logic

UNIT 1Latches and Flip flop

UNIT 2 multi vibrator

UNIT 3 Shift Register and counter

Recommended Texts

These texts and especially the internet resource links will be of enormous benefit to you in learning this course:

Ronald J. T.& Neal S., (2001).WidmerDigital Systems: Principle and Applications (8thEd.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals, (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
http://www.computersciencelab.com/ComputerHistory/HistoryPt3.htm
http://drstienecker.com/tech-332/3-logic-circuits-boolean-algebra-and-truth-tables/
http://www.courses.ebe.uct.ac.za/eee317w/1.%20Basic%20Logic%20Design . pdf
http://www.circuitstoday.com/half-adder-and-full-adder
http://www.electronics-tutorials.ws/combination/comb_5.html
http://www.circuitstoday.com/half-adder-and-full-adder
http://www.ccse.kfupm.edu.sa/~amin/eCOE200/Lesson4_4.pdf
http://www.indiabix.com/digital-electronics/combinational-logic-circuits /116006
http://www.allaboutcircuits.com
https://maxwell.ict.griffith.edu.au/yg/teaching/.../dns_module3_p3.pd...
http://www.ce.rit.edu/studentresources/reference.../341/.../EECC341-08.pdf
http://www.techterms.com/definition/integratedcircuit

Assignment File

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are tutor marked assignments for this course.

Presentation Schedule

The presentation schedule included in this course guide provides you with important dates for completion of each tutor marked assignment. You should therefore endeavour to meet the deadlines.

Assessment

There are two aspects to the assessment of this course. First, there are tutor marked assignments; and second, the written examination. Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 40% of your total course mark.

At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score.

Tutor Marked Assignments (Tmas)

There are TMAs in this course. You need to submit all the TMAs. The best 10 will therefore be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

Final Examination and Grading

The final examination for CIT 322 will last for a period of 3 hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self assessment exercise and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

The following are practical strategies for working through this course

1. Read the course guide thoroughly

2. Organize a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all these information in one place such as a diary, a wall chart calendar or an organizer. Whatever method you choose, you should decide on and write in your own dates for working on each unit.

3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.

4. Turn to Unit 1 and read the introduction and the objectives for the unit.

5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further readings, on your desk at the same time.

6. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books

7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.

8. Review the objectives of each study unit to confirm that you have achieved them.

If you are not certain about any of the objectives, review the study material and consult your tutor.

9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.

10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.

11. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide).

Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary: contact your tutor if:

> You do not understand any part of the study units or the assigned readings.
> You have difficulty with the self test or exercise.
> You have questions or problems with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only opportunity to have face to face contact with your tutor and ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from the course tutorials, have some questions handy before attending them. You will learn a lot from participating actively in discussions.

GOODLUCK!

Module 1

Introduction

UNIT 1: History of Digital Computer

UNIT 2 : Number system

UNIT 3 :Binary Codes and Arithmetic

Unit 1

History of Digital Computer

Content

1.0 Introduction

Digital Computer has undergone many developmental stages before we have what is available today. The word ‘computer’ is an old word that has changed its meaning several times in the last few centuries. Originating from the Latin, by the mid-17th century it meant ‘someone who computes’. The American Heritage Dictionary gives its first computer definition as “a person who computes.” The computer remained associated with human activity until about the middle of the 20th century when it became applied to “a programmable electronic device that can store, retrieve, and process data” as Webster’s Dictionary defines it. Today, the word computer refers to computing devices, whether or not they are electronic, programmable, or capable of ‘storing and retrieving’ data.

This unit examines the chain of events that led to today’s digital computers. We’ll begin by looking at the computing equivalent of ancient history, including the first mechanical calculators and their huge, electromechanical offshoots that were created at the beginning of World War II. Next, you’ll examine the technology—electronics—that made today’s computers possible, beginning with what is generally regarded to be the first successful electronic computer, the ENIAC of the late 1940s. We’ll then examine the subsequent history of electronic digital computers, divided into different “generations” of distinctive—and improving—technology.

2.0 Learning Outcomes

At the end of this unit, you should be able to:

Define the term “electronics” and describe some early electronic devices that helped launch the computer industry.
Discuss the role that the stored-program concept played in launching the commercial computer industry.
List the generations of computer technology.
Identify the key innovations that characterize each generation.

3.1 Steps Toward Modern Computing

Today’s electronic computers are recent inventions, stemming from work that began during World War II. Yet the most basic idea of computing—the notion of representing data in a physical object of some kind, and getting a result by manipulating the object in some way—is very old. In fact, it may be as old as humanity itself. Throughout the ancient world, people used devices such as notched bones, knotted twine, and the abacus to represent data and perform various sorts of calculations..

First Steps: CalculatorDuring the sixteenth and seventeenth centuries, European mathematicians developed a series of calculators that used clockwork mechanisms and cranks . As the ancestors of today’s electromechanical adding machines, these devices weren’t computers in the modern sense. A **calculator** is a machine that can perform arithmetic functions with numbers, including addition, subtraction, multiplication, and division.

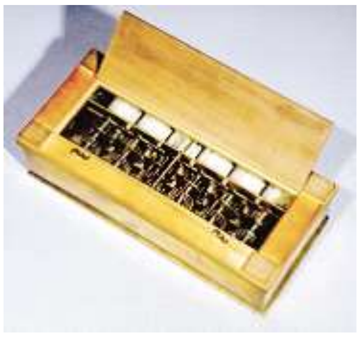


Figure1 :Pascal’s calculator (1642)

French mathematician and philosopher Blaise Pascal, the son of an accountant, invents an adding machine to relieve the tedium of adding up long columns of tax figures.

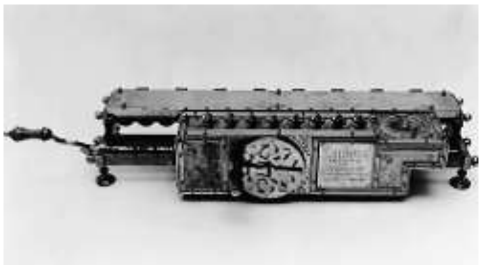


Figure 2:Leibniz’s calculator (1674)

German philosopher Gottfried Leibniz invents the first mechanical calculator capable of multiplication.

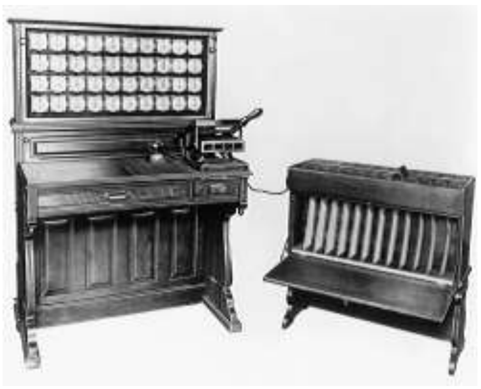


Figure 3: Babbage’s difference engine (1822)

English mathematician and scientist Charles Babbage designs a complex, clockwork calculator capable of solving equations and printing the results. Despite repeated attempts, Babbage was never able to get the device to work.

Self Assessment Questions

Identify the early types of Electronic calculators

Self Assessment Answers

Pascal Calculator

Babbage's difference Engine

Leibniz's calculator

3.2 The Technological Edge: Electronics

Today's computers are **automatic,** in that they can perform most tasks without the need for human intervention. They require a type of technology that was unimaginable in the nineteenth century. Nineteenth century inventor Charles Babbage came up with the first design for a recognizably-modern computer. It would have used a clockwork mechanism, but the technology of his day could not create the various gears needed with the precision that would have been required to get the device to work.

The technology that enables today's computer industry is called electronics. In brief, electronics is concerned with the behavior and effects of electrons as they pass through devices that can restrict their flow in various ways. The earliest electronic device, the vacuum tube, is a glass tube, emptied of air, in the flow of electrons that can be controlled in various ways. Created by Thomas Edison in the 1880s, vacuum tubes can be used for amplification, which is why they powered early radios and TVs, or switching, their role in computers. In fact, vacuum tubes powered all electronic devices (including stereo gear as well as computers) until the advent of solidstate devices. Also referred to as a semiconductor, a solid-state device acts like a vacuum tube, but it is a "sandwich" of differing materials that are combined to restrict or control the flow of electrical current in the desired way.

3.3. The Electronic NumericalIntegrator AndComputer (ENIAC)

With the advent of vacuum tubes, the technology finally existed to create the first truly modern computer—and the demands of warfare created both the funding and the motivation. In World War II, the American military needed a faster method to calculate shell missile trajectories. The military asked Dr. John Mauchly (1907–1980) at the University of Pennsylvania to develop a machine for this purpose. Mauchly worked with a graduate student, J. Presper Eckert (1919–1995), to build the device. Although commissioned by the military for use in the war, the ENIAC was not completed until 1946, after the war had ended .Although it was used mainly to solve challenging math problems, **ENIAC** was a true programmable digital computer rather than an electronic calculator.

One thousand times faster than any existing calculator, the ENIAC gripped the public's imagination after newspaper reports described it as an "Electronic Brain." The ENIAC took only 30 seconds to compute trajectories that would have required 40 hours of hand calculations.

ENIAC

3.4 The Stored-Program Concept

ENIAC had its share of problems. It was frustrating to use because it wouldn't run for more than a few minutes without blowing a tube, which caused the system to stop working. Worse, every time a new problem had to be solved, the staff had to enter the new instructions the hard way: by rewiring the entire machine. The solution was the stored program concept, an idea that occurred to just about everyone working with electronic computers after World War II.

With the *stored-program concept,*the computer program, as well as data, is stored in the computer's memory.One key advantage of this technique is that the computer can easily go back to a previous instruction and repeat it. Most of the interesting tasks that today's computers perform stem from repeating certain actions over and over. But the most important advantage is convenience. You don't have to rewire the computer to get it to do something different. Without the stored-program concept, computers would have remained tied to specific jobs, such as cranking out ballistics tables. All computers that have been sold commercially have used the stored program concept.

Self Assessment Questions

What is full Meaning of the ENIAC

What is the fundamental electronic device employed on the ENIAC

Self Assessment Answers

ENIAC stands for   Electronic Numerical Integrator And Computer

The fundamental Electronic device is the Vaccum Tube.

3.5 The Computer Generations

The computer has evolcved through many generations and it is being view differently by many authors. According to The Computational Science Education Project, US, the computer has evolved through the following stages from the mechanical era which have been described above:

First Generation Electronic Computers (1937-1953)

These devices used electronic switches, in the form of vacuum tubes, instead of electromechanical relays. The earliest attempt to build an electronic computer was by J. V. Atanasoff, a professor of physics and mathematics at Iowa State in 1937. Atanasoff set out to build a machine that would help his graduate students solve systems of partial differential equations. By 1941 he and graduate student Clifford Berry had succeeded in building a machine that could solve 29 simultaneous equations with 29 unknowns. However, the machine was not programmable, and was more of an electronic calculator.

A second early electronic machine was Colossus, designed by Alan Turing for the British military in 1943. The first general purpose programmable electronic computer was the Electronic Numerical Integrator and Computer (ENIAC), built by J. Presper Eckert and John V. Mauchly at the University of Pennsylvania. Research work began in 1943, funded by the Army Ordinance Department, which needed a way to compute ballistics during World War II. The machine was completed in 1945 and it was used extensively for calculations during the design of the hydrogen bomb. Eckert, Mauchly, and John von Neumann, a consultant to the ENIAC project, began work on a new machine before ENIAC was finished.

The main contribution of EDVAC, their new project, was the notion of a stored program. ENIAC was controlled by a set of external switches and dials; to change the program required physically altering the settings on these controls. EDVAC was able to run orders of magnitude faster than ENIAC and by storing instructions in the same medium as data, designers could concentrate on improving the internal structure of the machine without worrying about matching it to the speed of an external control. Eckert and Mauchly later designed what was arguably the first commercially successful computer, the UNIVAC; in 1952. Software technology during this period was very primitive.

Second Generation (1954-1962)

The second generation witnessed several important developments at all levels of computer system design, ranging from the technology used to build the basic circuits to the programming languages used to write scientific applications. Electronic switches in this era were based on discrete diode and transistor technology with a switching time of approximately 0.3 microseconds. The first machines to be built with this technology include TRADIC at Bell Laboratories in 1954 and TX-0 at MIT's Lincoln Laboratory. Index registers were designed for controlling loops and floating point units for calculations based on real numbers.

A number of high level programming languages were introduced and these include FORTRAN (1956), ALGOL (1958), and COBOL (1959). Important commercial machines of this era include the IBM 704 and its successors, the 709 and 7094. In the 1950s the first two supercomputers were designed specifically for numeric processing in scientific applications.

The transistor heralded the second generation of computers.

Early second-generation computers were frustrating to use because they could run onlyone job at a time. Users had to give their punched cards to computer operators, who would run their program and then give the results back to the user.

Self Assessment Questions

What were the punched cards used for on the Second Generation computers

The main electronic device on the second generation computer is the _________

Self assessment Answers

The Punched cards were used as storage devices

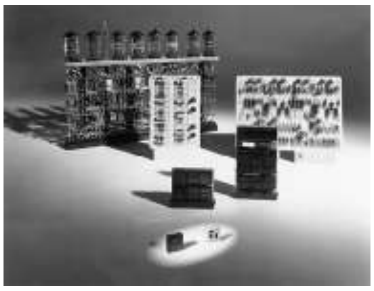
The main electronic device on the second generation computer is the Transistors

Third Generation (1963-1972)

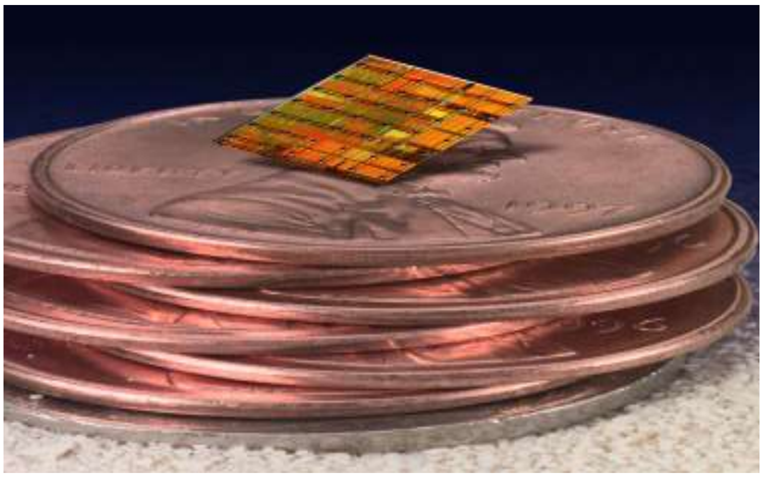
Technology changes in this generation include the use of integrated circuits, or ICs (semiconductor devices with several transistors built into one physical component), semiconductor memories, microprogramming as a technique for efficiently designing complex processors and the introduction of operating systems and time-sharing. The first ICs were based on small-scale integration (SSI) circuits, which had around 10 devices per circuit (or 'chip'), and evolved to the use of medium-scale integrated (MSI) circuits, which had up to 100 devices per chip. Multilayered printed circuits were developed and core memory was replaced by faster, solid state memories.

In 1964, Seymour Cray developed the CDC 6600, which was the first architecture to use functional parallelism. By using 10 separate functional units that could operate simultaneously and 32 independent memory banks, the CDC 6600 was able to attain a computation rate of one million floating point operations per second (Mflops). Five years later CDC released the 7600, also developed by Seymour Cray. The CDC 7600, with its pipelined functional units, is considered to be the first vector processor and was capable of executing at ten Mflops. The IBM 360/91, released during the same period, was roughly twice as fast as the CDC 660.

Early in this third generation, Cambridge University and the University of London cooperated in the development of CPL (Combined Programming Language, 1963). CPL was, according to its authors, an attempt to capture only the important features of the complicated and sophisticated ALGOL. However, like ALGOL, CPL was large with many features that were hard to learn. In an attempt at further simplification, Martin Richards of Cambridge developed a subset of CPL called BCPL (Basic Computer Programming Language, 1967). In 1970 Ken Thompson of Bell Labs developed yet another simplification of CPL called simply B, in connection with an early implementation of the UNIX operating system. comment):



Integrated chips are shown here with first-generation vacuum tubes and second generation transistors



An integrated circuit ("silicon chip") [photo courtesy of IBM]

Fourth Generation (1972-1984)

Large scale integration (LSI - 1000 devices per chip) and very large scale integration (VLSI - 100,000 devices per chip) were used in the construction of the fourth generation computers. Whole processors could now fit onto a single chip, and for simple systems the entire computer (processor, main memory, and I/O controllers) could fit on one chip. Gate delays dropped to about 1ns per gate. Core memories were replaced by semiconductor memories. Large main memories like CRAY 2 began to replace the older high speed vector processors, such as the CRAY 1, CRAY X-MP and CYBER

In 1972, Dennis Ritchie developed the C language from the design of the CPL and Thompson's B. Thompson and Ritchie then used C to write a version of UNIX for the DEC PDP-11. Other developments in software include very high level languages such as FP (functional programming) and Prolog (programming in logic).

IBM worked with Microsoft during the 1980s to start what we can really call PC (Personal Computer) life today. IBM PC was introduced in October 1981 and it worked with the operating system (software) called 'Microsoft Disk Operating System (MS DOS) 1.0. Development of MS DOS began in October 1980 when IBM began searching the market for an operating system for the then proposed IBM PC and major contributors were Bill Gates, Paul Allen and Tim Paterson. In 1983, the Microsoft Windows was announced and this has witnessed several improvements and revision over the last twenty years.

Self Assessment Questions

What electronic devices serve as backbone of the third and fourth generation computers.

Self assessment answers

The Large scale Integrated Circuit and the Microprocessors respectively are the backbone electronics for the third and fourth Generation computers

Fifth Generation (1984-1990)

This generation brought about the introduction of machines with hundreds of processors that could all be working on different parts of a single program. The scale of integration in semiconductors continued at a great pace and by 1990 it was possible to build chips with a million components - and semiconductor memories became standard on all computers. Computer networks and single-user workstations also became popular.

Parallel processing started in this generation. The Sequent Balance 8000 connected up to 20 processors to a single shared memory module though each processor had its own local cache. The machine was designed to compete with the DEC VAX-780 as a general purpose Unix system, with each processor working on a different user's job. However Sequent provided a library of subroutines that would allow programmers to write programs that would use more than one processor, and the machine was widely used to explore parallel algorithms and programming techniques.

*Please Insert Relevant Images /Graphics

The Intel iPSC-1, also known as 'the hypercube' connected each processor to its own memory and used a network interface to connect processors. This distributed memory architecture meant memory was no longer a problem and large systems with more processors (as many as 128) could be built. Also introduced was a machine, known as a data-parallel or SIMD where there were several thousand very simple processors which work under the direction of a single control unit. Both wide area network (WAN) and local area network (LAN) technology developed rapidly.

Sixth Generation (1990 -)

Most of the developments in computer systems since 1990 have not been fundamental changes but have been gradual improvements over established systems. This generation brought about gains in parallel computing in both the hardware and in improved understanding of how to develop algorithms to exploit parallel architectures.

Workstation technology continued to improve, with processor designs now using a combination of RISC, pipelining, and parallel processing. Wide area networks, network bandwidth and speed of operation and networking capabilities have kept developing tremendously. Personal computers (PCs) now operate with Gigabit per second processors, multi-Gigabyte disks, hundreds of Mbytes of RAM, colour printers, high-resolution graphic monitors, stereo sound cards and graphical user interfaces.

Thousands of software (operating systems and application software) are existing today and Microsoft Inc. has been a major contributor. Microsoft is said to be one of the biggest companies ever, and its chairman – Bill Gates has been rated as the richest man for several years.

Finally, this generation has brought about micro controller technology. Micro controllers are ’embedded’ inside some other devices (often consumer products) so that they can control the features or actions of the product. They work as small computers inside devices and now serve as essential components in most machines.

Self Assessment Questions

What is the common characteristic feature of the fifth and sixth generation computers

Self Assessment Answers

The common feature is the parallel computing capability

4.0 Conclusion

This unit has given a broad view of chronological order on the genesis of modern day digital computers. You also read abou the various developmental stages which digital computers have gone through. This is to enable us trace the history of the modern day digital computers,

5.0 Summary

You have learnt:

The technology that enables today’s computer industry is called electronics. Electronics is concerned with the behavior and effects of electrons as they passthrough devices that can restrict their flow in various ways. The vacuum tube was the earliest electronic device.

The first successful large-scale electronic digital computer, the ENIAC, laid the foundation for the modern computer industry.

The stored-program concept fostered the computer industry’s growth because it enabled customers to change the computer’s function easily by running a different program.

First-generation computers used vacuum tubes and had to be programmed in difficult-to-use machine languages.

Second-generation computers introduced transistors and high-level programming languages such as COBOL and FORTRAN.

Third-generation computers introduced integrated circuits, which cut costs and launched the minicomputer industry. Key innovations included timesharing, wide area networks, and local area networks.

Fourth-generation computers use microprocessors. Key innovations include personal computers, the graphical user interface, and the growth of massive computer networks.

An unparalleled public medium for communication and commerce, the Internet has created a massive public computer network of global proportions

As computers become more powerful and less expensive, the rise of global networking is making them more valuable. The combination of these two forces is driving major changes in every facet of our lives.

6.0 Tutor Marked Assignment

1.Explain why ENIAC is considered the first true programmable digital computer. What kinds of problems did it have?

2. Explain the stored-program concept. How did this concept radically affect the design of computers we use today?

3. What major hardware technology characterized each of the four generations of computers?

4. What are the differences between a command line interface and a user interface? Which one is easier to use and why?

5. How does a machine language differ from a high-level programming language?

7.0 References/Further Readings

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,

Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.

Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall

Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

http//www.en.wikipedia.org/wiki/Computer

http://www.computersciencelab.com/ComputerHistory/HistoryPt3.htm

http://www.computersciencelab.com/ComputerHistory/HistoryPt3.htm

http://www.ieeeghn.org/wiki/images/5/57/Onifade.pdf

http://www.cs.ncl.ac.uk/publications/articles/papers/398.pdf

Unit 2

Number Systems

Content

1.0 Introduction

Before the inception of *digital* computers, the only number system that was in common use is the *decimal* number system which has a total of 10 digits (0 to 9). However, signals in *digital* computers may represent a digit in some number system. It was also found that the binary number system is more reliable to use compared to the more familiar decimal system. The binary number system and digital codes are fundamental to computers and to digital electronics in general.

In this unit, the binary number system and its relationship to other number systems such as decimal, hexadecimal, and octal is presented. Arithmetic operations with binary numbers are covered to provide a basis for understanding how computers and many other types of digital systems work. Also, digital codes such as binary coded decimal (BCD), the Gray code, and the ASCII will be covered in the next unit..

2.0 Learning Outcomes

At the end of this unit, you should be able to:

Know what is meant by a weighted number system.
Know the asic features of weighted number systems.
Know and review commonly used number systems, e.g. decimal, binary, octal and hexadecimal.
Convert from decimal to binary and from binary to decimal
Convert between the binary and hexadecimal number systems
Convert between the binary and octal number systems
How to convert from one number system to another

3.0 Learning Contents

3.1 Weighted Number Systems:

A number D consists of *n* digits with each digit has a particular *position*.D = d_{n-1} d_{n-2}…….. d_2 d_2d_0

Every digit *position* is associated with a *fixed weight*. If the weight associated with the *i*th.position is w*i*, then the value of D is given by:

D = $d_{n-1}\, w_{n-1} + d_{n-2} w_{n-2} + \ldots + d_2\, w_2 + d_1 + d_0\, w_0$

Example of Weighted Number Systems:

• The Decimal number system is a weighted system.

• For Integer decimal numbers, the weight of the rightmost digit (*at position 0*) is 1, the weight of *position 1* digit is 10, that of *position 2* digit is 100, *position 3* is 1000, etc.

Thus,

w_0= 1, w_1= 10, w_2=100, w_3= 1000, etc.

Example Show how the value of the decimal number 9375 is estimated

Position	3	2	1	0
Number	9	3	7	5
Weight	1000	100	10	1
Value	9x1000	3x100	7x10	5x1
Value	9000 +300+70+5			

The Radix (Base)

For *digit position i*, most weighted number systems use weights (w*i*) that are *powers of some constant valu*e called the radix (r) or the base such that w*i*= r*i*.. A number system of radix *r*, typically has a set of *r* allowed digits ∈ {0, 1, …, (r-1)}. The leftmost digit has the highest weight Most Significant Digit (MSD). The rightmost digit has the lowest weight Least Significant Digit (LSD)

Self assessment questions

What Is The Weight Value Of 7 In 9786

Self assessment Answers

The weight value of 7 in 9786 is 100

Example Decimal Number System

1. Radix (Base) = *Ten*

2. Since $w_i = r^i$, then $w_0 = 10^0 = 1$, $w_1 = 10^1 = 10$, $w_2 = 10^2 = 100$, $w_3 = 10^3 = 1000$, etc.

3. Number of Allowed Digits is Ten = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Thus:

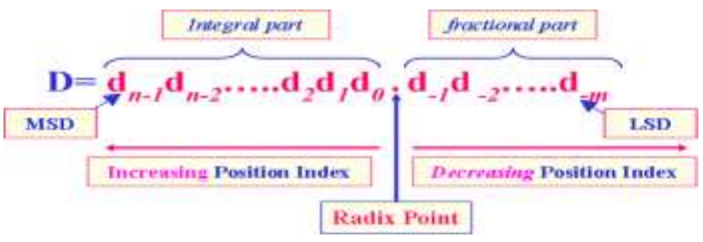
$9375 = 5x10^0 + 7x10^1 + 3x10^2 + 9x10^3$

$= 5x1 + 7x10 + 3x100 + 9x\ 1000$

Position	3	2	1	0
Weight	1000 $=10^3$	100 $=10^2$	10 $=10^1$	1 $=10^0$

The Radix Point

Consider a number system of radix r, a number D of *n integral* digits and *m fractional* digits is represented as shown


Integral part
fractional part
$D = d_{n-1}d_{n-2}\ldots d_2d_1d_0 . d_{-1}d_{-2}\ldots d_{-m}$
MSD
LSD
Increasing Position Index
Decreasing Position Index
Radix Point

Digits to the left of the radix point (*integral digits*) have positive position indices, while digits to the right of the radix point (*fractional digits*) have negative position indices.

Position *indices* of digits to the *left* of the *radix point* (*the integral part of D*) start with a 0 and are incremented as we move lefts ($d_{n-1}\, d_{n2} \ldots\ldots d_2\, d_1 d_0$.)

Position *indices* of digits to the *right* of the *radix point* (*the fractional part of D*) are *negative* starting with –1 and are decremented as we move rights ($d_{-1} d_{-2} \ldots\ldots d_{-m}$).

The *weight* associated with digit position *i* is given by $w_i = r_i$,*where i is the position index*

∀i= -m, -m+1, …, -2, -1, 0, 1, ……, n-1

The Value of D is Computed as :

Example Show how the value of the following decimal number is estimated

D = 5 2. 9 4 6

Number	5	2	.	9	4	6
Position	1	0	.	-1	-2	-3
Weight	10^1 = 10	10^0 = 1	.	10^{-1} = 0.1	10^{-2} = 0.01	10^{-3} = 0.001
Value	5x10	2x1	.	9x0.1	4x0.01	6x0.001
Value	50 +2+0.9+0.04+0.006					

$$D=5x10^1 + 2x10^0 + 9x10^{-1} + 4x\ 10^{-2} +6x10^{-3}$$

Notation

• Let $(D)_r$ denotes a number D expressed in a number system of radix r.

Note: In this notation, r will be expressed in decimal

Example:

$(29)_{10}$ Represents a decimal value of 29. The radix “10” here means ten.

$(100)_{16}$ is a Hexadecimal number since r = “16” here means sixteen. This number is equivalent to a decimal value of 162.

$(100)_2$ is a Binary number (radix =2, i.e. two) which is equivalent to a decimal value of 2^2 = 4.

Self assessment questions

Show how the decimal value of 100.01 is evaluated

Self assessment answers

$1x10^2+0x10^1 + 0x10^0 + 0x10^{-1} + 1x\ 10^{-2}$

3.2 Important Number Systems

The Decimal System

In the decimal number system each of the ten digits, 0 through 9.represents a certain quantity. The ten symbols (digits) do not limit you to expressing only ten different quantities because you use the various digits in appropriate positions within a number to indicate the magnitude of the quantity. You can express quantities up through nine before running out of digits; if you wish to express a quantity greater than nine, you use two or more digits, and the position of each digit within the number tells you the magnitude it represents. So,

r = 10 (ten Radix is not a Power of 2)

Ten Possible Digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

The Binary System

The binary number system is another way to represent quantities. It is less complicated than the decimal system because it has only two digits. The decimal system with its ten digits is a base-ten system; the binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and O. The position of a 1 or 0 in a binary number indicates its weight. or value within the number, just as the position of a decimal digit determines the value of that digit. The weights in a binary number are based on powers of two. So,

r = 2

Two Allowed Digits {0, 1}

A **B**inary Dig**IT** is referred to as Bit

The leftmost bit has the highest weight i.e. Most Significant Bit (MSB)

The rightmost bit has the lowest weight i.e. Least Significant Bit (LSB)

Conversion from Binary to decimal

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

Examples

Find the decimal value of the two Binary numbers (101)2 and (1.101)2

(1 0 1)2 = $1x2^3 + 0x2^1 + 1x2^0$

= 1x4 + 0x2 + 1x1

= $(5)_{10}$

(1. 1 0 1)2 = $1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

= 1 + 0.5 + 0.25 + 0.125

= $(1.875)_{10}$

Conversion from decimal to base 2 (Binary).

Example: Convert $(53)_{10}$ to base 2 (binary)(?)2

Soln

Division Step	Quotient	Remainder
53 ÷ 2	Q0 =26	1 = a0 ↑
26 ÷ 2	Q1 =13	0 = a1
13 ÷ 2	Q2 =6	1 = a2

$6 \div 2$	Q3 =3	0 = a3s
$3 \div 2$	Q4 =1	1 = a4
$1 \div 2$	0	1 = a5

Thus $(53)_{10}=(110101)_2$

Or by using sum of weigth method

$$53= 32+16+4+1=2^5 + 2^4 + 2^2 +2^0 =110101_2$$

To convert 546_{10} to binary

$546 \div 2$	Q0=273	R=0
$273 \div 2$	Q1=136	R=1
$136 \div 2$	Q2=68	R=0
$68 \div 2$	Q3=34	R=0
$34 \div 2$	Q4=17	R=0
$17 \div 2$	Q5=8	R=1
$8 \div 2$	Q6=4	R=0
$4 \div 2$	Q7=2	R=0
$2 \div 2$	Q8=1	R=0
$1 \div 2$	0	R=1

Thus $546_{10} = 1000100010_2$

To convert Fraction:

Convert $(0.731)_{10}$ to base 2

soln

0.731*2=**1**.462

0.462*2=**0**.924

0.924*2=**1**.848

0.848*2=**1**.696

0.696*2=**1**.392

0.392*2=**0**.784

0.784*2=**1**.568

(0.731) $_{10}$ = (.1011101)2

For a number that has both integral and fractional parts, conversion is done separately for both parts, and then the result is put together with a system point in between both parts.

E.gconvert 5. 275_{10} to base 2

For the integral part.

5_{10} is 101_2 from example above

Then the fraction part is computed as follows

.275 *2= 0.550

.550*2= 1.100

.100*2= 0.200

.200*2= 0.400

.400*2= 0.800

.800*2= 1.600

.600*2= 1.200

$5.275_{10} = 101.0100011_2$

Convert the following base 10 numbers into their binary equivalent

50 , 100, 10.2, 0.234

$50 = 110010_2$

$100 = 1100100_2$

10.2 =

0.234 =

3.3 Octal System

The octal number system provides a convenient way to express binary numbers and codes. However, it is used less frequently than hexadecimal in conjunction with computers and microprocessors to express binary quantities for input and output purposes. Here we have,

r = 8 (Eight = 2^3)

Eight Allowed Digits {0, 1, 2, 3, 4, 5, 6, 7}

Conversion from octal to decimal

Examples

Find the decimal value of the two Octal numbers $(375)_8$and $(2.746)_8$

$(375)_8 = 3x8^2 + 7x8^1 + 5x8^0$

$= 3x64 + 7x8 + 5x1$

$= (253)_{10}$

$(2.746)_8 = 2x8^0 + 7x8^{-1} + 4x8^{-2} + 6x8^{-3}$

$=2 + 7/8+4/64 + 6/512$

$= (2.94921875)_{10}$

Binary To Octal Conversion

Group of 3 binary bits	Octal equivalence
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Example :

Convert (1110010101.1011011)2 into Octal.

We first partition the Binary number into groups of 3 bits

001__110__010__101_._101__101__100

$=1625.554_8$

Convert 11110010100010100 into octal

Group into group of 3 bits

011 /110 /010 /100 /010/100

$=362424_8$

3.4 Hexadecimal System:

The hexadecimal number system has sixteen characters; it is used primarily as a compact way of displaying or writing 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 1 s and 0s, it is necessary to use these digits when you program in "machine language." Imagine writing a sixteen bit instruction for a microprocessor system in 1s and 0s. It is much more efficient to use hexadecimal or octal; octal numbers are covered in the previous section. Hexadecimal is widely used in computer and microprocessor applications. So,

r = 16 (Sixteen = 2^4)

Sixteen Allowed Digits {0-to-9 and A, B, C, D, E, F} Where: A = ten, B = Eleven, C = Twelve,D = Thirteen, E = Fourteen & F = Fifteen.

• Q: Why is the digit following 9 assigned the character A and not “10”?

• A: What we need is a *single* digit whose value is *ten*, but “10” is actually

two digits not *one*. Thus, in Hexadecimal system the 2-digit number $(10)_{16}$ actually represents a value of *sixteen* not *ten* {(10)16 = $0x16^{1}+1x16^{0}.=(16)_{10}$}.

Conversion from Hexadecimal to Decimal

One way to convert a hexadecimal number to its decimal equivalent is to multiply the decimal value of each hexadecimal digit by its weight and then take the sum of these products. The weights of a hexadecimal number are increasing powers of 16 (from right to left). Anotherway is to convert first to binary and then to decimal

Example

Find the decimal value of the two Hexadecimal numbers $(9E1)_{16}$ and$(3B.C\,)_{16}$

$(9E1)_{16}$ or $9E1_{hex}$ = $9x16^2 + Ex16^1 + 1x16^0$

= 9x256 x 14x16 x 1x1

= $(\,2529\,)_{10}$

$(3B.C\,)_{16}$= $3x16^1 + Bx16^0 + Cx16^{-1}$

= $3x16^1 + 11x16^0 + 12x16^{-1}$

= $(59.75\,)_{10}$

Conversion from Binary To Hexadecimal

Converting a binary number to hexadecimal is a straightforward procedure. Simply break the binary number into 4-bit groups, starting at the right-most bit and replaces each 4-bit group with the equivalent hexadecimal symbol.

Group of 4 bits equivalence	Hexadecimal
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	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Example :

Convert (1110010101.1011011)2 into Hexadecimal.

First group into 4bits

0011__1001__0101_._1011__0110

$=(395.B6)_{16}$

Convert 1100101001010111 in binary to hexadecimal

Break into four 4bits

=1100/1010/0101/0111

=CA57$_{16}$

To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits.

CF8E $_{I6}$= C-1100

F-1111

8- 1000

E-1110

= 1100111110001110$_2$

Conversion from Octal to Hexadecimal

Example: Convert 356$_8$ to hexadecimal

Soln

356$_8$ =11101110$_2$

= 1110/1110

= EE$_{16}$

Question. What is the result of adding 1 to the largest digit of some number system??

Answer.

For the decimal number system, $(1)_{10} + (9)_{10} = (10)_{10}$

For the octal number system, $(1)_8 + (7)_8 = (10)_8 = (8)_{10}$

For the hex number system, $(1)_{16} + (F)_{16} = (10)_{16} = (16)_{10}$

For the binary number system, $(1)_{2} + (1)_{2} = (10)_{2} = (2)_{10}$

Conclusion. Adding 1 to the largest digit in any number system always has a result of (10) in that number system.

Question. What is the largest value representable in 3-integral digits?

Answer. The largest value results when all 3 positions are filled with the largest digit in the number system.

For the decimal system, it is (999)10

For the octal system, it is (777)8

For the hex system, it is (FFF)16

For the binary system, it is (111)2

Clarification (c)

Q. What is the result of adding 1 to the largest 3-digit number?

?

A.

For the decimal system, (1)10 + (999)10 = (1000)10 = (103)10

For the octal system, (1)8 + (777)8 = (1000)8 = (83)10

For the hex system, (1)16 + (FFF)16 = (1000)16 = (163)16

For the binary system, (1)2 + (111)2 = (1000)2 = (23)10

In general, for a number system of radix r, adding 1 to the largest *n*-digit

number = r^{n}

Accordingly, the value of largest *n*-digit number = r^{n}-1

Self Assessment questions

Convert A4C in Hex to Decimal

Self assessment Answers

A4C= 2636

Conclusions.

1. In any number system of radix *r*, the result of adding 1 to the *largest n-digit* number equals r ^{n}

2. Thus, the value of the *largest n-digit* number is equal to r^{n} *-1*

3. Thus, *n digits* can represent *r n* different values (digit combinations) starting from a 0 value up to the largest value of r ^{n} *-1*

The table below gives the comparison between decimal, binary, octal and hexadecimal

Table 1:Comparison Table

Base, b	Byte (8-bits)	Word (16-bits)
Decimal	0 to 255_{10}	0 to $65{,}535_{10}$
Binary	0000 0000 to $1111\ 1111_2$	0000 0000 0000 0000 to $1111\ 1111\ 1111\ 1111_2$
Hexadecimal	00 to FF_{16}	0000 to $FFFF_{16}$
Octal	000 to 377_8	000 000 to $177\ 777_8$

Self Assessment Questions

1. The octal equivalent of 247_{10}

2. The hexadecimal number for 95.5_{10} is

3. Convert $(10001011.011)_2$ to decimal

4. Convert 567_8 to hexadecimal

5.The binary equivalent of FA_{16} is

Self Assessment answers

4.0 Conclusion

In this unit various number system are considered namely decimal, binary, hexadecimal and octal. How to convert from one number system to another were also discussed.

5.0 Summary

You have learnt:

A binary number is a weighted number in which the weight of each whole number digit is a positive power of two and the weight of each fractional digit is a negative power of two.
The whole number weights increase from right to left-from least significant digit to most significant.
. A binary number can be converted to a decimal number by summing the decimal values of the weights of all the Is in the binary number.
.A decimal whole number can be converted to binary by using the sum-of-weights or the repeated division-by-2 method.

A decimal fraction can be converted to binary by using the sum-of-weights or the repeated multiplication-by-2 method.
The hexadecimal number system consists of 16 digits and characters, 0 through 9 followed by A through F.
One hexadecimal digit represents a 4-bit binary number, and its primary usefulness is in simplifying bit patterns and making them easier to read.
A decimal number can be converted to hexadecimal by the repeated division-by-16 method.
The octal number system consists of eight digits, 0 through 7.
A decimal number can be converted to octal by using the repeated division-by-8 method.
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.

6.0 Tutor Marked Assignment

Convert each binary number to decimal, octal and hexadecimal:

(a) 11001111 (d) 1111000.101 (g) 10110101010 (e) 101I100.10101 (h) 111111 1.1 1111

Convert the following from Hexadecimal to binary ,octal and decimal
ADE$_{16}$ (b) FFFF$_{16}$ (c) 52C$_{16}$
Convert the following from decimal to binary, octal and hexadecimal

(a)245 (b)75 (c) 120

7.0 References/Further Reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
http://www.swarthmore.edu/NatSci/.../**BinaryMath**/**BinaryMath**.html
http://www.pages.cpsc.ucalgary.ca/~jacob/Courses/.../04-BitsAnd**Arithmetic**.pdf
http://www.allaboutcircuits.com
http://www.l3d.cs.colorado.edu/courses/CSCI1200-96/**binary**.html

Unit 3

Binary Codes and Arithmetic

Content

1.0 Introduction

Internally, digital computers operate on binary numbers. When interfacing to humans, digital processors, e.g. pocket calculators, communication is decimal based. Input is done in decimal then converted to binary for internal processing. For output, the result has to be converted from its internal binary representation to a decimal form. To be handled by digital processors, the decimal input (output) must be coded in binary in a digit by digit manner.

2.0 Learning Outcomes

At the end of this unit, you should be able to:

Explain several binary codes includingBinary Coded Decimal (BCD), Error detection codes, and Character codes
Differentiate between coding and binary conversion.
Calculate the 2's complement of a given n-bit binary number.
Explain the differences between signed-magnitude, signed-1's complement and signed-2's complement representations of negative binary numbers.
Perform subtraction problems in base 2 using the 2's complement method.
Express numbers in binary-coded decimal (BCD),
Perform integer binary arithmetic that is addition and subtraction
Describe and use two's complement and sign and magnitude to represent negative integers

3.0 Learning Contents

3.1 Binary Codes for Decimal Digits

There is a variety ofdecimal binary codesand they are shown in table 3.1

BINARY CODED DECIMAL (BCD)

One commonly used code is the *Binary Coded Decimal* (**BCD**) code which corresponds tothe first 10 binary representations of the decimal digits 0-9. Binary Coded Decimal is one of the early memory encodings. Rather than converting the entire denary value into its pure binary form, it converts each digit, separately, into its 4-bit binary equivalent. The table below shows the 4-bit BCD equivalents of the ten denary digits:

The BCD code requires 4 bits to represent the 10 decimal digits. Since 4 bits may have up to 16 different binary combinations, a total of 6 combinations will be unused. The position weights of the BCD code are 8, 4, 2, 1.

Other codes (shown in the table) use position weights of 8, 4, -2, -1 and 2, 4, 2, 1.

What does the BCD stand for ?

BCD stands for Binary Coded Decimal

An example of a non-weighted code is the *excess-3 code* where digit codes is obtained from their binary equivalent after adding 3. Thus the code of a decimal 0 is 0011, that of 6 is1001, etc.

Table 3.1 decimal codes

Decimal Digit	BCD 8 4 2 1	8 4 -2 -1	2 4 2 1	Excess-3
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 1 1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 1 0	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 0 1	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 1 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 0 1 0	1 1 0 0	1 0 0 1
7	0 1 1 1	1 0 0 1	1 1 0 1	1 0 1 0
8	1 0 0 0	1 0 0 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 1 1	1 1 0 0
U	1 0 1 0	0 0 0 1	0 1 0 1	0 0 0 0
N	1 0 1 1	0 0 1 0	0 1 1 0	0 0 0 1
U	1 1 0 0	0 0 1 1	0 1 1 1	0 0 1 0
S	1 1 0 1	1 1 0 0	1 0 0 0	1 1 0 1
E	1 1 1 0	1 1 0 1	1 0 0 1	1 1 1 0
D	1 1 1 1	1 1 1 0	1 0 1 0	1 1 1 1

Number Conversion VersusCoding

Converting a decimal number into binary is done by repeated division (multiplication) by 2for integers (fractions) (see lesson 2).Coding a decimal number into its BCD code is done by replacing each decimal digit of thenumber by its equivalent 4 bit BCD code.

Example Converting **(13)10** into binary, we get **1101**, coding the same number into BCD, weobtain **00010011**.

Example: convert 011010000011 from BCD to decimal

Soln

Group in four bits 0110 1000 0011= 683

Convert **(95)10** into its binary equivalent value and give its BCD code as well.

{(1011111)2, and 10010101}

3.2 Compliment Representation

Positive numbers (+N) are represented in ***exactly*** the same way as in signed magnitude system

Negative numbers (-N) are represented by the *complement* of N (**N'**)

One's and Two's Compliment Notation:

Two's compliment is a method of representing negative numbers in binary, whereby the most significant bit maintains its magnitude, but is made negative. In order to subtract one number from another, we need some means of representing negative numbers in binary notation. The "two's complement" convention is almost universally used for this purpose.

The '1's complement' of a binary number 0110 is just 1001 (obtained by inverting or negating each bit in the number). i.e. by bitwise complementing of each bit, i.e. each 1 is replaced by a 0 and each 0 is replaced by a 1.

The '2's complement' is formed by taking the '1's complement' and adding 1. Thus, the '2's complement' of 0110 is 1001 + 0001 = 1010. To show that this is the negative of the initial number (0110) simply add them 0110 + 1010 = 0000 plus a carry bit, which is ignored with an 8 bit number

Negative numbers will always start with a '1' and positives will start will a '0';• the range of integers that can be represented using one byte is from – 128 up to + 127.

e.g. decimal 18 is represented in a byte as 0 0 0 1 0 0 1 0 while -18 is 1 0 0 1 0 0 1 0

Likewise 1 0 0 0 0 0 0 0 = – 128

0 1 1 1 1 1 1 1 = + 127

0 0 0 0 0 1 1 0 the number

1 1 1 1 1 0 0 1 its 1's complement

1 1 1 1 1 0 1 0 its 2's complement

0 0 0 0 0 1 1 0 number to be added

1 0 0 0 0 0 0 0 0 the sum (difference)

as before, we neglect the carry bit. Notice that the most significant bit of the negative number is a 1 while that of the positive number is a 0. This is a necessary feature of two's complement arithmetic. This makes it easy to test whether a number is positive or negative, you simply check its most significant bit.

Find the 1's and the 2's complement of 110101010

001010101- 1's Complement

001010110- 2's Complement

Converting A Negative Denary Integer Into Two's Complement

Taking the denary integer – 52 as an example

First convert to unsigned binary

+52= 0 0 1 1 0 1 0 0

Convert to 1's complement

1 1 0 0 1 0 1 1

Then finally to 2’s complement

1 1 0 0 1 1 0 0

Converting a Two’s Complement Number Into Denary

This is the same as converting any binary number into denary, as long as you remember that the most significant bit is negative. For example the ‘signed’ binary number 1 1 0 1 0 1 0 1 is converted as follows:

– 128	64	32	16	8	4	2	1
1	1	0	1	0	1	0	1

= – 128 + 64 + 16 + 4 + 1

= – 43

Sign and Magnitude

The alternative to using two’s complement to represent negative numbers is to use the ‘sign and magnitude’ method – here, the most significant bit is used as a sign bit without a numerical value.

Convert 1 1 0 0 1 1 0 0 in 2’s complement to decimal

–	64	32	16	8	4	2	1
1	1	0	0	1	1	0	0

= – (64 + 8 + 4)

= – 76

Notes:s

The range of integers that can be represented using one byte is from – 127 up to + 127. although the sign and magnitude method is easier for humans it is much harder to use for computers performing arithmetic.

Self Assessment Questions

a). Assuming a single byte is used, convert the following numbers into two's compliment binary:(a) – 5 (b) – 10 (c) – 20
(b). What is the denary value of 1010 1011 if the binary codes represent: (a) a two's compliment number (b) a sign and magnitude number

Addition And Subtraction of Numbers Using1's And 2's Compliment

Addition: Computers will only ever add two numbers at a time – if three numbers need to be added, a computer will add the first two and then add the third number will be added to the result.

Subtraction: To perform subtraction, the number to be subtracted is converted into its two's compliment negative and then added

Perform 12 + 25, 25-12, 12-25

12= 0 0 0 0 1 1 0 0

25 = 0 0 0 1 1 0 0 1

+ = 0 0 1 0 0 1 0 1

= 37

– 12 = 1 1 1 1 0 1 0 0 (2's compliment of 12)

25 = 0 0 0 1 1 0 0 1

+ = 10 0 0 0 1 1 0 1

= 13

Discard the carry 1

12= 0 0 0 0 1 1 0 0

-25= 1 1 1 0 0 1 1 1

+= 1 1 1 1 0 0 1 1

The answer is in two's compliment which is equal to -13

NB 2's compliment of 00001101 is 11110011. So the answer is -13

Try these questions

Computer the following using 2's compliments arithmetic

34-78
34+78
100-34
34-100

3.3 Error-Detection Codes

Binary information may be transmitted through some communication medium, e.g. usingwires or wireless media. A corrupted bit will have its value changed from 0 to 1 or vice versa. To be able to detect errors at the receiver end, the sender sends an extra bit (*parity bit*) with the original binary message.

A *parity bit* is an extra bit included with the *n-bit binary message* to make the total numberof 1's in this message (*including the parity bit*) either odd or even. If the *parity bit* makes the total number of 1's an odd (even) number, it is called odd (even) parity.

The table shows the ***required*** odd (***even***) ***parity*** for a 3-bit message. At the receiver end, an error is detected if the message does not match have the proper parity (odd/even). Parity bits can detect the occurrence 1, 3, 5 or any odd number of errors in the transmitted message.

Table 2

Three-Bit Message X Y Z	Odd Parity Bit P	Even Parity Bit P
0 0 0	1	0
0 0 1	0	1
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

No error is detectable if the transmitted message has 2 bits in error since the total number of 1's will remain even (or odd) as in the original message.

In general, a transmitted message with even number of errors cannot be detected by the parity bit.

Self assessment Question

What do you understand by parity bit

Self Assessment Answers

A *parity bit* is an extra bit included with the *n-bit binary message* to make the total number of 1's in this message either odd or even.

Gray Code

The Gray code is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that it exhibits only a single bit change from one code word to the next in sequence..The Gray code consist of 16 4-bit code words to represent the decimal Numbers 0 to 15.It is useful in applications like analog to digital conversion. This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence For Gray code, successive code words differ by only one bit from one to the next as shownin the table and further illustrated in the table below.

Gray Code	Decimal Equivalent
0 0 0 0	0
0 0 0 1	1
0 0 1 1	2
0 0 1 0	3
0 1 1 0	4
0 1 1 1	5
0 1 0 1	6
0 1 0 0	7
1 1 0 0	8
1 1 0 1	9
1 1 1 1	10
1 1 1 0	11
1 0 1 0	12
1 0 1 1	13
1 0 0 1	14
1 0 0 0	15

What is the characteristic feature of the Gray Code

The important feature of the Gray code is that it exhibits only a single bit change from one code word to the next in sequence

Binary-to-Gray Code Conversion

Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.

2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

For example, the conversion of the binary number 10110 to Gray code is as follows:

1+0+l+l+0 binary

1 1 1 0 1   gray

the gray code is 11101

Gray-to-Binary Conversion

To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.

2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code word 11011 to binary is as follows:

1 then 1+1= 0 then 0 + 0= 0 then 0+1=1 then 1 +0=0

=10010

Self Assessment Question

(a) Convert the binary number 11000110 to Gray code.
(b) Convert the Gray code 10101111 to binary.

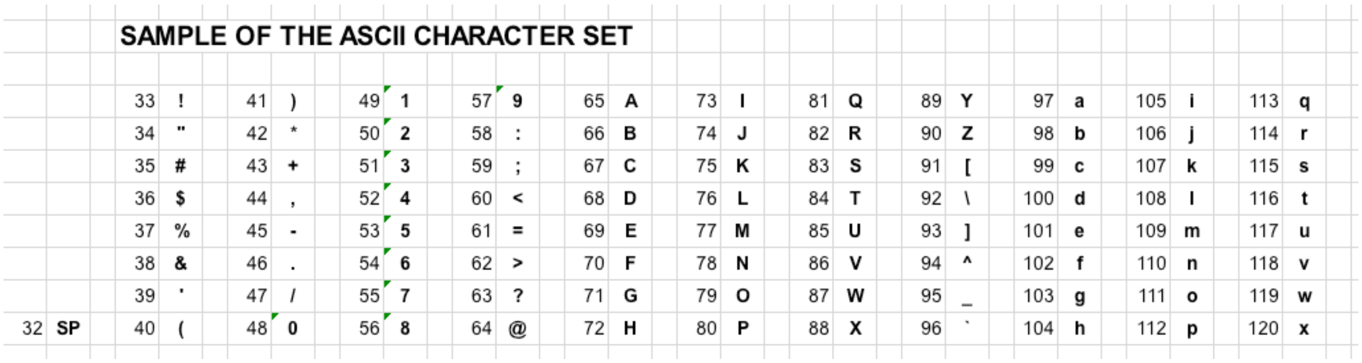
Self Assessment Answers

Character Codes

ASCII Character Code

ASCII code is a 7-bit code.*American Standard Code for Information Interchange* (*ASCII*) is used for character encoding by most Windows™ PCs. ASCII can be used to translate alphanumeric characters into a 7-bit binary code that represents all the characters available from the keyboard including punctuation and some special symbols such as '@', # and $:

It is used in communicating information between a computer and its peripherals or other computers.



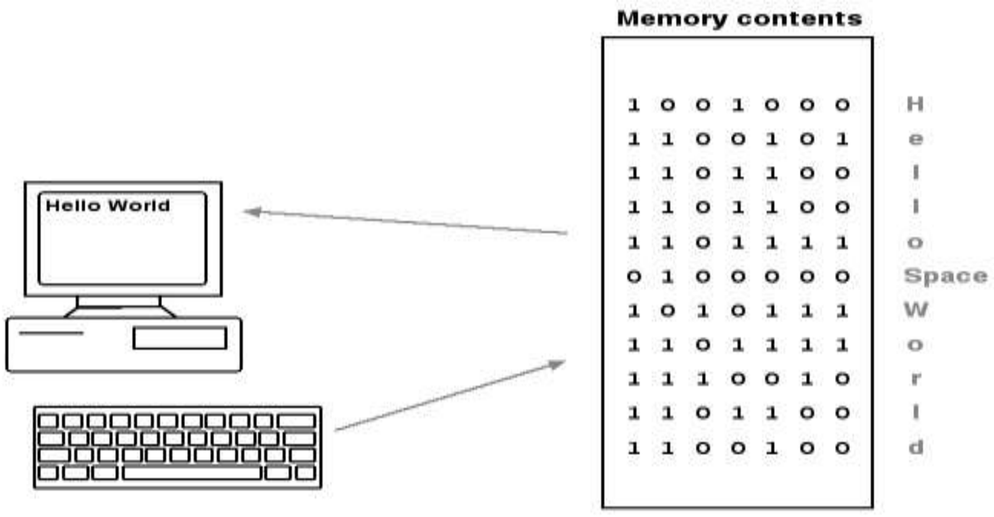
SAMPLE OF THE ASCII CHARACTER SET

		33	!	41	)	49	1	57	9	65	A	73	I	81	Q	89	Y	97	a	105	i	113	q
		34	"	42	*	50	2	58	:	66	B	74	J	82	R	90	Z	98	b	106	j	114	r
		35	#	43	+	51	3	59	;	67	C	75	K	83	S	91	[	99	c	107	k	115	s
		36	$	44	,	52	4	60	<	68	D	76	L	84	T	92	\	100	d	108	l	116	t
		37	%	45	-	53	5	61	=	69	E	77	M	85	U	93	]	101	e	109	m	117	u
		38	&	46	.	54	6	62	>	70	F	78	N	86	V	94	^	102	f	110	n	118	v
		39	'	47	/	55	7	63	?	71	G	79	O	87	W	95	_	103	g	111	o	119	w
32	SP	40	(	48	0	56	8	64	@	72	H	80	P	88	X	96	`	104	h	112	p	120	x

A development of ASCII, known as *Extended ASCII*, uses an 8-bit code that also defines codes for additional characters, including some graphical ones. Note that using an 8-bit code means the maximum number of characters that can be represented is 256.

How Character Encoding Works

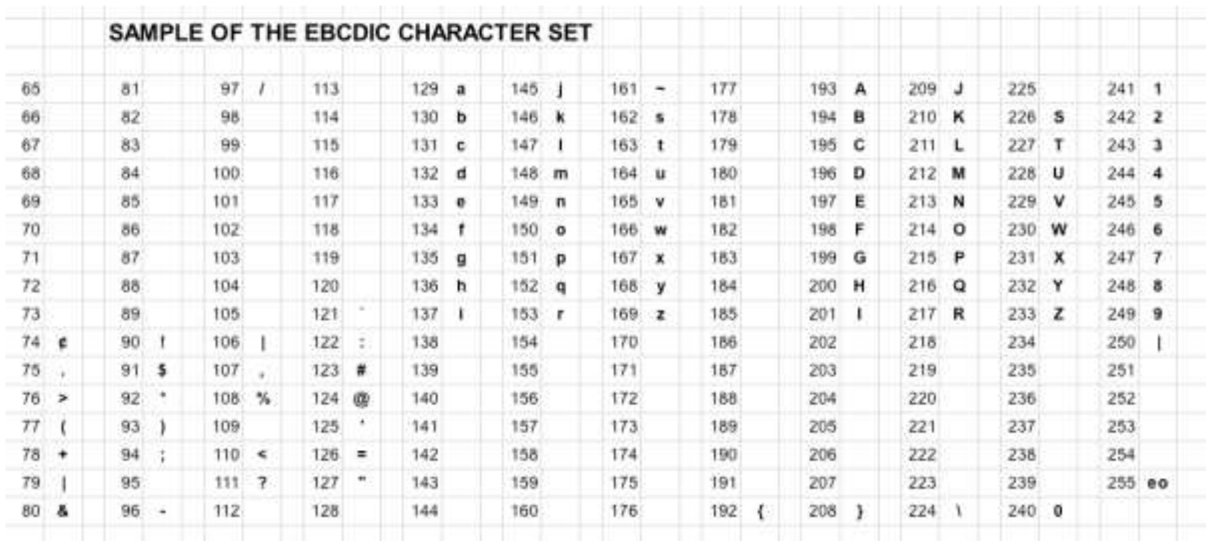
The diagram below shows how the message "Hello World" is stored in the memory of a computer using the ASCII codes:



The message is typed at the keyboard. Electronics in the keyboard convert the typed characters intoASCII binary codes that are sent from the keyboard along a cable to the computer. The computerstores these codes in its internal memory. The computer also provides a visual display of thecharacters as they are typed. To be able to do this, electronics inside the computer convert thestored binary codes back into their character equivalents.

Extended Binary Coded Decimal Interchange Code (EBCDIC)

Extended Binary Coded Decimal Interchange Code (*EBCDIC*) was developed by IBM for use in their mainframe systems. It has the same limitation as ASCII in that its 8-bit code can only define 256 different characters. Notice how the EBCDIC codes are completely different to ASCII – if a message was sent that had been encoded using ASCII, but received by a system that used EBCDIC, then the resulting message would not make sense.



SAMPLE OF THE EBCDIC CHARACTER SET

65		81		97	/	113		129	a	145	j	161	~	177		193	A	209	J	225		241	1
66		82		98		114		130	b	146	k	162	s	178		194	B	210	K	226	S	242	2
67		83		99		115		131	c	147	l	163	t	179		195	C	211	L	227	T	243	3
68		84		100		116		132	d	148	m	164	u	180		196	D	212	M	228	U	244	4
69		85		101		117		133	e	149	n	165	v	181		197	E	213	N	229	V	245	5
70		86		102		118		134	f	150	o	166	w	182		198	F	214	O	230	W	246	6
71		87		103		119		135	g	151	p	167	x	183		199	G	215	P	231	X	247	7
72		88		104		120		136	h	152	q	168	y	184		200	H	216	Q	232	Y	248	8
73		89		105		121	`	137	i	153	r	169	z	185		201	I	217	R	233	Z	249	9
74	¢	90	!	106	|	122	:	138		154		170		186		202		218		234		250	|
75	.	91	$	107	,	123	#	139		155		171		187		203		219		235		251	
76	>	92	*	108	%	124	@	140		156		172		188		204		220		236		252	
77	(	93	)	109		125	'	141		157		173		189		205		221		237		253	
78	+	94	;	110	<	126	=	142		158		174		190		206		222		238		254	
79	|	95		111	?	127	"	143		159		175		191		207		223		239		255	eo
80	&	96	-	112		128		144		160		176		192	{	208	}	224	\	240	0		

Unicode Character Code

Unicode is an international system of representing characters using 16 bits. Using 16 bits means that 2^{16} = 65 536 different characters can be represented (thus overcoming the limitation of ASCII and EBCDIC).Unicode allows every character from most alphabets to have a code of its own – Chinese, Russian, Greek, Urdu etc, including Egyptian Hieroglyphics. Note that there are plenty of spare codes that are used for mathematical symbols, common graphics and even the Braille symbols.

4.0 Conclusion

This unit discussed the various codes that are used in digital computer. Binary Coded Decimal (BCD) is also discussed as well as compliment arithmetic

5.0 Summary

You have learnt:

The 1's complement of a binary number is derived by changing 1s to 0s and 0s to 1s.
The 2's complement of a binary number can be derived by adding 1 to the l's complement.
Binary subtraction can be accomplished with addition by using the 1 's or 2's complement method.
A positive binary number is represented by a 0 sign bit.
A negative binary number is represented by a 1 sign bit.
For arithmetic operations. negative binary numbers are represented in I's complement or 2's complement form.
A decimal number is converted to BCD by replacing each decimal digit with the appropriate 4-bit binary code.
The ASCII is a 7-bit alphanumeric code that is widely used in computer systems for input and output of information.
A parity bit is used to detect an error in a code.
Key terms and other bold terms in the chapter are defined in the end-of-book glossary.
Alphanumeric Consisting of numerals, letters, and other characters.
ASCII American Standard Code for Information Interchange; the most widely used alphanumeric code.
BCD Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

6.0 Tutor Marked Assignment

1. Convert the following binary numbers to the Gray code:

(a) 1100 (b) 1010 (c) 11010

2. Convert the following Gray codes to binary:

(a) 1000 (b) 1010 (c) 11101

3. Convertthe following decimal numbers to BCD:

(a) 124 (f) 520 (b) 128 (g) 329

4. Convert each of the BCD numbers to decimal:

(a) 0001 (b) 0001 1000 (c) 01000101 (d) 0110

7.0 References/ Further Reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJ Prentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
www.swarthmore.edu/NatSci/.../**BinaryMath**/**BinaryMath**.html
www.pages.cpsc.ucalgary.ca/~jacob/Courses/.../04-BitsAnd**Arithmetic**.pdf
www.allaboutcircuits.com
www.l3d.cs.colorado.edu/courses/CSCI1200-96/**binary**.html

Module 2

Logic Gates

Unit 1

Basic Logic Gates

Content

1.0 Introduction

Digital circuits operate in the binary mode where each input and output voltage is either 0 or 1 which represent a pred defined voltage ranges. In this chapter basic logic circuits (logic gates) which are the fundamental building block from which other logic circuits are constructed will be studied.

2.0 Learning Outcomes

At the end of this unit, you should be able to

Expalin the meaning logic
Contruct truth table is and how to consrtuct them
List different types of gates and their symbol
Draw logic diagrams
Explain boolean expression
Describe boolean expression from truth table and vice versa
Describe how to convert from truth table, logic circcuit diagrams and boolean expression
Implement logic circuit using logic gates
KnowUniversal gates - NAND and NOR.

3.0 Learning Contents

3.1 Logic Representation

The term logic is applied to digital circuits used to implement logic functions. Several kinds of digital logic circuits are the basic elements that form the building blocks for such complex digital systems as the computer.

There are three common ways in which to represent logic.

1. Truth Tables

2. Logic Circuit Diagram

3. Boolean Expression

We will discuss each herein and demonstrate ways to convert between them.

3.2 Truth Table

A truth table is a chart of 1's and 0's arranged to indicate the results (or outputs) of all possible inputs. The lists of all possible inputs are arranged in columns on the left and the resulting outputs are listed in columns on the right. There are 2 to the power n possible states (or combination of inputs). For example with three inputs there are 2^3=8 possible combination of inputs.

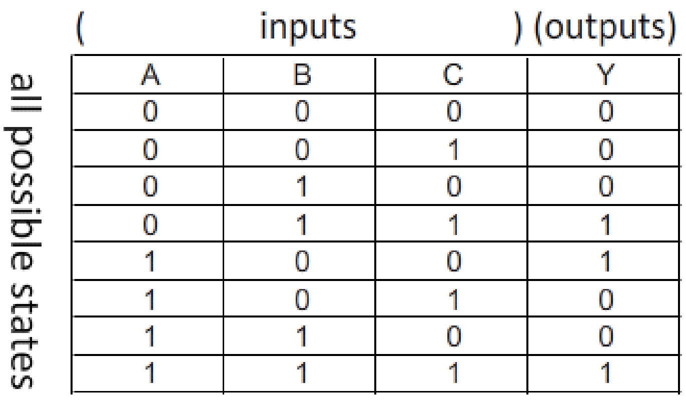
Below are some example truth-tables:

1. Consider a CL block with two inputs, A&B, and a single output Y. The output Y has value 1 if one, but not both, of the inputs is a 1.

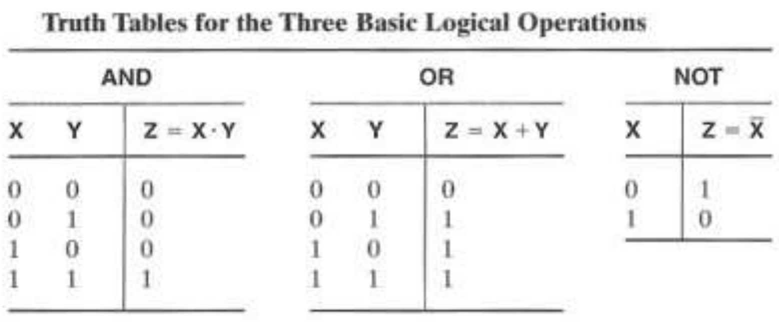
(inputs) (output)

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

2. a three input truth table with one output is shown below


(inputs) (outputs)
all possible states

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



Truth Tables for the Three Basic Logical Operations

AND

X	Y	Z = X·Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

X	Y	Z = X + Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

X	$Z = \overline{X}$
0	1
1	0

For example the truth table for F=X + Y’Z is shown below

X	Y	Z	Y’	Y’.Z	F
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0

0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

N.B Y' means Y complement and it could also be denoted as $\overline{Y}$

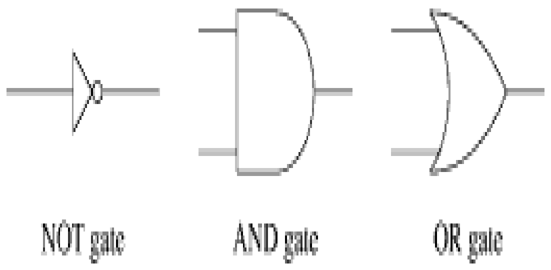
Construct a truth table for a two input OR and AND operation.

X	Y	OR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

3.3 Logic Diagram

At the most basic level, a computer is an electrical circuit. In this chapter, we'll examine a system that computer designers use for designing circuits, called a **logic circuit**.Logic gates are simple circuits (each with only a handful of transistors) that can be wired together to implement any CL function. Consider logic gates are primitive elements; they are the basic building blocks for circuits.

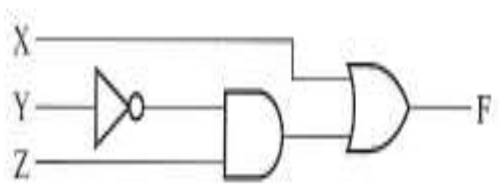
A logic circuit consists of lines, representing wires, and peculiar shapes called **logic gates**. There are three baisctypes of logic gates:



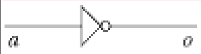
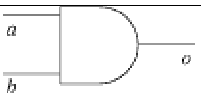
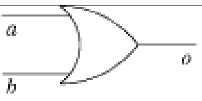
The relationship of the symbols to their names can be difficult to remember. It is handy to remember that the word *AND* contains a *D*, and this is what an AND gate looks like. We'll see how logic gates work in a moment.

Each wire carries a single information element, called a **bit**. A bit's value can be either 0 or 1. In electrical terms, you can think of zero volts representing 0 and five volts representing 1. It can also means Low or High, False or True.(In practice, there are many systems for representing 0 and 1 with different voltage levels. For our purposes, the details of voltage are not important.) The word *bit*, incidentally, comes from *B*inary dig*IT*; the term *binary* comes from the two (hence *bi*-) possible values.

Here is a diagram of one example logic circuit.



We'll think of a bit travelling down a wire until it hits a gate. You can see that some wires intersect in a small, solid circle: This circle indicates that the wires are connected, and so values coming into the circle continue down all the wires connected to the circle. If two wires intersect with no circle, this means that one wire goes over the other, like an Interstate overpass, and a value on one wire has no influence on the other.

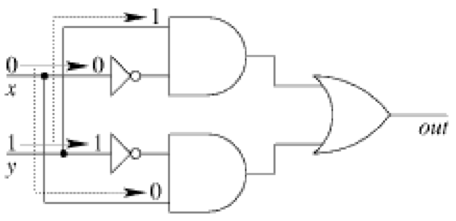
Figure3.1: Logicgate behaviour. 		
a o / 0 1 / 1 0	a b o / 0 0 0 / 0 1 0 / 1 0 0 / 1 1 1	A b O / 0 0 0 / 0 1 1 / 1 0 1 / 1 1 1
(a) NOT gate	(b) AND gate	(c) OR gate

a	o
0	1
1	0

a	b	o
0	0	0
0	1	0
1	0	0
1	1	1

A	b	O
0	0	0
0	1	1
1	0	1
1	1	1

Suppose that we take our example circuit and send a 0 bit on the upper input (*x*) and a 1 bit on the lower input (*y*). Then these inputs would travel down the wires until they hit a gate.



To understand what happens when a value reaches a gate, we need to define how the three gate types work.

NOT gate

Takes a single bit and produces the opposite bit figure 3.1(a). In our example circuit, since the upper NOT gate takes 0 as an input, it will produce 1 as an output.

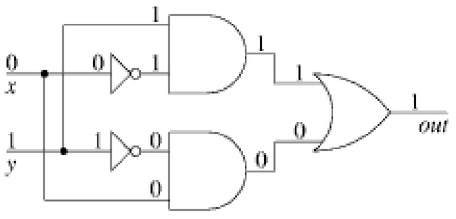
AND gate

Takes two inputs and outputs 1 only if both the first input *and* the second input are 1 figure 3.1(b). In our example circuit, since both inputs to the upper AND gate are 1, the AND gate will output a 1.

OR gate

Takes two inputs and outputs 1 if either the first input *or* the second input are 1 (or if both are 1). Figure 3.1(c)

After the values filter through the gates based on the behaviors of logic circuit below, the values in the circuit will be as follows.



Based on this diagram, we can see that when *x* is 0 and *y* is 1, the output *out* is 1.

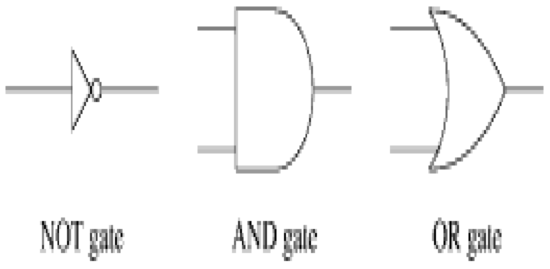
By doing the same sort of propagation for other combinations of input values, we can build up a table of how this circuit works for different combinations of inputs. We would end up with the following results.

x	Y	Out
0	0	0
0	1	1
1	0	1

1	1	0	

Draw the symbolic representation for NOT AND and OR Gates

The symbolic representation of the NOT AND and OR gates is as given below



Self Assessment Question 1

Use truth table to verify
(XYZ)’ = X’+Y’+Z’
X+YZ =(X+Y).(X+Z)

Self Assessment Answers

3.4 Universal Gate

A universal gate is a gate which can implement any Boolean function without need to use any other gate type.

The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not the other way around!!

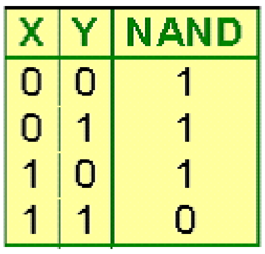
Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter not the other way around!!

NAND Gate

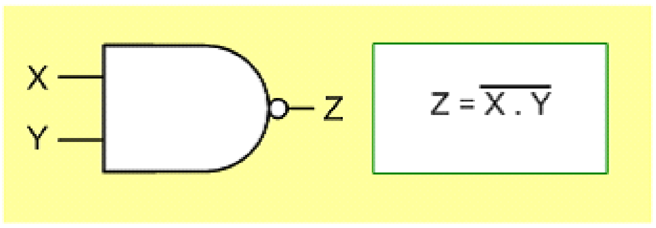
The NAND gate represents the complement of the AND operation. Its name is an abbreviation of NOT AND.

The graphic symbol for the NAND gate consists of an AND symbol with a bubble on the output, denoting that a complement operation is performed on the output of the AND gate.

The truth table and the graphic symbol of NAND gate is shown in the figure below.



X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

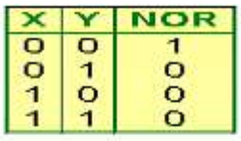


The truth table clearly shows that the NAND operation is the complement of the AND.

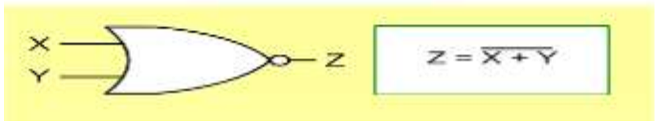
NOR Gate

The NOR gate represents the complement of the OR operation. Its name is an abbreviation of NOT OR.

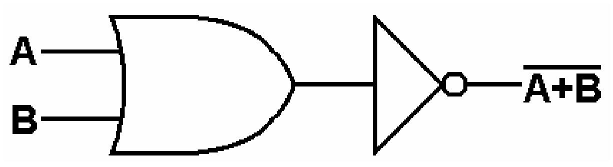
The graphic symbol for the NOR gate consists of an OR symbol with a bubble on the output, denoting that a complement operation is performed on the output of the OR gate. The truth table and the graphic symbol of NOR gate is shown in the figure below.



X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0



OR



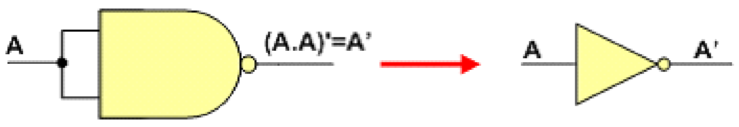
NAND Gate is a Universal Gate:

To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

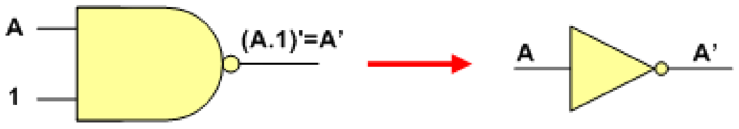
Implementing an Inverter Using only NAND Gate

The figure shows two ways in which a NAND gate can be used as **an inverter (NOT gate)**.

1. All NAND input pins connect to the input signal **A**gives an output **A**’.

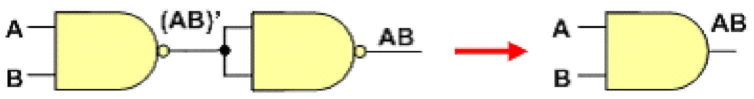


2. One NAND input pin is connected to the input signal **A** while all other input pins are connected to logic **1**. The output will be **A’**.

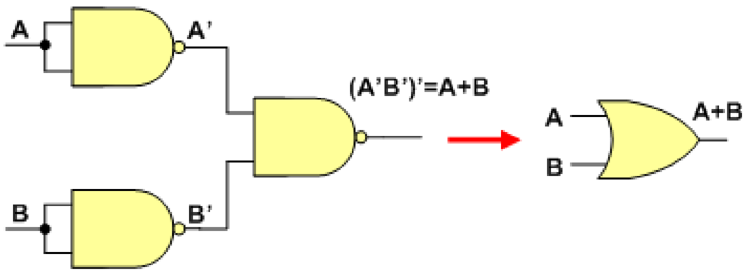


Implementing AND Using only NAND Gates

An AND gate can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).



An OR gate can be replaced by NAND gates as shown in the figure (The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters).



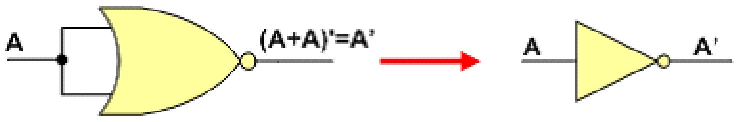
NOR Gate is a Universal Gate:

To prove that any Boolean function can be implemented using only NOR gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

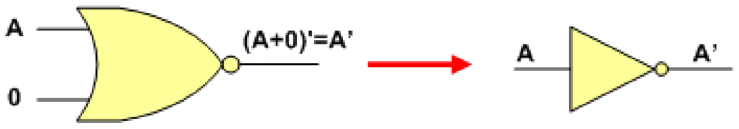
Implementing an Inverter Using only NOR Gate

The figure shows two ways in which a NOR gate can be used as **an inverter (NOT gate)**.

1.All NOR input pins connect to the input signal **A** gives an output **A'**.



2. One NOR input pin is connected to the input signal **A** while all other input pins are connected to logic **0**. The output will be **A'**.



Self Assessment Questions

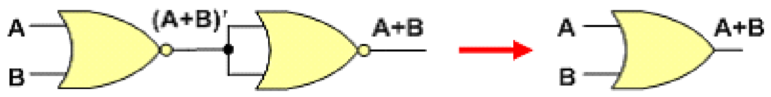
What is the common entity in the NOR NAND Gates

Self Assessment Answers

The common entity feature in the NOR and NAND gates is the NOT gate

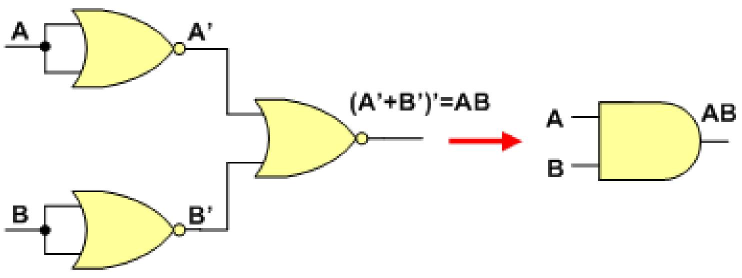
Implementing OR Using only NOR Gates

An OR gate can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter)



Implementing AND Using only NOR Gates

An AND gate can be replaced by NOR gates as shown in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters)



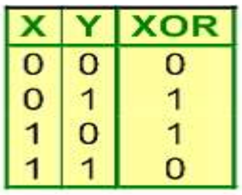
Thus, the NOR gate is a universal gate since it can implement the AND, OR and NOT functions.

3.5 XOR Gate:

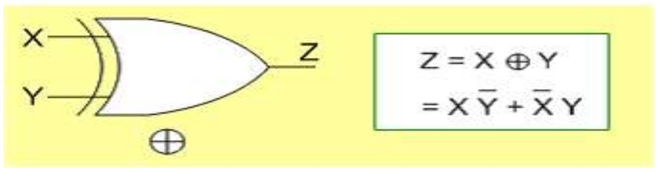
The exclusive-OR (XOR), operator uses the symbol ⊕, and it performs the following logic operation:

$X \oplus Y = X\,Y' + X'\,Y$

The graphic symbol and truth table of XOR gate is shown in the figure.This gate gives a high output when A OR B are high, but not both. The equivalent circuit is:



X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0



$$Z = X \oplus Y$$
$$= X\overline{Y} + \overline{X}Y$$

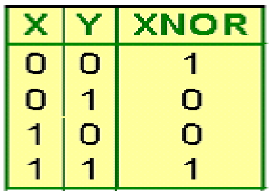
For more than two inputs, the XOR gate generates a 1 at its output if the number of 1’s at its input is odd

3.6 XNOR Gate:

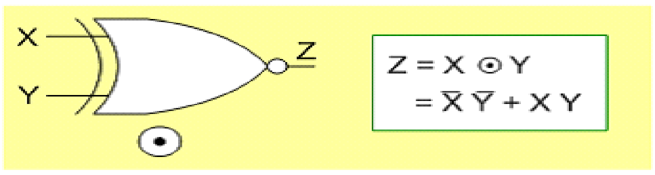
The exclusive-NOR (XNOR), operator uses the symbol □, and it performs the following logic operation

X □□Y = X Y + X Y = (X ⊕Y)’

The graphic symbol and truth table of XNOR (Equivalence) gate is shown in the figure



X	Y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1



$$Z = X \odot Y$$
$$= \overline{X}\,\overline{Y} + XY$$

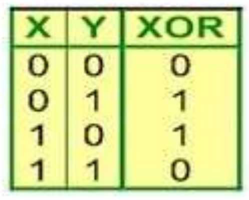
The result is 1 when either both X and Y are 0’s or when both are 1’s. That is why this gate is often referred to as the **Equivalence** gate.

The truth tables clearly show that the exclusive-NOR operation is the complement of the exclusive-OR.

N.B: XOR and XNOR gates are usually found as 2-input gates. No multiple-input XOR/XNOR gates are available since they are complex to fabricate with hardware.

Reconstruct the XOR Truth table for a two input system

The truth table for the XOR is as given below

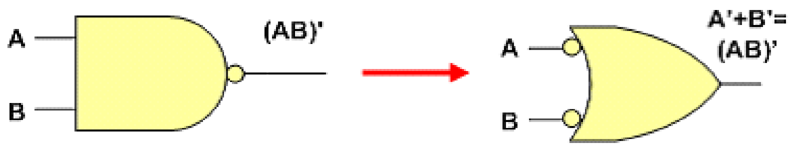


X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

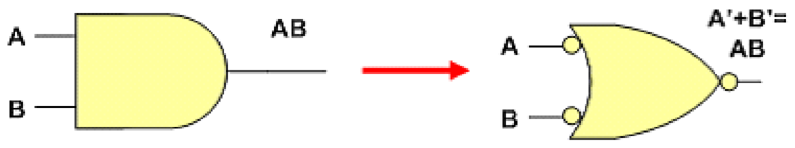
3.7 Equivalent Gates

The shown figure summarizes important cases of gate equivalence

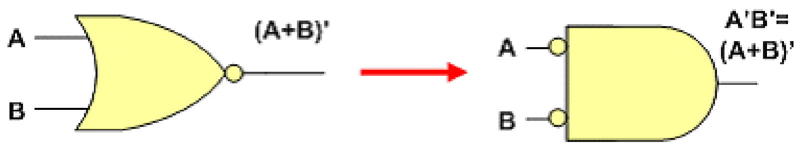
A NAND gate is equivalent to an inverted-input OR gate



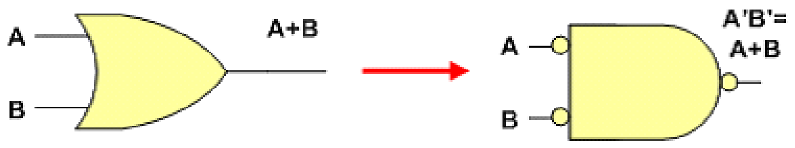
An AND gate is equivalent to an inverted-input NOR gate.



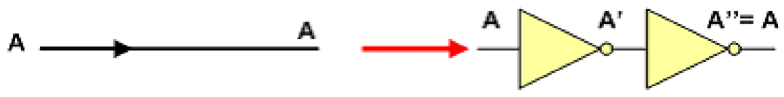
A NOR gate is equivalent to an inverted-input AND gate.



An OR gate is equivalent to an inverted-input NAND gate.



Two NOT gates in series are same as a buffer because they cancel each other as A'' = A.



STEPs In Designing A Circuit

Step Description

Step 1 **Capture** the function Create a truth table or equations, ***whichever is*** most natural for the given problem, to describe the desired behavior of the combinational logic.

Step 2 **Convert** to equations This step is only necessary if you captured the truth table instead of equations.

Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.

Step 3 **Implement** as agate-based circuit For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

Design example:

1. Converting English Problem to Boolean Logic: A fire sprinkler system should spray water if high heat is sensed and the system is set to enable.

Answer

Identify and label variables:

hrepresents "high heat is sensed"

erepresents "enabled"

F represents "spraying water"

Write Boolean Equation expressing functionality described

F=h AND e

Design example 2

2. Circuit Description. Turn warning light on if driver in driver's seat, key inserted, seat belt not fastened

answer

Identify and label variables

s=1: seat belt fastened

k=1: key inserted

p=1: person in seat

w=1: warning light on

The Boolean Equation expressing functionality described person in seat, and seat belt not fastened, and key inserted is stated below.

w = p AND NOT(s) AND k

Self AssessmentQuestion

Circuit Description
Design an automatic sliding door. Open the door if the door is set to be manually held open, Open the door if the door is not set to be manually open, and a person is detected, However, in either case, we only open the door if the door is not set to stay closed. Drwa the circuit

Self assessment Answers

4.0 Conclusion

This unit discussed truth tables, logic circuits and gates as well as universal gates and their implementation. This has enabled you to know the different logic circuits and their corresponding truth table.

5.0 Summary

You have learnt:

The inverter output is the complement of the input.
The AND gate output is HIGH only when all the inputs are HIGH.

The OR gate output is HIGH when any of the inputs is HIGH.
The NAND gate output is LOW only when all the inputs are HIGH.
The NAND can be viewed as a negative-OR whose output is HIGH when any input is LOW.
The NOR gate output is LOW when any ofthe inputs is HIGH.
The NOR can be viewed as a negative-AND whose output is HIGH only when all the inputs
are LOW.
The exclusive-OR gate output is HIGH when the inputs aTe not the same.
The exclusive-NOR gate output is LOW when the inputs are not the same.
Distinctive shape symbols and truth tables for various logic gates (limited to 2 inputs) are shown

6.0 Tutor-Marked Assignment

Draw the truth truth table and circuit diagram for the following expressios
Y= AB + BC
Z= AB'C
K= XYZ +XZ'
Design a circuit that can detect two consecutive 1s in a 4-bit input: abc for example: 0111 yields 1 while 1001 yields 0 etc

Verify that X'Y'+X'Y+XY=X'Y by using truth table

How can you implement an inverter using NAND and NOR gates respective

7.0 References/Further Reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

http://drstienecker.com/tech-332/3-logic-circuits-boolean-algebra-and-truth- tables/
http://www.courses.ebe.uct.ac.za/eee317w/1.%20Basic%20Logic%20Design.
http://ozark.hendrix.edu/~burch/cs/230/cso/ch07-gates/index.html
http://opencourseware.kfupm.edu.sa/colleges/ces/ee/ee200/files%5C3HandoutLecture_19.pdf
http://www.indiabix.com/digital-electronics/combinational-logic-circuits

Unit 2

Boolean Algebra

Content

1.0 Introduction

In 1854, George Boole published a work titled An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities. It was in this publiccationthe a “logica1 algebra:' known today as Boolean algebra, waa formulated. Boolean algebral is a convenient and systematic way of expressing and analyzing the operation of logic circuits. Claude Shannon was the first to apply Boole's work to the analysis and design of logic circuits. In 1938, Shannon wrote a thesis at MIT titled A Symbolic Analysis of Relay and Switching Circuits. This chapter covers the laws, rules, and theorems of Booleanalgebral and their application to digita1 circuits. You willlearn how to define a given circuit with a Boolean expression and then evaluate its operation. You will also learn how to simplify logic circuits using the methods of Boolean algebra.

2.0 Learning Outcomes

At the end of this unit, you should be able to:

Apply the basic laws and rules of Boolean algebra
Apply DeMorgan's theorems to Boolean expressions
Describe gate networks with Boolean expressions
Evaluate Boolean expressions
Simplify expressions by using the laws and rules of Boolean algebra

3.0 Learning Contents

3.1 Boolean Algebra

In the previous section, we saw how logic circuits work. This is helpful when you want to understand the behavior of a circuit diagram. But computer designers face the opposite problem: Given a desired behavior, how can we build a circuit with that behavior?

It is a system of logic designed by George Boole in the middle of the nineteenth century, forms the foundation for modern computers. George Boole noticed that logical functions could be built from AND, OR, and NOT gates and that this observation leads one to be able to reason about logic in a mathematical system.

As Boole was working in the nineteenth century, of course, he wasn't thinking about logic circuits. He was examining the field of logic, created for thinking about the validity of philosophical arguments. Philosophers have thought about this subject since the time of Aristotle. Logicians formalized some common mistakes, such as the temptation to conclude that if *A* implies *B*, and if *B* holds, then *A* must hold also. (Brilliant people wear glasses, and I wear glasses, so I must be brilliant.)

As a mathematician, Boole sought a way to encode sentences like this into algebraic expressions, and he invented what we now call **Boolean expressions**. An example of a Boolean expression is y *x*+ y x. A line over a variable (or a larger expression) represents a NOT; for example, the expression *y* corresponds to feeding *y* through a NOT gate.

Multiplication (as with *x y*) represents AND. After all, Boole reasoned, the AND truth table is identical to a multiplication table over 0 and 1. Addition (as with *x* + *y*) represents OR. The OR truth table doesn't match an addition table over 0 and 1 exactly although 1 plus 1 is 2, the result of 1 OR 1 is 1 but, Boole decided, it's close enough to be a worthwhile analogy.

In Boolean expressions, we observe the regular order of operations: Multiplication (AND) comes before addition (OR). Thus, when we write *y x*'+ *y*'*x*, we mean (*y x'*) + (*y' x*). We can use parentheses when this order of operations isn't what we want. For NOT, the bar over the expression indicates the extent of the expression to which it applies; thus, (*x* + *y)'* represents NOT (*x* OR *y*), while *x*'+ y' represents (NOT *x*) OR (NOT *y*).

A warning: Students new to Boolean expressions frequently try to abbreviate *x' y'* as (*x y)'* — that is, they draw a single line over the whole expresion, rather than two separate lines over the two individual pieces. This abbreviation is *wrong*. The first, *x' y'*, translates to (NOT *x*) AND (NOT *y*) (that is, both *x* and *y* are 0), while (*x y)'* translates to NOT (*x* AND *y*) (that is, *x* and *y* aren't both 1). We could draw a truth table comparing the results for these two expressions.

x	y	x'	y'	x' y'	x y	(x y)'
0	0	1	1	1	0	1
0	1	1	0	0	0	1
1	0	0	1	0	0	1
1	1	0	0	0	1	0

NB $x' = \overline{x}$

$(xy)' = x\overline{y}$

Since the fifth column (*x' y'*) and the seventh column (*x y*)' aren't identical, the two expressions aren't equivalent. Every expression directly corresponds to a circuit and vice versa.

Self Assessment Questions

In Boolean operation what logic gate is used to implement multiplication ?
In Boolean operation what logic gate is used to implement addition ?

Self Assessment Answers

The AND gate is used to implement Multiplication in Boolean operation
The OR gate is used to implement Addition in Boolean operation

3.2 Algebraic Laws

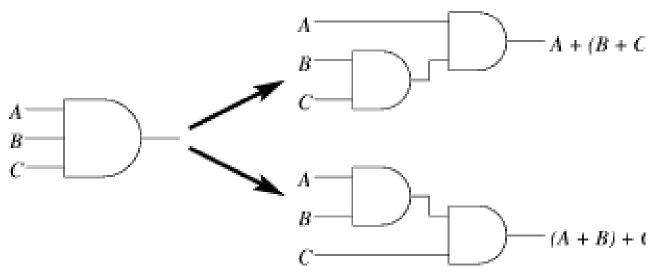
Boole's system for writing down logical expressions is called an *algebra* because we can manipulate symbols using laws similar to those of algebra. For example, the commutative law applies to both OR and AND. To prove that OR is commutative (that is, that $A + B = B + A$), we can complete a truth table demonstrating that for each possible combination of A and B, the values of $A + B$ and $B + A$ are identical.

A	B	$A + B$	$B + A$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Since the third and fourth columns match, we would conclude that $A + B = B + A$) is a universal law.

Since OR (and AND) are commutative, we can freely reorder terms without changing the meaning of the expression. The commutative law of OR would allow us to transform y x'+ y' x into y' x + y x', and the commutative law of AND (applied twice) allows us to transform y 'x + y x' to x y'+ x'y.

Similarly, both OR and AND have an associative law (that is, $A + (B + C) = (A + B) + C$). Because of this associativity, we won't bother writing parentheses across the same operator when we write Boolean expressions. In drawing circuits, we'll freely draw AND and OR gates that have several inputs. A 3-input AND gate would actually correspond to two 2-input AND gates when the circuit is actually wired. There are two possible ways to wire this.



Because of the associative law for AND, it doesn't matter which we choose.

T**able 3:** A sampler of important Boolean identities

	AND	OR
Commutative	$A.B = B.A$	$A + B = B + A$
Associative	$A (B C) = (A B) C$	$A + (B + C) = (A + B) + C$
Identity	$A \cdot 1 = A$	$A + 0 = A$
Distributive	$A (B + C) = A B + A C$	$A + B C = (A + B) (A + C)$
one/zero	$A \cdot 0 = 0$	$A + 1 = 1$
Idempotency	$A A = A$	$A + A = A$
Inverse	$A A = 0$	$A + \overline{A} = 1$
DeMorgan's law	$\overline{A B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$
double negation	$\overline{\overline{A}} = A$	

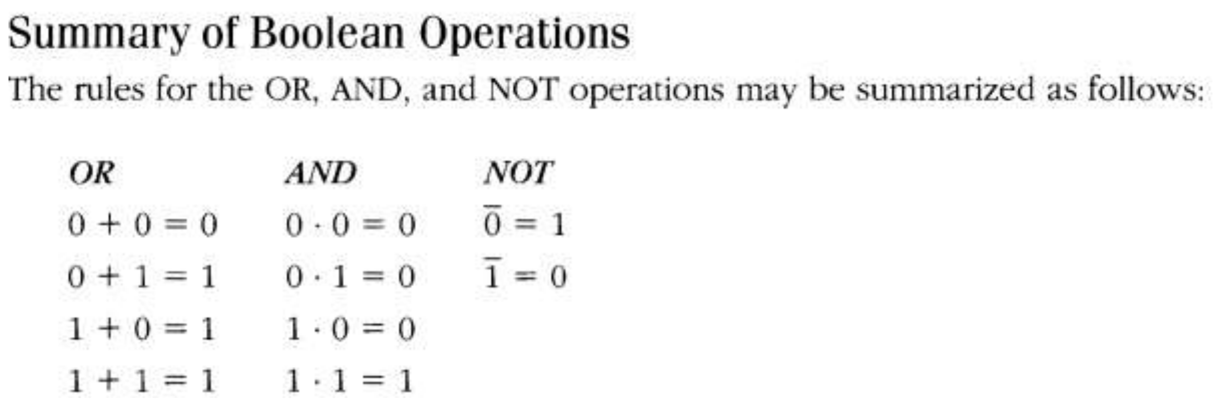
Other useful Boolean rules for simplification are:

A+AB=A

A+A’B=A+B

NB A’= $\overline{A}$

There are many such laws, summarized in table above. This includes analogues to all of the important algebraic laws dealing with multiplication and addition.



Summary of Boolean Operations

The rules for the OR, AND, and NOT operations may be summarized as follows:

OR	*AND*	*NOT*
$0 + 0 = 0$	$0 \cdot 0 = 0$	$\overline{0} = 1$
$0 + 1 = 1$	$0 \cdot 1 = 0$	$\overline{1} = 0$
$1 + 0 = 1$	$1 \cdot 0 = 0$	
$1 + 1 = 1$	$1 \cdot 1 = 1$	

Self Assessment Questions

What Law does the expression $A + (B + C) = (A + B) + C)$ obeys?

Self Assessment Answers

The expression obeys the Associative Law

3.3 DeMorgan’s Theorem

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms.DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates, which were shown in the table 3.

One of DeMorgan's theorems is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables.Stated another way, The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is

(XY)’ = X’ + Y’

DeMorgan's second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables. Stated another way, The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables,

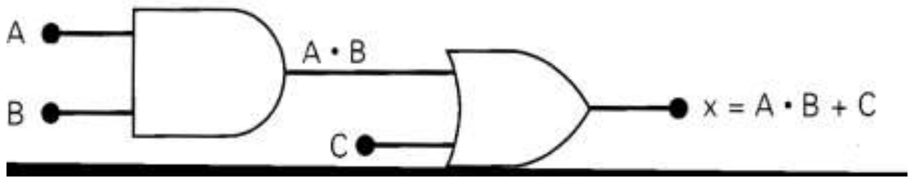
The formula for expressing this theorem for two variables is

(X + Y)’ = X’Y’

The two laws can be confirmed by using truth table (exercise)

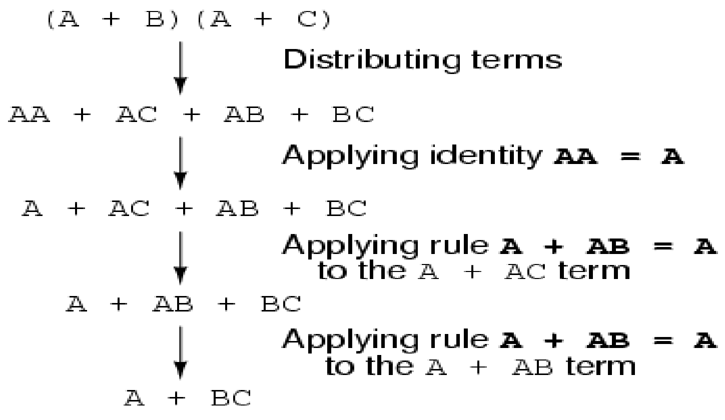
DeMorgan’s theorem can be applied to three or more variables.

e.g (ABC)’= A’+B’+C’ ==$\overline{ABC}$ =$\overline{A}$+$\overline{B}$+$\overline{C}$



Example 1: Simplify (A+B) (A+C)

Soln


(A + B) (A + C)

↓ Distributing terms

AA + AC + AB + BC

↓ Applying identity **AA = A**

A + AC + AB + BC

↓ Applying rule **A + AB = A** to the A + AC term

A + AB + BC

↓ Applying rule **A + AB = A** to the A + AB term

A + BC

Simplify AB+ BC(B+C)

Simplify $\overline{A+\overline{BC}}$

The solution of the AB+ BC (B+C) is as given

$$\begin{tikzcd}
AB + BC(B + C) \arrow[d, "\text{Distributing terms}"] \\
AB + BBC + BCC \arrow[d, "\text{Applying identity } AA = A \text{ to 2nd and 3rd terms}"] \\
AB + BC + BC \arrow[d, "\text{Applying identity } A + A = A \text{ to 2nd and 3rd terms}"] \\
AB + BC \arrow[d, "\text{Factoring } B \text{ out of terms}"] \\
B(A + C)
\end{tikzcd}$$

$$\begin{tikzcd}
\overline{A + \overline{BC}} \arrow[d, "\text{Breaking shortest bar (multiplication changes to addition)}"] \\
\overline{A + (\overline{B} + \overline{C})} \arrow[d, "\text{Applying associative property to remove parentheses}"] \\
\overline{A + \overline{B} + \overline{C}} \arrow[d, "\text{Breaking long bar in two places, between 1st and 2nd terms; between 2nd and 3rd terms}"] \\
\overline{A}\ \overline{\overline{B}}\ \overline{\overline{C}} \arrow[d, "\text{Applying identity } \overline{\overline{A}} = A \text{ to } \overline{\overline{B}} \text{ and } \overline{\overline{C}}"] \\
\overline{A}BC
\end{tikzcd}$$

Self Assessment Question 1

1. Simplify $\overline{\overline{A + BC} +}\ \overline{\overline{A}}B$

2. Simplify $\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

Also we have the following examples in addition to the above:

Simplify y = ab + a + c

= a(b + 1) + c distribution, identity

= a(1) + c law of 1's

= a + c identity

Prove that X + X'Y=X +Y

X + X'Y = X.1 + X'Y

= X.(1+Y) + X'Y

= X + XY + X'Y

= X + (XY +X'Y)

= X + Y(X +X')

= X + Y

Example ``Consensus Theory``

Show that XY + X`Z + YZ = XY + X`Z

Proof:

LHS = XY + X`Z + YZ

= XY + X`Z + YZ **.**1

= XY + X`Z + YZ **.**(X +X`)

= XY + X`Z + YZX + YZX`

= XY + YZX + X`Z + YZX`

= XY(1 + Z) + X`Z(1 + Y)

= XY **.**1 + X`Z **.**1= XY + X`Z **= LHS**

Example: Using the above laws, simplify the following expression: (A + B) (A + C)

Q = (A + B)(A + C)

	AA + AC + AB + BC	- Distributive law
	A + AC + AB + BC	- Identity AND law (A.A = A)
	A(1 + C) + AB + BC	- Distributive law
	A.1 + AB + BC	- Identity OR law (1 + C = 1)
	A(1 + B) + BC	- Distributive law
	A.1 + BC	- Identity OR law (1 + B = 1)
Q =	A + BC	- Identity AND law (A.1 = A)

Then the expression: (A + B)(A + C) can be simplified to A + BC

Converting A Truth Table

Now we can return to our problem: If we have a particular logical function we want to compute, how can we build a circuit to compute it? We'll begin with a description of the logical function as a truth table. Suppose we start with the following function for which we want a circuit.

x	Y	z	out
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Given such a truth table defining a function, we'll build up a Boolean expression representing the function. For each row of the table where the desired output is 1, we describe it as the AND of several factors.

X	Y	z	out	description

0	0	1	1	x'y' z
0	1	0	1	x' *y* z
1	1	0	1	*x y* z'
1	1	1	1	x y z

To arrive at a row's description, we choose for each variable either that variable or its negation, depending on which of these is 1 in that row. Then we take the AND of these choices. For example, if we consider the first of the rows above, we consider that since *x* is 0 in this row, *x'* is 1; since *y* is 0, y' is 1; and *z* is 1. Thus, our description is the AND of these choices, x'y' z. This expression gives 1 for the combination of values on this row; but for other rows, its value is 0, since every other row is different in some variable, and that variable's contribution to the AND would yield 0.

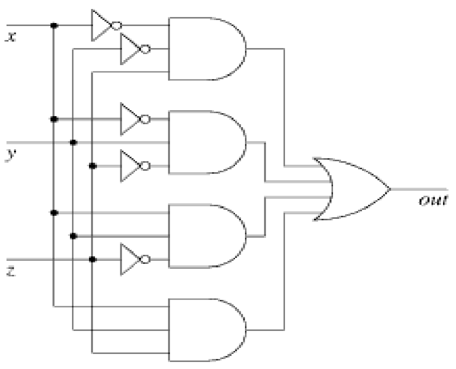
Once we have the descriptions for all rows where the desired output is 1, we observe the following: The value of the desired circuit should be 1 if the inputs correspond to the first 1-row, the second 1-row, the third 1-row, *or* the fourth 1-row. Thus, we'll combine the expressions describing the rows with an OR:

x'y' z + x'yz' + x yz' + x y z

Note that we do not include rows where the desired output is 0 — for these rows, we want none of the AND terms to yield 1, so that the OR of all terms gives 0.

This expression leads immediately to the circuit of Figure below

Figure : A circuit derived from a given truth table.



The final expression we get is called a **sum of products** expression. It is called this because it is the OR (a sum, if we understand OR to be like addition) of several ANDs (products, since AND corresponds to multiplication). We call this technique of building an expression from a truth table the **sum of products technique (next section)**.

In general, this technique allows us take any function over bits and build a circuit to compute that function. The existence of such a technique proves that circuits can compute any logical function.

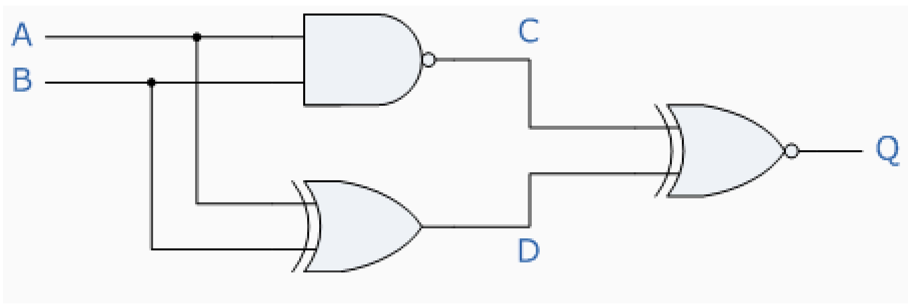
Note, incidentally, that the depth of this circuit will always be three (or less), since every path from input to output will go through a NOT gate (maybe), an AND gate, and an OR gate. Thus, this technique shows that it's never necessary to design a circuit that is more than three gates deep. Sometimes, though, designers build deeper circuits because they are concerned not only with speed, but also with size: A larger circuit can often accomplish the same task using fewer gates.

More Boolean Algebra Examples

Here are a few examples of how to use **Boolean Algebra** to simplify larger logic circuits.

Example No1

Construct a Truth Table for the logical functions at points C, D and Q in the following circuit and identify a single logic gate that can be used to replace the whole circuit.



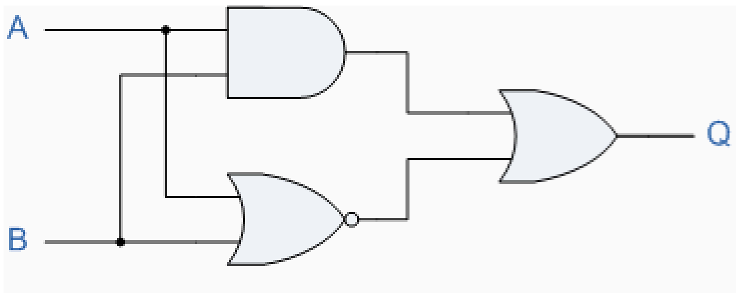
First observations tell us that the circuit consists of a 2-input NAND gate, a 2-input EX-OR gate and finally a 2-input EX-NOR gate at the output. As there are only 2 inputs to the circuit labelled A and B, there can only be 4 possible combinations of the input (2^2) and these are: 0-0, 0-1, 1-0 and finally 1-1. Plotting the logical functions from each gate in tabular form will give us the following truth table for the whole of the logic circuit below.

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

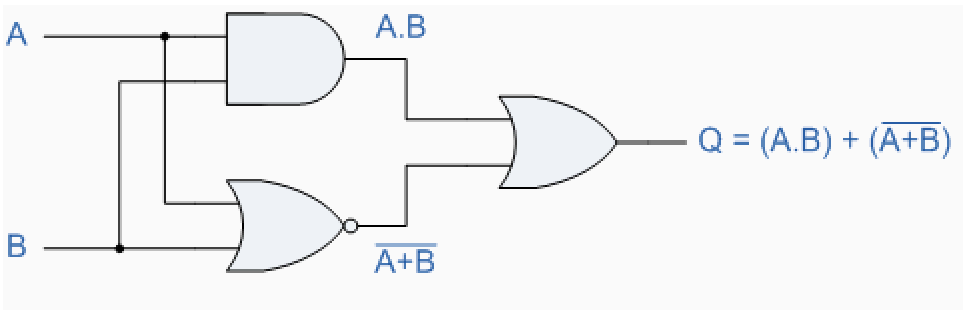
1	1	0	0	1

From the truth table above, column C represents the output function from the NAND gate and column D represents the output function from the Ex-OR gate. Both of these two output expressions then become the input condition for the Ex-NOR gate at the output. It can be seen from the truth table that an output at Q is present when any of the two inputs A or B are at logic 1. The only truth table that satisfies this condition is that of an OR Gate. Therefore, the whole of the above circuit can be replaced by just one single **2-input** OR Gate.

Find the Boolean algebra expression for the following system.



The system consists of an AND Gate, a NOR Gate and finally an OR Gate. The expression for the AND gate is A.B, and the expression for the NOR gate is A+B. Both these expressions are also separate inputs to the OR gate which is defined as A+B. Thus the final output expression is given as:



The output of the system is given as Q = (A.B) + (A+B), but the notation A+B is the same as the De Morgan´s notation A.B, Then substituting A.B into the output expression gives us a final output notation of Q = (A.B)+(A.B), which is the Boolean notation for an Exclusive-NOR Gate as seen in the previous section.

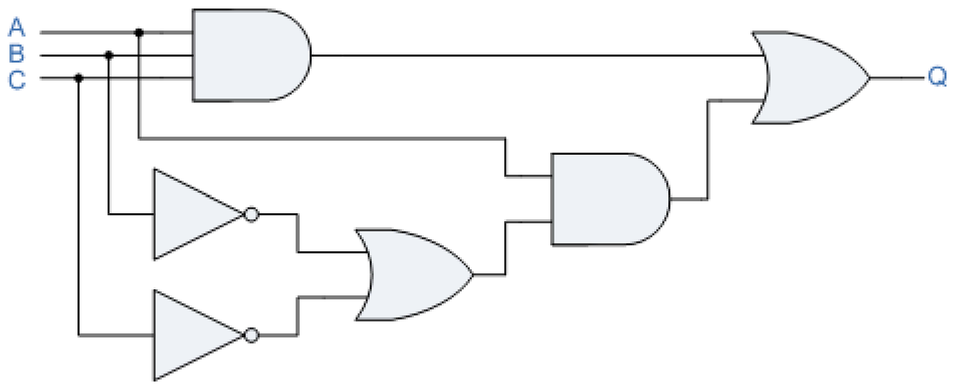
Inputs		Intermediates		Output
B	A	A.B	A + B	Q
0	0	0	1	1
0	1	0	0	0

1	0	0	0	0
1	1	1	0	1

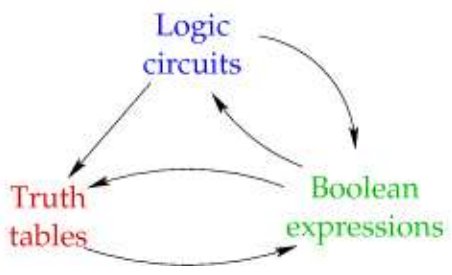
Then, the whole circuit above can be replaced by just one single Exclusive-NOR Gate and indeed an Exclusive-NOR Gate is made up of these individual gates.

Self Assessment Question

Find the Boolean algebra expression for the following system.



To summarize: We have seen three ways of describing a Boolean function: *logic circuits*, *truth tables*, and *Boolean expressions*. Moreover, we have seen systematic ways to convert between the three techniques, diagrammed as arrows in Figure 7below.



The only missing arrow is the conversion from truth tables to circuits; we can handle that, though, by converting the truth table to a Boolean expression (using the sum of products technique) and converting that into a circuit. This will be discussed in the next unit

4.0 Conclusion

This unit examines the concept ofBoolean algebra ant its various laws including DeMorgan’s theorem which all help in the simplification of logis circuit. We also discussed how to convert among the three representations as represente in the diagram above

5.0 Summary

You have learnt:

Boolean algebra is a mathematical tool used in the analysis and design of digital circuits
Tha basic Boolean operation are the OR,AND and NOT
Commutative laws: A + B = B + A, AB = BA
Associative laws: A + (B + C) = (A + B) + C   A(BC) = (AB)C
Distributive law: A(B + C) = AB + AC
Boolean rules: . A + 0 = A ,. A+l=l,   A. 0 = 0,   A.1=A,   A + A = A
DeMorgan's theorems:
The complement of a product is equal to the sum of the complements of the terms in the product. (XY)’=X’+Y’
The complement of a sum is equal to the product of the complements of the terms in the sum. (X + Y)’ = X’Y’
Complement is the inverse or opposite of a number. In Boolean algebra, the inverse function, expressed with a bar over a variable. The complement of a 1 is O. and vice versa.
Truth-table to Boolean Expression. Write the canonical form and follow with algebraic simplification if desired.
Boolean Expression to Truth-table. Evaluate expression for all input combinations and record output values.
Boolean Expression to Gates. Use AND gates for the AND operators, OR gates for the OR operators, and inverters for the NOT operator. Wire up the gates the match the structure of the expression.
Gates to Boolean Expression. Reverse the above process.
Gates to Truth-table. Pass through all input combination and evaluate output.
Truth-table to Gates. Map to Boolean expression then to gates.

6.0 Tutor-Marked Assignment

1.Show that (RXYZ)’ =R’+X’+Y’+Z’ by using a truth table

2. Show that   A ‘B+ B’ C’ + AB + B’ C = 1

3 Simplify Y + X ‘Z + X Y’

4. Prove the following

(a) $\overline{X}\,\overline{Y}+\overline{X}Y+XY=\overline{X}+Y$
(b) $\overline{A}B+\overline{B}\,\overline{C}+AB+\overline{B}C=1$
(c) $Y+\overline{X}Z+X\overline{Y}=X+Y+Z$
(d) $\overline{X}\,\overline{Y}+\overline{Y}Z+XZ+XY+Y\overline{Z}=\overline{X}\,\overline{Y}+XZ+Y\overline{Z}$

5.

+Given that $A\cdot B=0$ and $A+B=1$, use algebraic manipulation to prove that

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

6.

Simplify each of the following expressions using DeMorgan's theorems.

(a) $\overline{\overline{AB}\,\overline{C}}$ **(d)** $\overline{A+\overline{B}}$ **(g)** $\overline{A\overline{(B+\overline{C})}D}$
(b) $\overline{\overline{A}+\overline{BC}}$ **(e)** $\overline{\overline{AB}}$ **(h)** $\overline{(M+\overline{N})(\overline{M}+N)}$
(c) $\overline{AB\overline{CD}}$ **(f)** $\overline{\overline{A}+\overline{C}+\overline{D}}$ **(i)** $\overline{\overline{AB}CD}$

7.0 References/Further Reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
http://drstienecker.com/tech-332/3-logic-circuits-boolean-algebra-and-truth-tables/
http://www.courses.ebe.uct.ac.za/eee317w/1.%20Basic%20Logic%20Design.pdf
http://ozark.hendrix.edu/~burch/cs/230/cso/ch07-gates/index.html
http://www.indiabix.com/digital-electronics/combinational-logic-circuits

Unit 3

Standard Form

Content

1.0 Introduction

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of-sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

2.0 Learning Outcomes

At the end of this unit you should be able to:

Manipulate Boolean Expressions and Simplify Them
Learn how to derive a Boolean expression of a function defined by its truth table.
Identify a sum-of-products expression - Determine the domain of a Boolean expression
Convert any sum-of-products expression to a standard form
Evaluate a standard sum-of-products expression in terms of binary values - Identify a product-of- sums expression
Convert any product-of-sums expression to a standard form
Evaluate a standard product-of-sums expression in terms of binary values - Convert from one standard form to the other
Derive expressions in one of two possible standard forms: *The Sum of Min-terms* or the *Product of Max-Terms*.
Map these expressions into logic circuit implementations (2- Level Implementations).

3.1 Sum of Product (SOP) form

A product term was defined in Section 4-1 as a term consisting of the product (Boolean multiplication) of literals (variables or their complements). When two or more product terms are summed by Boolean addition the resulting expression is a sum-of-products (SOP). Some examples are

AB + ABC

AB’C + CD’E’ + BCD

In an SOP expression.a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar.

Implementing an SOP expression simply requires ORing the outputs of two or more AND gates. A product term is produced by an AND operation, and the sum (addition) of two or more product terms is produced by an OR operation. Therefore, an SOP expression can be implemented by AND-OR logic in which the outputs of a number (equal to the number of product terms in the expression) of AND gates connect to the inputs of an OR gate,

3.2 Standard SOP Form

So far, you have seen SOP expressions in which some of the product terms do not contain all ofthe variable in the domain of the expression. For example, the expression ABC + ABD + ABCD has a domain made up of the variables A, B, C. and D. However, notice that the complete set of variaes in the domain is not represented in the first two terms of the expression; that is, D or D is missing from the first term and Cor C is missing from the second term. A STandard SOP expression is one in which all the variables in the domain appear in each product term in the expression. For example, ABCD + AB'C'D + A'B'C'D' is a standard SOP expression. Standard SOP expressions are important in constructing truth tables covered in Unit 1, and in the Karnaugh map simplification method, which is covered will be covered in the next unit. Any nonstandard SOP expression (referred to simply as SOP) can be converted to the standard form using Boolean algebra.

3.3 Product of Sum

When two or more sum terms are multiplied, the resulting expression is a product-of-sums (POS). Some examples are

(A + B)(A + B + C)

(A' + B' + C')(C + D's + E)(B + C + D)

(A + B)(A + B + C)(A + C)

A POS expression can contain a single-variable term, as in A(A + B + C)(B + C + D).

In a POS expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term c an have an o verbar. For example, a POS expression can have the term A + B + C but not A + B + C.

Implementation of a POS Expression Implementing a pas expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation nd the product of two or more sum terms is produced by an AND operation. Therefore.a POS expression can be implemented by logic in which the outputs of a number (equal to the number of sum terms in the expression) of OR gates connect to the inputs of an AND gate

3.4 The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression. For example.the expression

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

has a domain made up of the variables A. B. C, and D. Notice that the complete set of variables in the domain is not represented in e first two tenns of the expression; that is, D or

D is missing from the first term and C or C is missing from the second term.

A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression. For example,

(A’ + B’ + C’ + D’)(A + B’ + C + D)(A + B + C + D’)

is a standard POS expression. Any nonstandard pas expression (referred to simply as POS) can be converted to the standard form using Boolean algebra.

Self Assessment Questions

Identify each of the following expressions as SOP, standard SOP, POS, or standard POS:
(a) AB + A’BD + A’CD’
(c) A’BC + ABC’
(b) (A + B’ + C)(A + B + C’)
(d) A(A + C’) (A + B)

Self Assessment Answers

a. SOP
b. Standard SOP
c. Standard POS
d. POS

3.5 Min Term

Consider a system of 3 input signals (variables) X, Y, & Z. A term which ANDs all input variables, either in the true or complement form, is called a minterm.

Thus, the considered 3-input system has 8 minterms, namely: *X’ Y’ Z’*, *X’ Y’ Z* , *X’ Y Z’* , *X’ Y Z* , *X Y’ Z’* , *X Y’ Z* , *X Y Z’* &*X Y Z*

Each minterm equals 1 at exactly one particular input combination and is equal to 0 at all other combinations.Thus, for example, *X Y Z* is always equal to 0 except for the input combination ***xyz*** = **000**, where it is equal to 1.

Accordingly, the minterm*X Y Z* is referred to as m_0.

In general, minterms are designated *mi*, where *i* corresponds the input combination at which this minterm is equal to 1.

For the 3-input system under consideration, the number of possible input combinations is 23, or 8. This means that the system has a total of 8 minterms as follows:

m0 = x’ y’ z’ =1	IFF **xyz** = **000**, otherwise it equals 0
m1 = x’ y’z=1	IFF **xyz** = **001**, otherwise it equals 0
m2 = **x’yz’** =1	IFF **xyz** = **010**, otherwise it equals 0
m3 = **x’yz**=1	IFF **xyz** = **011**, otherwise it equals 0
m4 = x y’ z’ =1	IFF **xyz** = **100**, otherwise it equals 0
m5 = **x y’z**=1	IFF **xyz** = **101**, otherwise it equals 0
m6 = **xy’z’** =1	IFF **xyz** = **110**, otherwise it equals 0
m7 = **xyz** =1	IFF **xyz** = **111**, otherwise it equals 0

In general,

For *n*-input variables, the number of minterms = the total number of possible input combinations = 2^n.

A minterm = 0 at all input combinations except one where the minterm = 1.

3.6 MaxTerm

Consider a circuit of 3 input signals (variables) X, Y, & Z.A term which ORs all input variables, either in the true or complement form, is called a Maxterm.

With 3-input variables, the system under consideration has a total of 8 Maxterms, namely:

$(X + Y + Z)$,$(X + Y + Z')$, $(X + Y' + Z)$,$(X + Y' + Z')$, $(X' + Y + Z)$,$(X' + Y + Z')$, $(X' + Y' + Z)$ & $(X' + Y' + Z')$

Each Maxterm equals 0 at exactly one of the 8 possible input combinations and is equal to 1 at all other combinations.

For example, **(*x* + *y* + *z*)** equals 1 at all input combinations except for the combination ***xyz*** = **000**, where it is equal to 0.

Accordingly, the Maxterm**(*x* + *y* + *z*)** is referred to as *M0*.

In general, Maxterms are designated *Mi*, where *i* corresponds to the input combination at which this Maxterm is equal to 0.

For the 3-input system, the number of possible input combinations is 23, or 8.

This means that the system has a total of 8 Maxterms as follows:

M0 = **(x + y + z)** =0 IFF **xyz** = **000**, otherwise it equals 1
M1 = **(x + y + z')** = 0 IFF **xyz** = **001**, otherwise it equals 1
M2 = **(x + y' + z)** = 0 IFF **xyz** = **010**, otherwise it equals 1
M3 = **(x + y' + z')** = 0 IFF **xyz** = **011**, otherwise it equals 1
M4 = **(x' + y + z)** = 0 IFF **xyz** = **100**, otherwise it equals 1
M5 = **(x' + y + z')** = 0 IFF **xyz** = **101**, otherwise it equals 1
M6 = **(x' + y' + z)** = 0 IFF **xyz** = **110**, otherwise it equals 1
M7 = **(x' + y' + z')** = 0 IFF **xyz** = **111**, otherwise it equals 1

Self Assessment Questions

What is the difference between the maxterm and minterm

Self Assessment Answers

The maxterm deals with OR's operation while the Minterm deals with AND's operation

In general,

For *n*-input variables, the number of Maxterms = the total number of possible input combinations = $2^{n.}$

A Maxterm = 1 at all input combinations except one where the Maxterm = 0.

e.g Given the truth table below

x y z	f(x,y,z)	f'(x,y,z)
0 0 0	1	0
0 0 1	1	0
0 1 0	1	0
0 1 1	1	0
1 0 0	0	1

1 0 1	0	1
1 1 0	1	0
1 1 1	0	1

Sum of product

F =xyz+xyz’+xy’z+xy’z’+x’y’z’

= $m_0+m_1+m_2+m_3+m_6$

= $\sum m(0,1,2,3,6)$

For product of sum

f = (x’ + y + z)(x’ + y + z’)(x’ + y’ + z’)

= $M_4\ M_5\ M_7$

= $\Pi M(4,5,7)$

For sum of product

F’= (x’yz+x’yz’+x’y’z’)

=$m_4+m_5+m_7$

= $\sum m(4,5,7)$

f’ = (x + y + z)(x + y + z’)(x + y’ + z)(x + y’ + z’)(x’ + y’ + z)

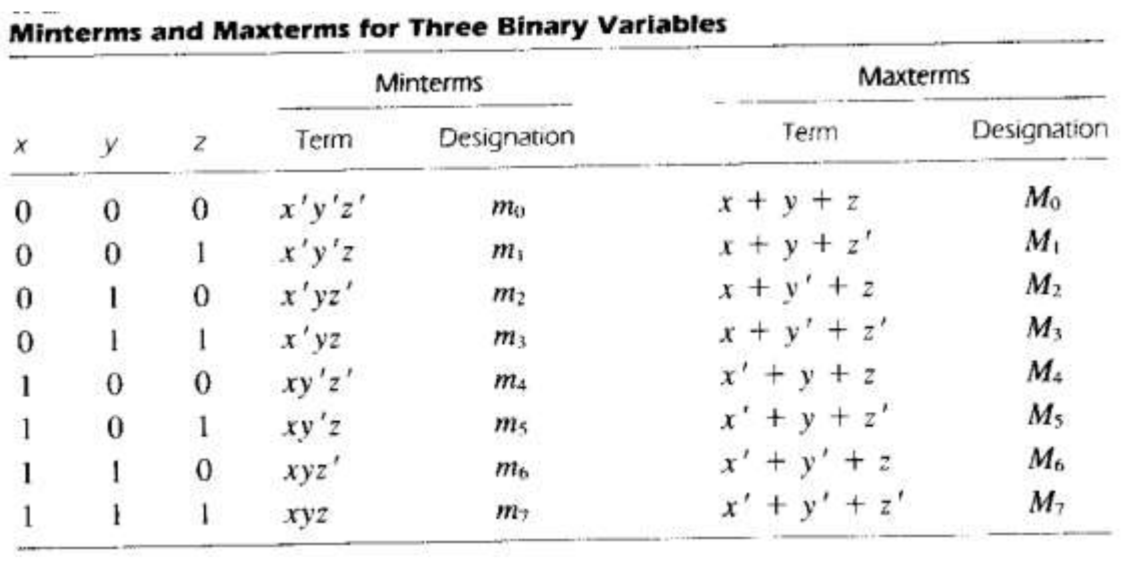
= $M_0\ M_1\ M_2\ M_3\ M_6$

= $\Pi M(0,1,2,3,6)$

f’ contains all the maxterms not in f

Minterms and maxterms are related

• Any minterm mi is the **complement** of the corresponding maxtermMi as shown in the table below



Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Self Assessment Questions

1. Distinguish between Minterms and Maxterms
2. Obtain the truth table and find the SOP and POS

(a) $(XY + Z)(Y + XZ)$

(b) $(\overline{A} + B)(\overline{B} + C)$

2. **(c)** $WX\overline{Y} + WX\overline{Z} + WXZ + Y\overline{Z}$

Self Assessment Answers

Example:

Given that **F (a, b, c, d) = Σ(0, 1, 2, 4, 5, 7)**, derive the product of maxtermsexpression of **F** and the 2 standard form expressions of **F'**.

Solution

Since the system has 4 input variables (a, b, c & d) The number of minterms andMaxterms = 2^4= 16

F (a, b, c, d) = Σ(0, 1, 2, 4, 5, 7)

List all maxterms in the Product of maxterms expression

F = Π (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15).

Cross out maxterms corresponding to input combinations of the minterms appearing in the sum of minterms expression

F = Π (~~0, 1, 2,~~ 3, ~~4, 5,~~ 6, ~~7~~, 8, 9, 10, 11, 12, 13, 14, 15).

F = Π (3, 6, 8, 9, 10, 11, 12, 13, 14, 15).

Similarly, obtain both canonical form expressions for F'

F' = Σ (3, 6, 8, 9, 10, 11, 12, 13, 14, 15).

F' = Π (0, 1, 2, 4, 5, 7)

Canonical Forms: The sum of minterms and the product of maxterms forms of Boolean expressions are known as the canonical forms of a function.

Converting between standard forms

• We can convert a sum of minterms to a product of maxterms

From before f = ∑m(0,1,2,3,6) and

f' = ∑m(4,5,7)

= m4 + m5 + m7

complementing (f')' = (m4 + m5 + m7)'

so f = m4' m5' m7' [DeMorgan's law]

= M4 M5 M7 [from the table on previous page]

= ΠM(4,5,7)

In general, just replace the minterms with maxterms, using maxterm numbers that don't appear in the sum of minterms: e.g.

f = Σm(0,1,2,3,6)

= ΠM(4,5,7)

The same thing works for converting from a product of maxterms to a sum of minterms

4.0 Conclusion

This unit discussed sum of product (SOP), product of sum (POS) ,minterm, maxterm and their standard form used in boolean expression

5.0 Summary

You have learnt:

A product term is a term with ANDed literals*. Thus, AB, A’B, A’CD are all product terms.

A minterm is a special case of a product term where all input variables appear in the product term either in the true or complement form.

A sum term is a term with ORed literals*. Thus, (A+B), (A’+B), (A’+C+D) are all sum terms.

A maxterm is a special case of a sum term where all input variables, either in the true or complement form, are ORed together.

Boolean functions can generally be expressed in the form of a Sum of Products (SOP) or in the form of a Product of Sums (POS).

The sum of minterms form is a special case of the SOP form where all product terms are minterms.

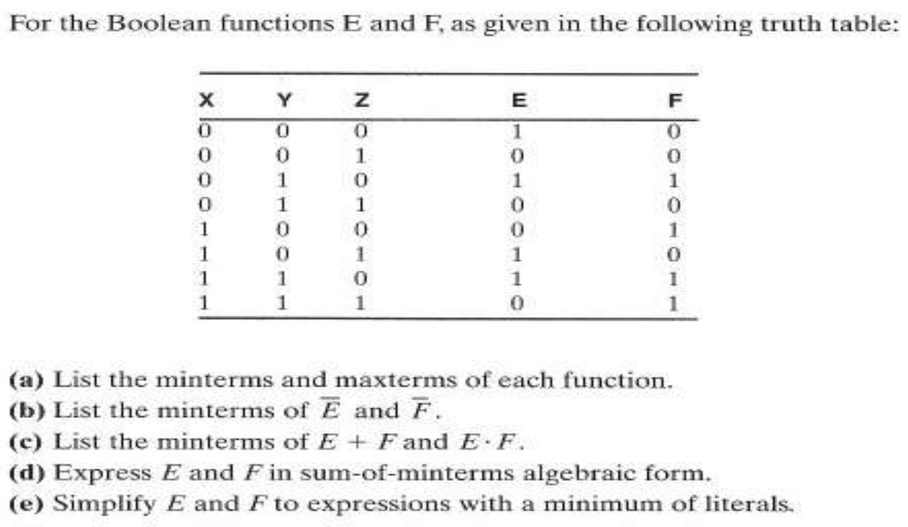
The product of maxterms form is a special case of the POS form where all sum terms are maxterms.

The SOP and POS forms are Standard forms for representing Boolean functions.

6.0 Tutor-Marked Assignment

1. Develop a truth table for the standard SOP expression A’ B’C + AB’ C’ + ABC.

2.



For the Boolean functions E and F, as given in the following truth table:

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

(a) List the minterms and maxterms of each function.
(b) List the minterms of $\overline{E}$ and $\overline{F}$.
(c) List the minterms of $E + F$ and $E \cdot F$.
(d) Express E and F in sum-of-minterms algebraic form.
(e) Simplify E and F to expressions with a minimum of literals.

7.0 References/Further Reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,

Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.

Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall

Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

http://drstienecker.com/tech-332/3-logic-circuits-boolean-algebra-and-truth-tables/

http://www.courses.ebe.uct.ac.za/eee317w/1.%20Basic%20Logic%20Design.pdf

http://ozark.hendrix.edu/~burch/cs/230/cso/ch07-gates/index.html

http://www.indiabix.com/digital-electronics/combinational-logic-circuits/16006

Unit 4

Karnaugh Map

Content

1.0 Introduction

Even though **Boolean expressions** can be simplified by algebraic manipulation, such an approach lacks clear regular rules for each succeeding step and it is difficult to determine whether the **simplest expression** has been achieved.

In contrast, Karnaugh map (K-map) method provides a straightforward procedure for simplifying Boolean functions. K-maps of up to 4 variables are very common to uses. Simplified expressions produced by K-maps are always either in the **SOP** or the **POS** form. The map provides the same information contained in a Truth Table but in a different format.

2.0 Learning Outcomes

At the end of this unit you should be able to:

Build a 2, 3, or 4 variables K-map.
Minimize Sum of Product(SOP) function using K-maps
Differentiate between prime implicants and essential prime implicants.
Minimize Sum of Product (SOP) andProduct of Sum (POS) function using a K-map.
Minimize a combinational circuit that is not completely specified (has don't care conditions).
Construct a 5 and 6 variable K-map given a truth table or a SOP representation.

3.1 Karnaugh Map (K Map)

A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression. As you have seen, the effectiveness of algebraic simplification depends on your familiarity with all the laws, rules, and theorems of Boolean algebra and on your ability to apply them. The Karnaugh map, on the other hand, provides a "cookbook" method for simplification.

A Kamaugh map i similar to a tnJth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. The cells are ananged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four. and five variables, but we will discuss only 3-variable and 4-variable situations to illustrate the principles.

The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is 2^3 = 8. For four variables, the number of cells is 2^4 = 16.

3.2 Code Distance

The distance between two binary code-words is the number of bit positions in which the two code-words have different values. For example, the distance between the code words **1001** and **0001** is **1** while the distance between the code-words **0011** and **0100** is **3**.

This definition of code distance is commonly known as the **Hamming distance** between two codes.

What is the significance of the K-Map

A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression.

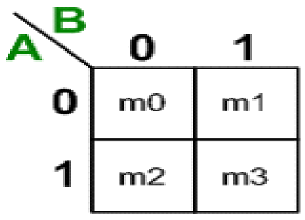
3.3 Two-Variable K-Maps

The 2-variable map is a table of 2 rows by 2 columns. The 2 rows represent the two values of the first input variable A, while the two columns represent the two values of the second input variable B.

Thus, all entries (squares) in the first row correspond to input variable A=0, while entries (squares) of the second row correspond to A=1.

Likewise, all entries of the first column correspond to input variable B = 0, while entries of the second column correspond to B=1.

Thus, each map entry (or square) corresponds to a unique value for the input variables A and B.



For example, the top left square corresponds to input combination AB=00. In other words, this square represents mintermm$_{0}$. Likewise, the top right square corresponds to input combination AB=01, or mintermm$_{1}$and the bottom left square corresponds to input combination AB=10, or mintermm$_{2}$. Finally, the bottom right square corresponds to input combination AB=11, or mintermm$_{3}$.

In general, each map entry (or square) corresponds to a particular input combination (or minterm).

Since, Boolean functions of two-variables have four minterms, a 2-variable K-map can represent any 2-variable function by plugging the function value for each input combination in the corresponding square.

Definitions/Notations:

Two K-map squares are considered **adjacent** if the input codes they represent have a Hamming distance of 1.

A K-map square with a function value of 1 will be referred to as a **1-Square**.

A K-map square with a function value of 0 will be referred to as a **0-Square**.

The simplification procedure is summarized below:

Step 1: Draw the map according to the number of input variables of the function.

Step 2: Fill "1's" in the squares for which the function is true.

Step 3: Form as big group of **adjacent 1-squares** as possible. There are some rules for this which you will learn with bigger maps.

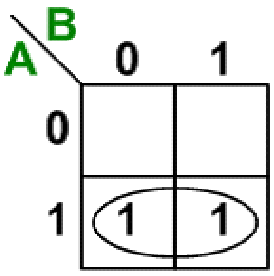
Step 4: Find the common literals for each group and write the simplified expression in SOP.

Example:

Consider the given truth table of two variable functions. Obtain the simplified function using K-map.

A	B	F
0	0	0
0	1	0
1	0	1
1	1	1

First draw a 2-variable K-map. The function F is true when AB' (m_2) is true and when AB (m_3) is true, so a 1 is placed inside the square that belongs to m_2 and a 1 is placed inside the square that belongs to m_3.



Since both of the 1-squares have different values for variable B but the same value for variable A, which is 1, i.e., wherever A = 1 then F = 1 thus F = A.

This simplification is justified by algebraic manipulation as follows:

$F = m_2 + m_3 = AB' + AB = A (B' + B) = A$

To understand how combining squares simplifies Boolean functions, the basic property possessed by the **adjacent squares** must be recognized.

In the above example, the two 1-squares are adjacent with the same value for variable A (A=1) but different values for variable **B** (one square has B=0, while the other has B=1).

This reduction is possible since both squares are adjacent and the net expression is that of the common variable (A).

Generally, this is true for any 2 codes of Hamming distance 1 (adjacent). For an n-variable K-map, let the codes of two **adjacent squares** (distance of 1) have the same value for all variables except the ith variable. Thus,

Code of 1st Square: $X_1, X_2 \ldots, X_{i-1}, X_i, X_{i+1} \ldots\ldots Xn$

Code of 2nd Square: $X_1, X_2 \ldots, X_{i-1}, X_i, X_{i+1} \ldots\ldots Xn$

Combining these two squares in a group will eliminate the different variable $\mathbf{X_i}$ and the combined expression will be

$X_1, X_2 \ldots, X_{i-1}, X_i, X_{i+1} \ldots\ldots Xn$

since:

$=(X_2 \ldots, X_{i-1}, X_i, X_{i+1} \ldots\ldots Xn) + (X_1, X_2 \ldots, X_{i-1}, X_i, X_{i+1} \ldots\ldots Xn)$

$=(X_1, X_2 \ldots, X_{i-1}, X_{i+1} \ldots\ldots Xn)(X_i + X_i,)$

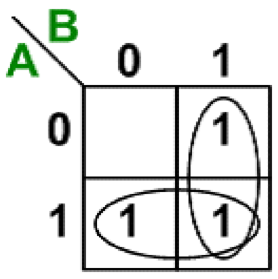
==(X_1, X_2 …, X_{i-1}, X_{i+1}……Xn)

The variable in difference is dropped.

Another Example:

Simplify the given function using K-map method:

F = Σ (1, 2, 3)



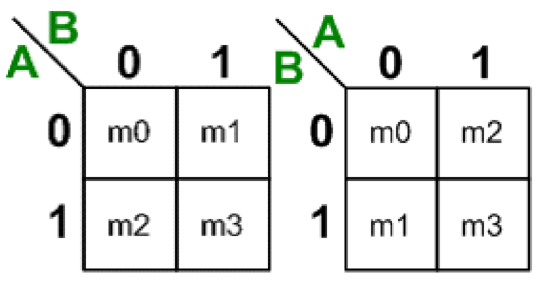
In this example:

$F = m_1 + m_2 + m_3 = m_1 + m_2 + (m_3 + m_3)$

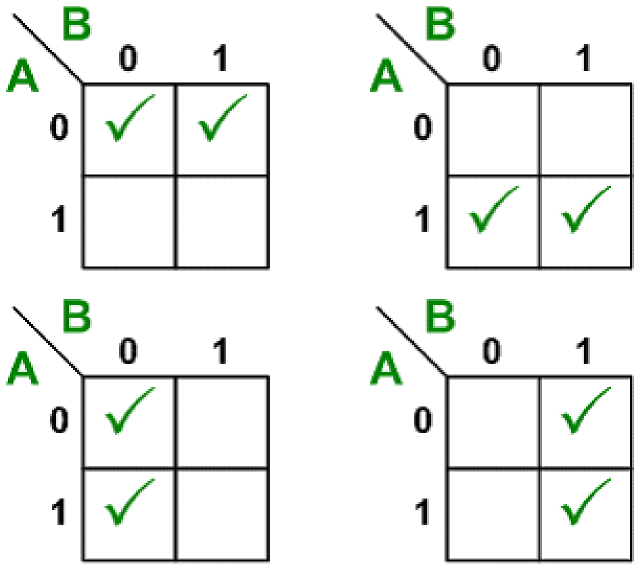
$F = (m_1 + m_3) + (m_2 + m_3) = A + B$

Rule: A 1-square can be member of more than one group.

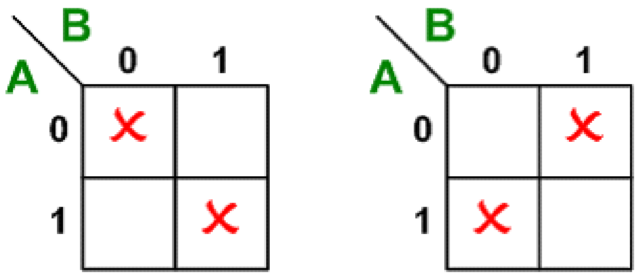
If we exchange the places of **A** and **B**, then minterm positions will also change. Thus, **m_1** and **m_2** will be exchanged as well.



In an n-variable map each square is **adjacent** to "**n**" other squares, e.g., in a 2-variable map each square is adjacent to two other squares as shown below:



Examples of non-adjacent squares are shown below:



Self Assessment Questions

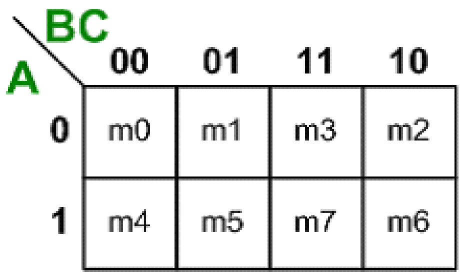
How many terms does a 2 variable K- map contains?

Self Assessment Answers

A 2 variable K-Map contains 4 element

3.4 Three-Variable K-Maps:

There are eight minterms for a Boolean function with three-variables. Hence, a **three-variable** map consists of **8 squares**.



A \ BC	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

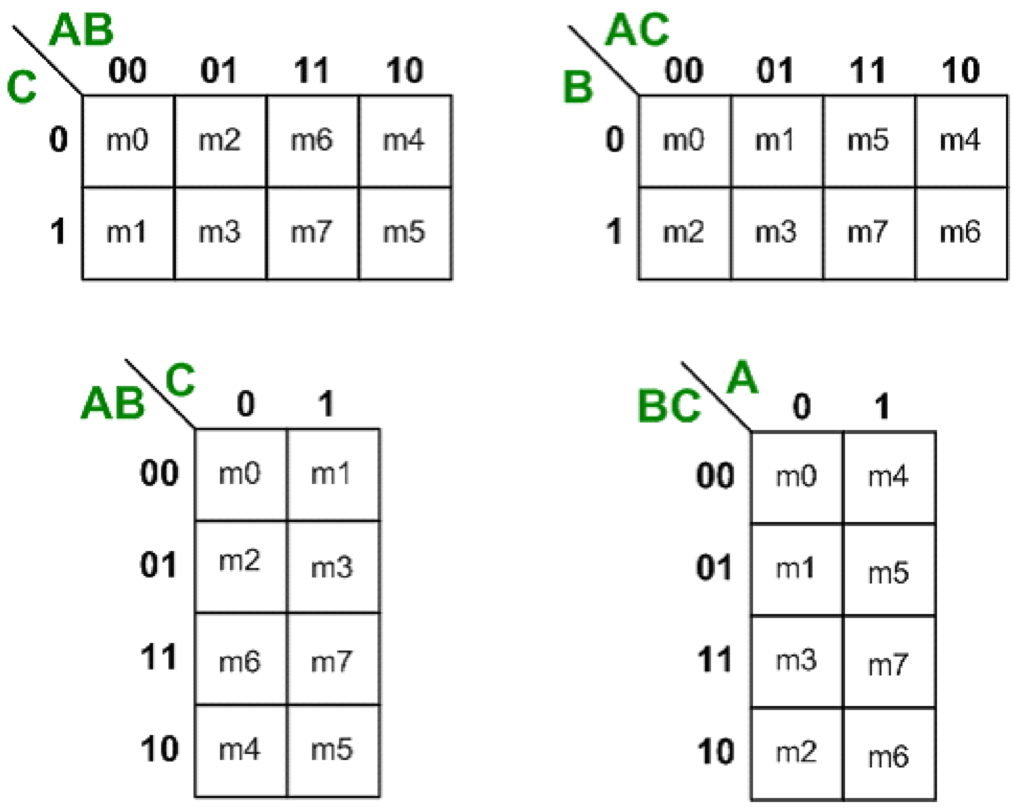
All entries (squares) in the first row correspond to input variable A=0, while entries (squares) of the second row correspond to A=1.

Likewise, all entries of the first column correspond to input variable B = 0, C = 0, all entries of the second column correspond to input variable B = 0, C = 1, all entries of the third column correspond to input variable B = 1, C = 1, while entries of the fourth column correspond to B=1, C = 0.

To maintain adjacent columns physically adjacent on the map, the column coordinates do not follow the binary count sequence. This choice yields unit distance between codes of one column to the next (00 – 01—11 – 10), like **Gray Code**.

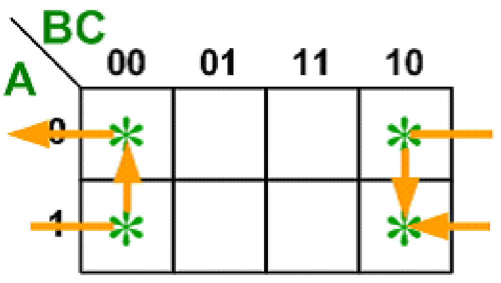
Variations of Three-Variable Map:

The figure shows variations of the three-variable map. Note that the minterm corresponding to each square can be obtained by substituting the values of variables **ABC** in order.



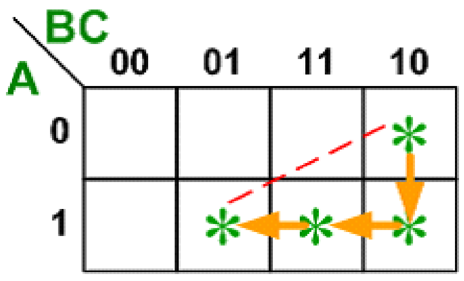
There are cases where two squares in the map are considered to be adjacent even though they do not physically touch each other.

In the figure of 3-variable map, m_0 is adjacent to m_2 and m_4 is adjacent to m_6 because the minterms differ by only one variable. This can be verified algebraically: **$m_0 + m_2 = A'B'C' + A'BC' = A'C' (B' + B) = A'C'$ $m_4 + m_6 = AB'C' + ABC' = AC' (B' + B) = AC'$**



Rule: Groups may only consist of **2**, **4**, **8**, **16**,… squares (always power of 2). For example, groups may not consist of 3, 6 or 12 squares.

Rule: Members of a group must have a closed loop adjacency, i.e., **L-Shaped 4** squares do not form a valid group.



Notes:

1. Each square is adjacent to 3 other squares.

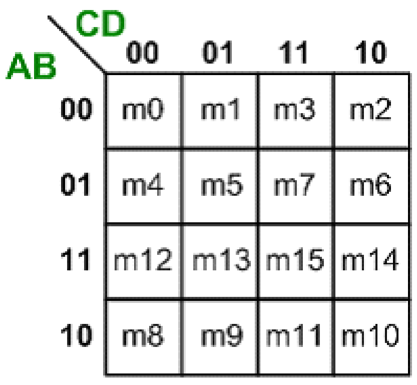
2. One square is represented by a minterm (i.e. a product term containing all 3 literals).

3. A group of 2 adjacent squares is represented by a product term containing only 2 literals, i.e., 1 literal is dropped.

4. A group of 4 adjacent squares is represented by a product term containing only 1 literal, i.e., 2 literals are dropped.

3.5 Four-Variable K-Maps:

There are **16 minterms** for a Boolean function with **four-variables**. Hence, four- variable map consists of 16 squares.



AB \ CD	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

Notes:

Each square is **adjacent** to **4** other squares.

One square is represented by a minterm (a product of all 4-literals).

Combining **2** squares drops **1**-literal.

Combining **4** squares drops **2**-literals.

Combining **8** squares drops **3**-literals.

1. 2. 3. 4. 5.

Rule: The combination of squares that can be chosen during the simplification process in the **n-variable** map are as follows:

A group of $\mathbf{2^n}$ **squares** produces a function that always equal to **logic 1**.

A group of $\mathbf{2^{n-1}}$ **squares** represents a product term of **one literal**.

A group of $\mathbf{2^{n-2}}$ **squares** represents a product term of **two literals** and so on.

One square represents a minterm of **n literals**.

3.6 Prime Implicant

A product term of a function is said to be an **implicant**.

A **Prime Implicant(PI)** is a product term obtained by combining the maximum possible number of adjacent 1-squares in the map.

If a minterm is covered only by one prime implicant then this prime implicant is said to be an **Essential Prime Implicant(EPI)**.

POS Simplification:

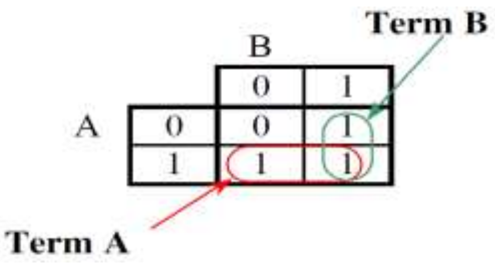
Until now we have derived simplified Boolean functions from the maps in SOP form. Procedure for deriving simplified Boolean functions **POS** is slightly different. Instead of making groups of **1's**, make the groups of **0's**.

Since the simplified expression obtained by making group of 1's of the function (say F) is always in SOP form. Then the simplified function obtained by making group of 0's of the function will be the complement of the function (i.e., F') in SOP form.

Applying DeMorgan's theorem to F' (in SOP) will give F in POS form.

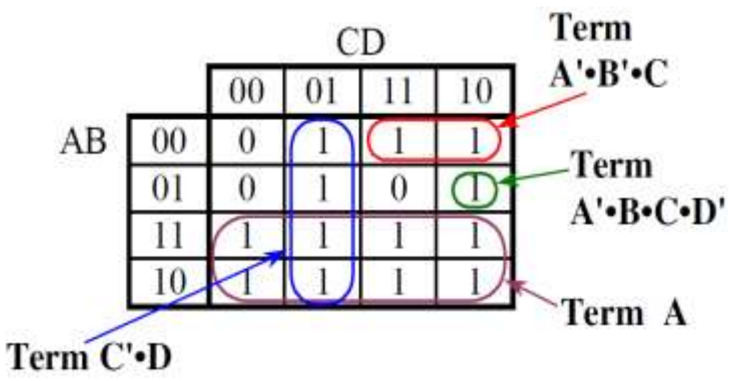
Examples 1: X = A + B

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



To do this we indicate simplifications by circling the adjacent ones as shown in Kmap above. Each circle will represent one term in the expression (NB we use the term circle loosely here to mean a closed boundary around adjacent squares on the Karnaugh map). The fact that a 1 in the Karnaugh map may appear in more than one circle is irrelevant. In K map above we see that the last row circled together is independent of B, and corresponds to the term A, and similarly the last column corresponds to the term B in the simplest form.

Figure below shows a four by four Karnaugh map, and indicates some possible circles on it. To find the term that corresponds to a circle we find the inputs that do not vary in that circle. The circle covering the second column is defined by C = 0, D = 1. The inputs A and B can be either 1 or 0 within the circle. To make sure that the output is a 1 at all four input values within that circle we simply need ensure that C'. D = 1. Notice that we always try to find the largest possible circle since this will correspond to the greatest simplification. In Figure 2 we could find a greater simplification by circling together all the ones in the fourth column rather than the single one corresponding to input 0110.



Don't Care Conditions:

In some cases, the function is not specified for certain combinations of input variables as 1 or 0.

There are two cases in which it occurs:

1. The input combination never occurs.

2. The input combination occurs but we do not care what the outputs are in response to these inputs.

In both cases, the outputs are called as **unspecified** and the functions having them are called as incompletely specified functions.

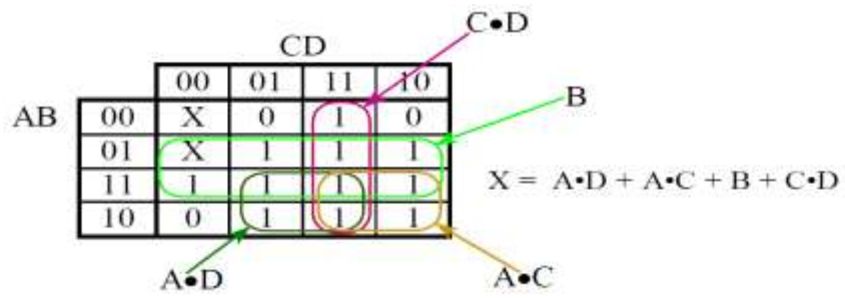
In most applications, we simply do not care what value is assumed by the function for unspecified minterms.

Unspecified minterms of a function are called as **don't care conditions**. They provide further simplification of the function, and they are denoted by X's to distinguish them from 1's and 0's.

In choosing adjacent squares to simplify the function in a map, the don't care minterms can be assumed either 1 or 0, depending on which combination gives the simplest expression.

A don't care minterm need not be chosen at all if it does not contribute to produce a larger implicant.

For the Kmap below



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

We can make these explicit in the truth table by placing a cross in the output column rather than a zero or one. This has the advantage in minimisation problems that the corresponding points in the Karnaugh map may be treated as either a 1 or a 0 for the purpose of simplification. For example, considering again the four input majority circuit, if we happen to know the input states 0000 and 0100 never occur we can treat them as "don't cares" in the Karnaugh map as shown in Figure . Clearly it is advantageous to treat 0100 as a 1 then we can obtain a major simplification with the central block of eight which corresponds to the term B in the second row.

Examples:

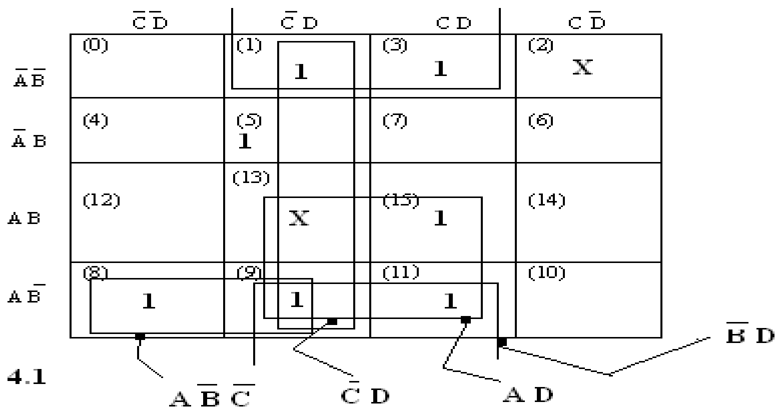
1.Minimize the following logic function using K-maps .

F(A,B,C,D) = ∑m(1,3,5,8,9,11,15) + d(2,13)

Soln

Ans:

Minimization of the logic function F(A, B, C, D) = ∑ m(1,3,5,8,9,11.15) + d(2,13) using Karnaugh Map for the logic function is given in table below



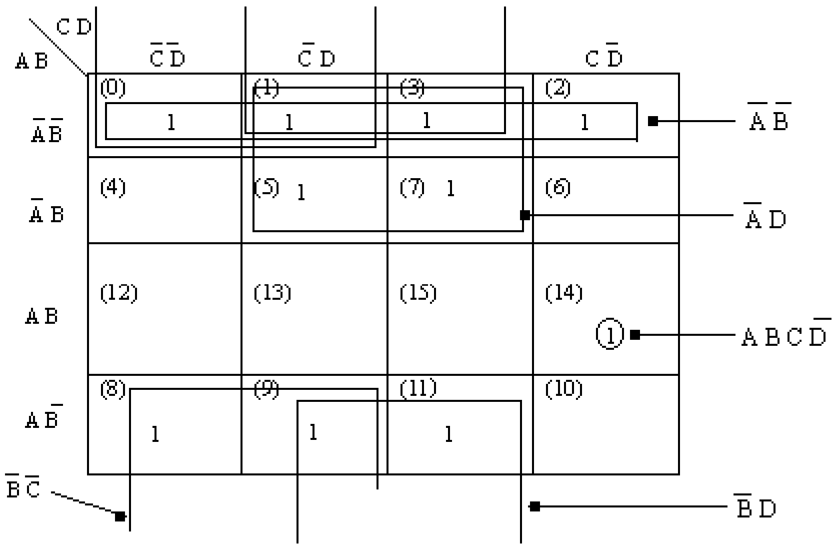
4.1

The minimized logic expression in SOP form is F = A *B’ C’* + *C’* D + *B’* D + AD

The minimized logic expression in POS form is F = (A + *B’*+*C’*) (*C’*+D) (*B’*+D) (A+D)

Minimize the logic functionY**(**A,B,C,D**)** =∑m**(**0,1,2,3,5,7,8,9,11,14**)** . Use Karnaugh map. Draw logic circuit for the simplified function.

The figure below shows the Karnaugh map. Since the expression has 4 variables, the map has 16 cells. The digit 1 has been written in the cells having a term in the given expression. The decimal number has been added as subscript to indicate the binary number for the concerned cell. The term *ABC D* cannot be combined with any other cell. So this term will appear as such in the final expression. There are four groupings of 4 cells each. These correspond to the min terms (0, 1, 2, 3), (0, 1, 8, 9), (1, 3,5,7) and (1, 3, 9, 11). These are shown in the map. Since all the terms (except 14) have been included in groups of 4 cells, there is no need to form groups of two cells.



The simplified expression is *Y* (A,*B*,C,*D) = ABC D' + A' B' + B' C' + B'* D+ *A'*D

Self Assessment Questions

Simplify the following expression into sum of products using Karnaugh map
F(A,B,C,D) =Σ(1,3,4,5,6,7,9,12,13)

Self Assessment Questions

4.0 Conclusion

This unit considered the use of K map in simplifyiny logic expressions. 2 variable, 3 variable and 4 variable K map were also considered.

5.0 Summary

In this unit the following aspects have been discussed:

Complement, the inverse or opposite of a number. In Boolean algebra, the inverse function, expressed with a bar over a variable. The complement of a I is O. and vice versa.

"Don't care" A combination of input literals that cannot occur and can be used as a I or a 0 on a Karnaugh map for simplification.
Karnaugh map An arrangement of cells representing the combinations of literals in a Boolean expression and used for a systematic simplification of the expression.
Minimization: The process that results in an SOP or POS Boolean expression that contains the fewest possible literals per term.

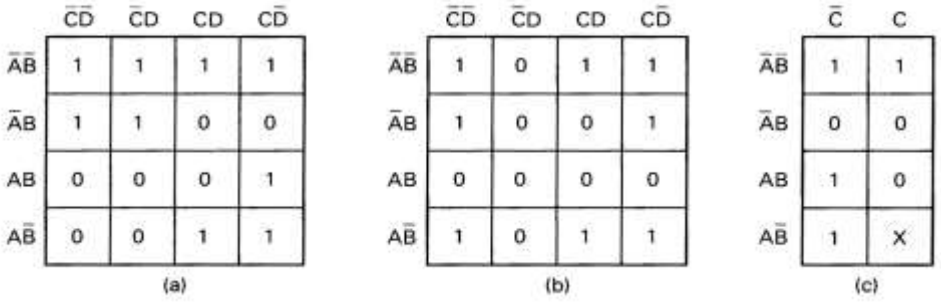
6.0 Tutor-Marked Assignment

1. The circuit is to have three inputs and one output. One of the inputs is specified as a control input (we shall designate it C). When this control input is at the logical 0 value, the output is the logical (or Boolean) AND function of the other two inputs. When C is at the logical 1 level, the output is equal to the Boolean OR function of the other two inputs.

2. You are to design a circuit that has three inputs A B C, and one output R. The output R should be 1 if exactly one or exactly two of the inputs are 1. Fill up the truth table for the circuit, then write down the canonical form of the boolean equation using a sum of minterms. Simplify the resulting expression as far as you can.

3. Write down the equation for the circuit using maxterms. Verify that the result is the same as your solution to Q2

4. Determine the minimum expression for each of the K map below



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	1	1	1
$\overline{A}B$	1	1	0	0
AB	0	0	0	1
$A\overline{B}$	0	0	1	1

(a)

	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	0	1	1
$\overline{A}B$	1	0	0	1
AB	0	0	0	0
$A\overline{B}$	1	0	1	1

(b)

	$\overline{C}$	C
$\overline{A}\,\overline{B}$	1	1
$\overline{A}B$	0	0
AB	1	0
$A\overline{B}$	1	X

(c)

Use a Karnaugh map to find the minimum SOP form for each expression:
(a) A'B'C' + A'B'C + AB'C
(c) AC(B' + C)

Optimise the following Boolean function by means of a three variable map

(a) $F(X,Y,Z) = \Sigma m(1,3,6,7)$

(b) $F(X,Y,Z) = \Sigma m(3,5,6,7)$

(c) $F(A,B,C) = \Sigma m(0,1,2,4,6)$

(d) $F(A,B,C) = \Sigma m(0,3,4,5,7)$

REFERENCES

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,

Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.

Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall

Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

http://www.ccse.kfupm.edu.sa/~amin/eCOE200/Lesson4_4.pdf

www.doc.ic.ac.uk/~dfg/hardware/HardwareLecture03.pd

www.doc.ic.ac.uk/~dfg/hardware/HardwareHandout02.pdf

MODULE 3

Combinational Logic

Unit 1

Adder and Subtractor

Content

1.0 Introduction

When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is in the category of combinational logic. In combinational logic, the output level is at all times dependent on the combination of input levels. This module expands on the material introduced in earlier chapters with coverage of the analysis, design, and troubleshooting of various combinational logic circuits. Specifically this unit covers adders and subtractors.

2.0 Learning Outcomes

At the end of this unit you should be able to

Differentiate between Combinational versus sequential circuits
Know Binary half adder and full adder and describe their functions
Differentiate between half adder and full adder
Know half subtractor and full subtractor and describe their functions
Differentiate between half and full subtractor
Know rhe circuit implementation of adder and subtractor

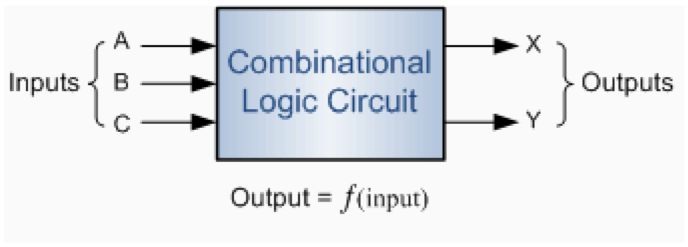
3.0 Learning Contents

3.1 Combinational versus. Sequential circuit

Digital circuits may be classified as combinational or sequential. In a combinational circuit, the present outputs depend only on present inputs (subject to reaction times). In a sequential circuit, the present outputs may also depend on past outputs and inputs. Sequential circuits usually contain combinational subcircuits. The two classes of circuits have different topologies. Sequential circuitscontain feedback paths from the outputs to the inputs, while combinationalcircuits do not.

Combinational Logic Circuits

Unlike **Sequential Logic Circuits** whose outputs are dependant on both their present inputs and their previous output state giving them some form of **Memory**, the outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time as they have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependant at all times on the combination of its inputs and if one of its inputs condition changes state so does the output as combinational circuits have "no memory", "timing" or "feedback loops".



Combinational Logic Circuits are made up from basic logic **NAND, NOR or NOT** gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.

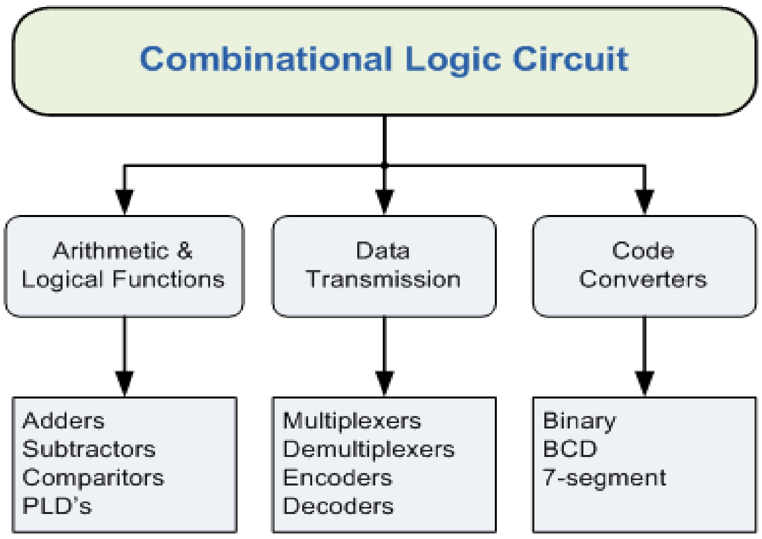
Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.

As combinational logic circuits are made up from individual logic gates only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application include **Multiplexers**, **De-multiplexers**, **Encoders**, **Decoders**, **Full** and **Half Adders** etc.

What is the fundamental difference between a Combinational Logic and the Sequential Logic

The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time as compared to the**Sequential Logic Circuits** whose outputs are dependant on both their present inputs and their previous output state giving them some form of **Memory**.

Classification of Combinational Logic


Combinational Logic Circuit
Arithmetic & Logical Functions
Data Transmission
Code Converters
Adders Subtractors Comparitors PLD's
Multiplexers Demultiplexers Encoders Decoders
Binary BCD 7-segment

One of the most common uses of combinational logic is in **Multiplexer** and **De-multiplexer** type circuits. Here, multiple inputs or outputs are connected to a common signal line and logic gates are used to decode an address to select a single data input or output switch. A multiplexer consist of two separate components, a logic decoder and some solid state switches, but before we can discuss multiplexers, decoders and de-multiplexers in more detail we first need to understand how these devices use these "solid state switches" in their design.

3.2 The Binary Adder

A common and very useful combinational logic circuit which can be constructed using just a few basic logic gates and adds together binary numbers is the **Binary Adder** circuit. The Binary Adder is made up from standard AND and Ex-OR gates and allow us to "add" together single bit binary numbers, a and b to produce two outputs, the SUM of the addition and a CARRY called the Carry-out, (**Cout**) bit. One of the main uses for the **Binary Adder** is in arithmetic and counting circuits.

Consider the addition of two denary (base 10) numbers below.

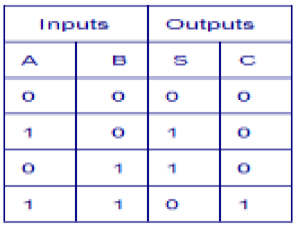
123	A	(Augend)
+ 789	B	(Addend)
912	SUM	

Each column is added together starting from the right hand side and each digit has a weighted value depending upon its position in the columns. As each column is added together a carry is generated if the result is greater or equal to ten, the base number. This carry is then added to the result of the addition of the next column to the left and so on, simple school math's addition. The adding of binary numbers is basically the same as that of adding decimal numbers but this time a carry is only generated when the result in any column is greater or equal to "2", the base number of binary.

Half Adder:

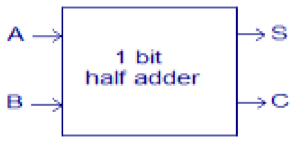
A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits. If A andB are the input bits, then sum bit (S) is the X-OR of A and B and the carry bit (C) will be the AND of A and B. From this it is clear that a half adder circuit can be easily constructed using one X-OR gate and one AND gate. Half adder is the simplest of all adder circuit, but it has a major disadvantage.

The half adder can add only two input bits (A and B) and has nothing to do with the carry if there is any in the input. So if the input to a half adder have a carry, then it will be neglected it and adds only the A and B bits. That means the binary addition process is not complete and that’s why it is called a half adder. The truth table, schematic representation and XOR//AND realization of a half adder are shown in the figure below.

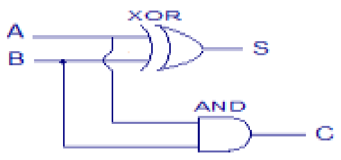


Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table

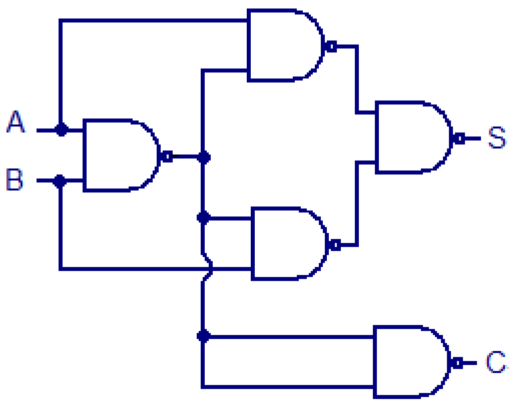


Schematic

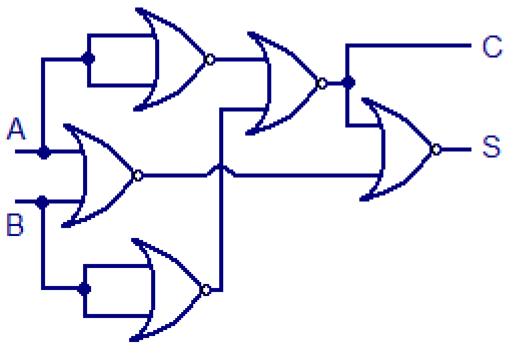


Realization

NAND gates or NOR gates can be used for realizing the half adder in universal logic and the relevant circuit diagrams are shown in the figure below



Half adder using NAND logic



Half adder using NOR logic

What are Combinational Logic made up of and their applications

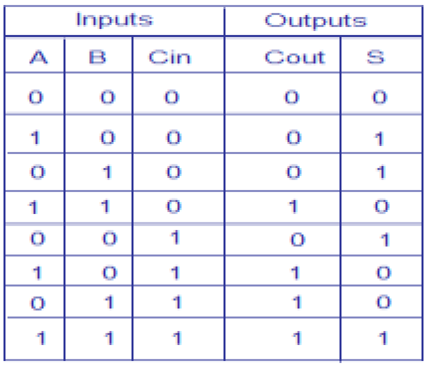
Comon combinational circuits made up from individual logic gates that carry out a desired application include Multiplexers, De-multiplexers, Encoders, Decoders, Full and Half Adders etc.

Full adder

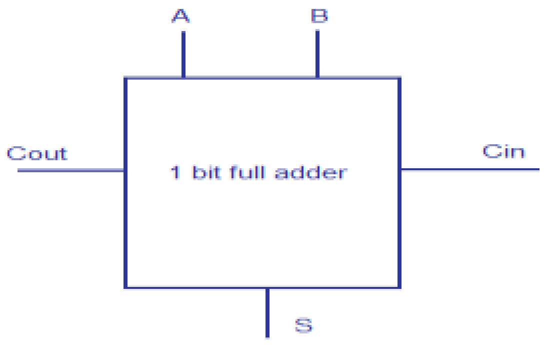
Full adder circuit adds three bit binary numbers (X,Y,Z) & outputs two nos. of one bit binary numbers, Sum & Carry . This type of adder is a little more difficult to implement than a half-adder. The main difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs. The first two inputs are A and B and the third input is an input carry designated as CIN. When a full adder logic is designed we will be able to string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

The output carry is designated as COUT and the normal output is designated as S

There is a simple trick to find results of a full adder. Consider the second last row of the truth table, here the operands are 1, 1, 0 ie (A, B, Cin). Add them together ie 1+1+0 = 10 . In binary system, the number order is 0, 1, 10, 11……. and so the result of 1+1+0 is 10 just like we get 1+1+0 =2 in decimal system. 2 in the decimal system corresponds to 10 in the binary system. Swapping the result "10" will give S=0 and Cout = 1 and the second last row is justified. This can be applied to any row in the table.



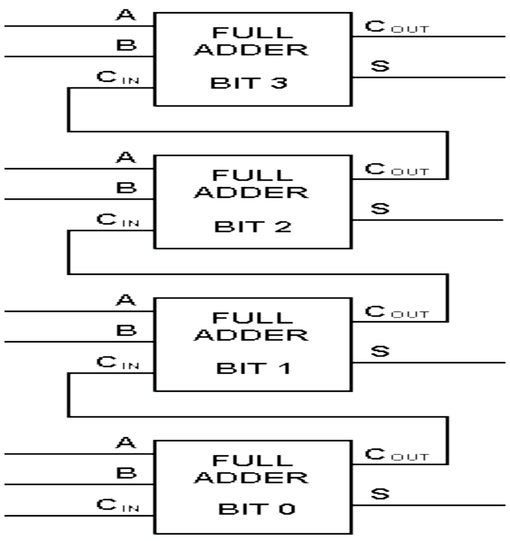
Inputs			Outputs	
A	B	Cin	Cout	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



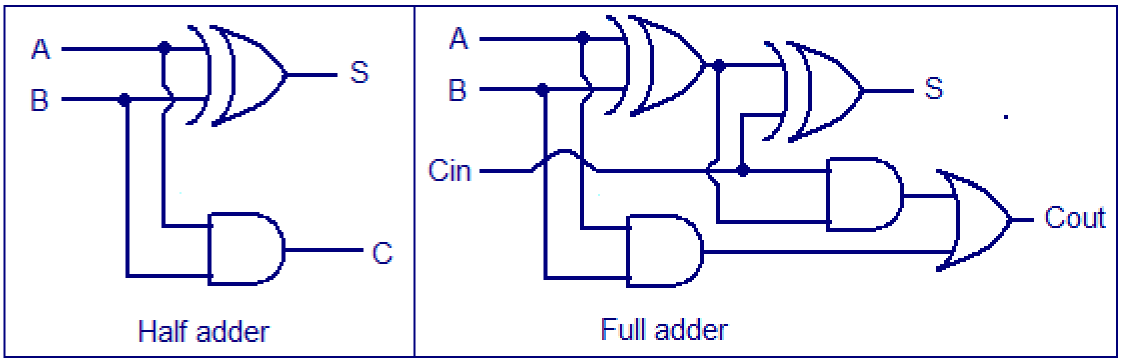
1 bit full adder truth table & schematic

With this type of schematic,

we can add two bits together taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude. In a computer, for a multi-bit operation, each bit must be represented by a full adder and must be added simultaneously. Thus, to add two 8-bit numbers, you will need 8 full adders which can be formed by cascading two of the 4-bit blocks. The addition of two 4-bit numbers is shown below.



A Full adder can be made by combining two half adder circuits together (a half adder is a circuit that adds two input bits and outputs a sum bit and a carry bit).



Self Assessment Questions

1. Describe the full Adder Circuit
2. What is the difference between Half and full adders

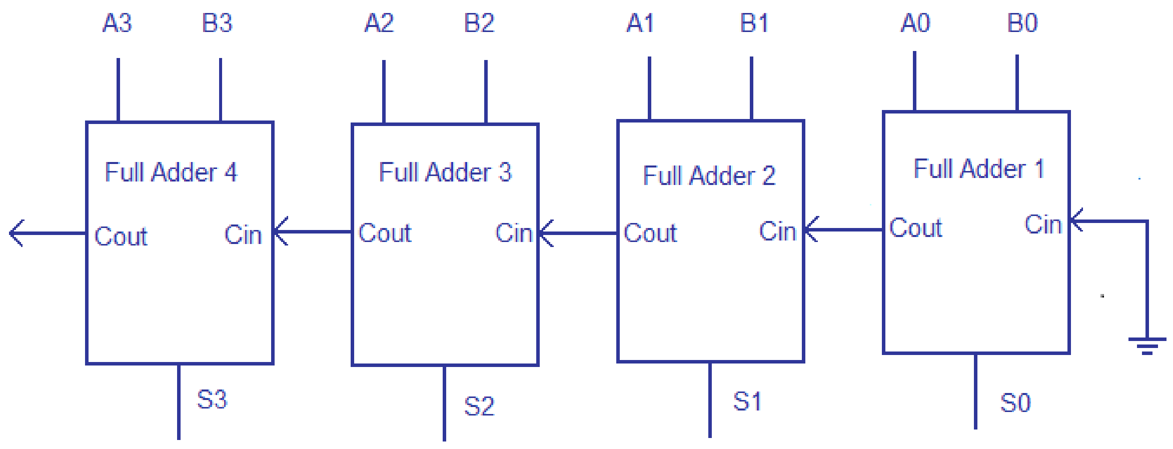
Self Assessment Questions

1. Full adder circuit adds three bit binary numbers (X,Y,Z) & outputs two nos. of one bit binary numbers, Sum & Carry .
2. The main difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs.

Realisea Full adder using NAND and NOR logic

Ripple Carry Adder Circuit:

Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N- bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs.Propagation delays inside the logic circuitry is the reason behind this. Propagation delay is time elapsed between the application of an input and occurance of the corresponding output. Consider a NOT gate, When the input is “0″ the output will be “1″ and vice versa. The time taken for the NOT gate’s output to become “0″ after the application of logic “1″ to the NOT gate’s input is the propagation delay here. Similarly the carry propagation delay is the time elapsed between the application of the carry in signal and the occurance of the carry out (Cout) signal. Circuit diagram of a 4-bit ripple carry adder is shown below.



Sum out S0 and carry out Cout of the Full Adder 1 is valid only after the propagation delay of Full Adder 1. In the same way, Sum out S3 of the Full Adder 4 is valid only after the joint propagation delays of Full Adder 1 to Full Adder 4. In simple words, the final result of the ripple carry adder is valid only after the joint propogation delays of all full adder circuits inside it.

3.3 Subtractor

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this method, the subtraction operation becomes an addition operation requiring full-adders for its machine implementation. It is possible to implement subtraction with logic circuits in a direct manner, as done with paper and pencil. By this method, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position. The fact that a 1 has been borrowed must be conveyed to the next higher pair of bits by means of a binary signal coming out (output) of a given stage and going into (input) the next higher stage. Just as there are half- and full-adders, there are half- and full-sub tractors.

Half Subtractor

A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. It also has an output tospecify if a 1 has been borrowed. Designate the minuend bit by x and the subtrahend bit by y. To perform $x — y$, we have to check the relative magnitudes of *x* and y. If *x* ³y, we have three possibilities: 0 — 0 = 0, 1 — 0 = 1, and 1-1=0, the result is called the **difference bit.** *If x < y,* we have 0-1, and it is necessary to borrow a 1 from the next higher stage. The 1 borrowed from the next higher stage adds 2 to the minuend bit, just as in the decimal system a borrow adds 10 to a minuend digit. With the minuend equal to **2**, the difference becomes 2-1 = 1.

The half-subtracted needs two outputs. One output generates the difference and will be designated by the symbol *D.* The second output, designated *B* for borrow, generates the binary signal that informs the next stage that a 1 has been borrowed. The truth table for the input-output relationships of a half-subtracter can now be derived as follows:

X	**Y**	**B**	**D**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

The output borrow B is a 0 as long as x³ *y.* It is a 1 for *x* = 0 and*y* = 1. The *D* output is the result of the arithmetic operation *2B + x — y.*

The Boolean functions for the two outputs of the halfsubtractor are derived directly from the truth table:

D = x ‘y + xy’

B = x’y

It is interesting to note that the logic for *D* is exactly the same as the logic for output *S* in the half-adder.

Full subtractor

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage. This circuit has three inputs and two outputs. The three inputs, *x, y,* and z, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, *D* and *B,* represent the difference and outputs borrow, respectively. The truth table for the circuit is

X	**Y**	**Z**	**B**	**D.**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The eight rows under the input variables designate all possible combinations of 1's and 0's that the binary variables may take. The 1's and O's for the output variables are determined from the subtraction of x — y — z. The combinations having input borrow z = 0 reduce to the same four conditions of the halfadder. For x = 0, y = 0, and z = 1, we have to borrow a 1 from the next stage, which makes B = 1 and adds 2 to x. Since 2 - 0 - 1 = 1, D = 1. For x = 0 and yz = 11, we need to borrow again, making B = 1 and x = 2. Since 2 - 1 - 1 = 0, D =0. For x = 1 and yz = 01, we have x –y-z = 0, which makes B = 0 and D = 0.Finally, for x = 1, y = 1, z = 1, we have to borrow 1, making B = 1 and x = 3, and 3 — 1 — 1 = 1, making D = 1.

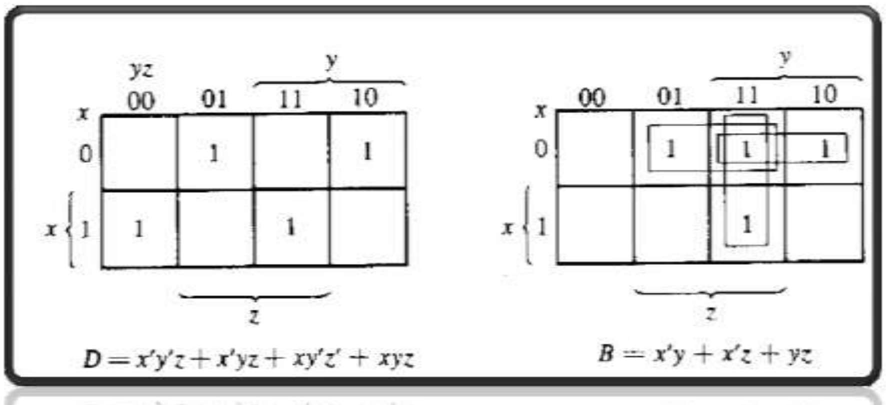
The simplified Boolean functions for the two outputs of the full-subtractor are derived in the maps of figure below. Thesimplified sum of products output functions are

D = x'y'z + x'yz' + xy'z' + xyz

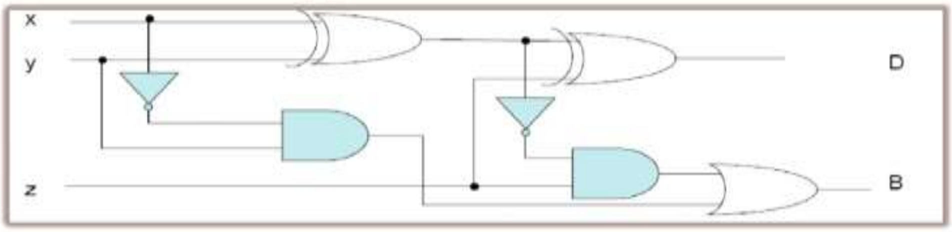
B = x'y + x'z + yz

$$D = x'y'z + x'yz' + xy'z' + xyz$$

$$B = x'y + x'z + yz$$



$D = x'y'z + x'yz + xy'z' + xyz$ $\quad$ $B = x'y + x'z + yz$



D=z⊕(x⊕y)

D=z'(xy'+x'y)+z(xy'+x'y)'

D=xy'z'+x'yz'+z(xy+x'y')

D=xy'z'+x'yz'+xyz+x'y'z

B=z(xy'+x'y)'+x'y

B=z(x'y'+xy)+x'y

=x'y'z+xyz+x'y

A full subtractor circuit can be implemented with two half subtractors and one OR gate**.**

N.B Full adder can be converted into full subtractor with an additional inverter.

•Four bit binary parallel adder can be constructed by using three full adders and one half adder or by using four full adders with input carry for least significant bit full adder is zero.

• Four bit binary parallel adder shown in figure is also called as Ripple carry adder.

What is a Ripple Carry adder and why is it called a Ripple Carry adder.
What are the various types of subtractor.

A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage.
The various types of subtractor are: Full subtractor and Half Subtractor

4.0 Conclusion

In this unit we discussed adders and subtractor and how their circuit diagram andfuctions.

5.0 Summary

You have learnt:

The half-adder accepts two binary digits on its inputs and produces two binary

digits on its outputs, a sum bit and a carry

The full-adder accepts two input bits and an input carry and generates a sum

output and an output carry.

The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry
A half-subtractor is a combinational circuit that subtracts two bits and produces their difference
A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage

6.0 Tutor-Marked Assignment

1. With the help of a truth table explain the working of a half subtractor. Draw the logic diagram using gates.

2. Draw the logic diagram of a full subtractor using half subtractors and explain its working withthe help of a truth table.

3. Discuss in detail, the working of Full Adder logic circuit and extend your discussion to explain a binary adder, which can be used to add two binary numbers

4. Distinguish between adder and subtractor

7.0 References/Further reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,

Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.

Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall

Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

http://www.circuitstoday.com/half-adder-and-full-adder

http://www.matestop.com/forum/thread/1200/a-complete-discription-of-half-adder-and-full-adder-circuit-with-diagrams-n/

http://www.scribd.com/doc/57832395/6/Half-Subtractor-x-y

http://media.careerlauncher.com.s3.amazonaws.com/gate/material/2.pdf

http://www.circuitstoday.com/half-adder-and-full-adder

http://www.electronics-tutorials.ws/combination/comb_5.html

http://www.electronics-tutorials.ws/combination/comb_8.html

Unit 2

Multiplexer and Demultiplexer

Content

1.0 Introduction

Multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination. The basic multiplexer has several data-input lines and a single output line. It also has data-select inputs. which permit digital data on anyone of the inputs to be switched to the output line. Multiplexers are also known as data selectors. On the other hand, a demultiplexer (DEMUX) basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of outputs. This unit discusses both.

2.0 Learning outcomes

At the end of this unit you should be able to

Define a multiplexer
Explain the basic operation of a multiplexer
Define a demultiplexer
Explain the basic operation of a de multiplexer

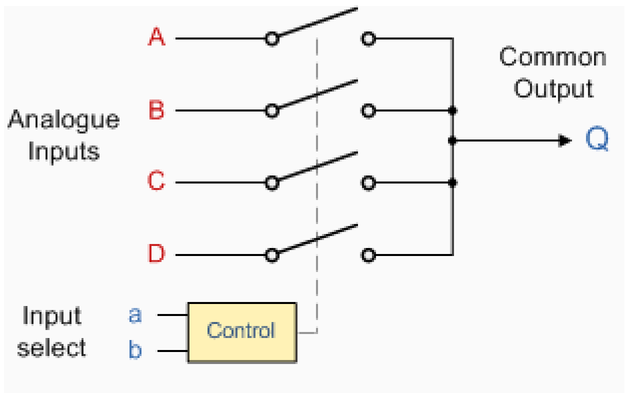
3.1 The Multiplexer

A data selector, more commonly called a **Multiplexer**, shortened to "Mux" or "MPX", are combinational logic switching devices that operate like a very fast acting multiple position rotary switch. They connect or control, multiple input lines called "channels" consisting of either 2, 4, 8 or 16 individual inputs, one at a time to an output. Then the job of a multiplexer is to allow multiple signals to *share* a single common output. For example, a single 8-channel multiplexer would connect one of its eight inputs to the single data output. Multiplexers are used as one method of reducing the number of logic gates required in a circuit or when a single data line is required to carry two or more different digital signals.

Digital **Multiplexers** are constructed from individual **analogue switches** encased in a single IC package as opposed to the "mechanical" type selectors such as normal conventional switches and relays. Generally, multiplexers have an even number of data inputs, usually an even power of two, n^2 , a number of "control" inputs that correspond with the number of data inputs and according to the binary condition of these control inputs, the appropriate data input is connected directly to the output. An example of a **Multiplexer** configuration is shown below.

4-to-1 Channel Multiplexer

The diagram for a 4 to 1 channel multiplexer is shown below



Addressing		Input Selected
B	A	
0	0	A
0	1	B
1	0	C
1	1	D

The Boolean expression for this 4-to-1 **Multiplexer** above with inputs A to D and data select lines a, b is given as:

Q = abA + abB + abC + abD

In this example at any one instant in time only ONE of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q. As to which switch is closed depends upon the addressing input code on lines "a" and "b", so for this example to select input B to the output at Q, the binary input address would need to be "a" = logic "1" and "b" = logic "0". Adding more control address lines will allow the multiplexer to control more inputs but each control line configuration will connect only ONE input to the output.

Then the implementation of this Boolean expression above using individual logic gates would require the use of seven individual gates consisting of AND, OR and NOT gates as shown.

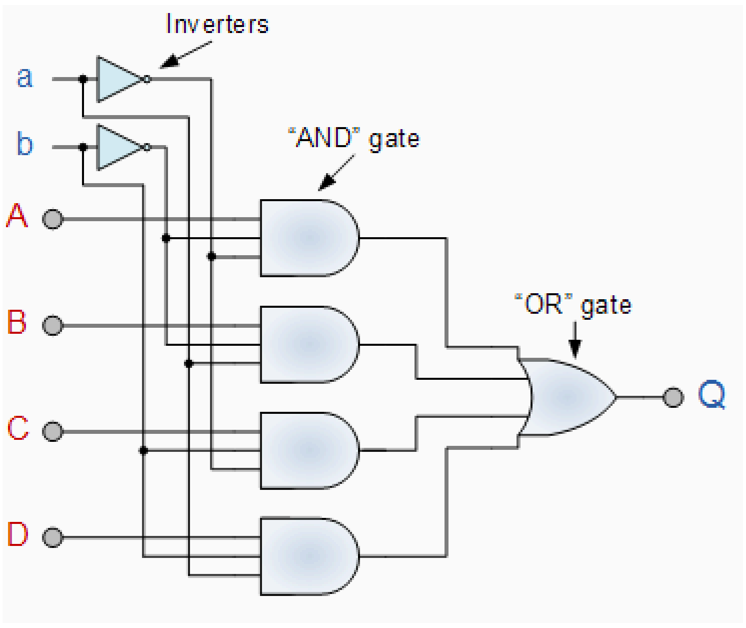
Self Assessment Questions

What do you understand as a Multiplexer?

Self Assessment Answers

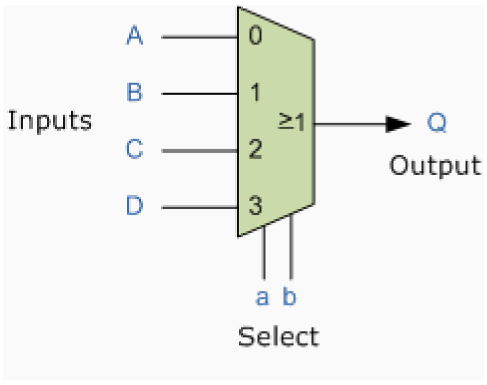
A Multiplexer isa combinational logic switching device that operate like a very fast acting multiple position rotary switch and are used to connect or control, multiple input lines called "channels" consisting of either 2, 4, 8 or 16 individual inputs, one at a time to an output.

4 Channel Multiplexer using Logic Gates



The symbol used in logic diagrams to identify a multiplexer is as follows.

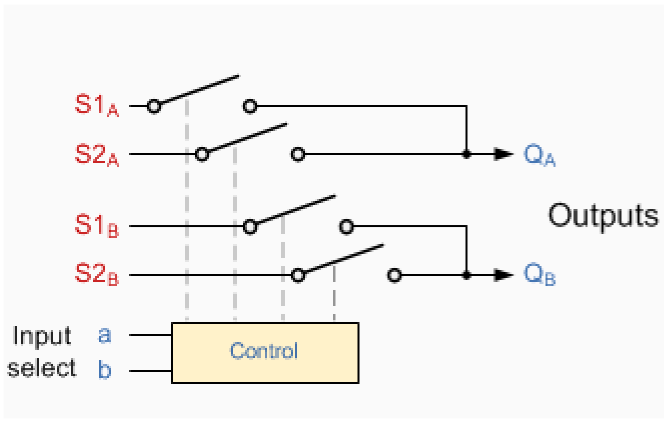
Multiplexer Symbol



Multiplexers are not limited to just switching a number of different input lines or channels to one common single output. There are also types that can switch their inputs to multiple outputs and have arrangements or 4 to 2, 8 to 3 or even 16 to 4 etc configurations and an example of a simple Dual channel 4 input multiplexer (4 to 2) is given below:

4-to-2 Channel Multiplexer

Here in this example the 4 input channels are switched to 2 individual output lines but larger arrangements are also possible. This simple 4 to 2 configuration could be used for example, to switch audio signals for stereo pre-amplifiers or mixers.



Digital multiplexers are sometimes also referred to as "Data Selectors" as they select the data to be sent to the output line and are commonly used in communications or high speed network switching circuits such as LAN´s and Ethernet applications. Some multiplexer IC´s have a single inverting buffer (NOT Gate) connected to the output to give a positive logic output (logic "1", HIGH) on one terminal and a complimentary negative logic output (logic "0", LOW) on another different terminal.

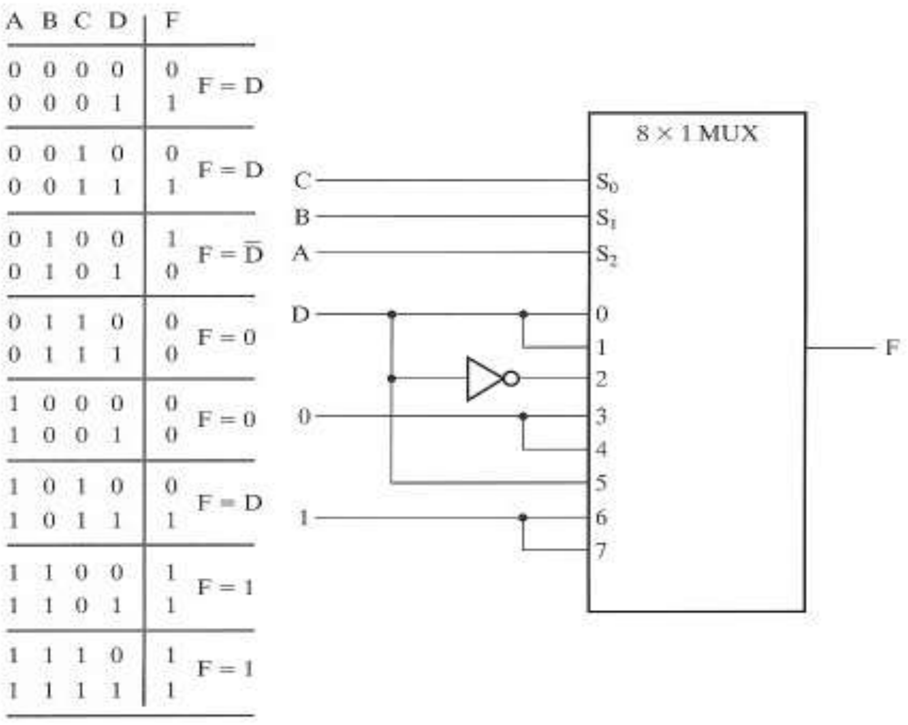
It is possible to make simple multiplexer circuits from standard **AND** and **OR** gates as we have seen above, but commonly multiplexers/data selectors are available as standard i.c. packages such as the common TTL 74LS151 8-input to 1 line multiplexer or the TTL 74LS153 Dual 4-input to 1 line multiplexer. Multiplexer circuits with much higher number of inputs can be obtained by cascading together two or more smaller devices.

The **Multiplexer** is a very useful combinational device that has its uses in many different applications such as signal routing, data communications and data bus control. When used with a demultiplexer, parallel data can be transmitted in serial form via a single data link such as a fibre-optic cable or telephone line. They can also be used to switch either analogue, digital or video signals, with the switching current in analogue power circuits limited to below 10mA to 20mA per channel in order to reduce heat dissipation.

Example: Implement the function below using multiplexer

$F(A,B,C,D)=\Sigma(1,2,3,11,12,13,14,15)$

Answer: The function is implemented an 8X1 multiplexer as shown below



A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = $\bar{D}$
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	

Example: A 4x1 Mux has 4 input lines (D0, D1, D2, D3), two select inputs (S0 & S1), and one output

line Y.

IF S1S0=00, **then** Y= D0

IF S1S0=01, **then** Y= D1

IF S1S0=10, **then** Y= D2

IF S1S0=11, **then** Y= D3

Thus, the output signal Y can be expressed as:

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

mintermmintermmintermminterm

m0 m1 m2 m3

Self Assessment Question

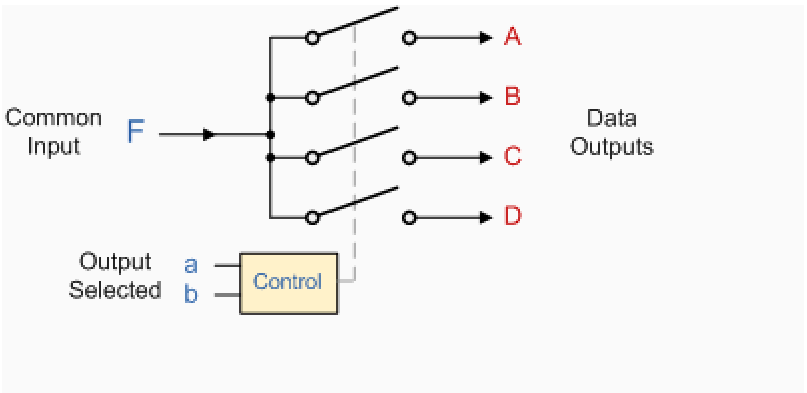
Consider the function F(A,B,C,D)=Σ(1,3,4,11,12,13,14,15). Implememt it using a 8X1 multiplexer

Self Assessment Answers

3.2 The Demultiplexer

The data distributor, known more commonly as a **Demultiplexer** or "Demux", is the exact opposite of the **Multiplexer** we saw in the previous tutorial. The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The **demultiplexer** converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer



Addressing		Input Selected
B	a	
0	0	A
0	1	B
1	0	C

1	1	D

The Boolean expression for this 1-to-4 **Demultiplexer** above with outputs A to D and data select lines a, b is given as:

F = ab A + abB + abC + abD

The function of the **Demultiplexer** is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" and by adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs. Some standard demultiplexer IC´s also have an "enable output" input pin which disables or prevents the input from being passed to the selected output.

Also some have latches built into their outputs to maintain the output logic level after the address inputs have been changed. However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic "0".

The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.

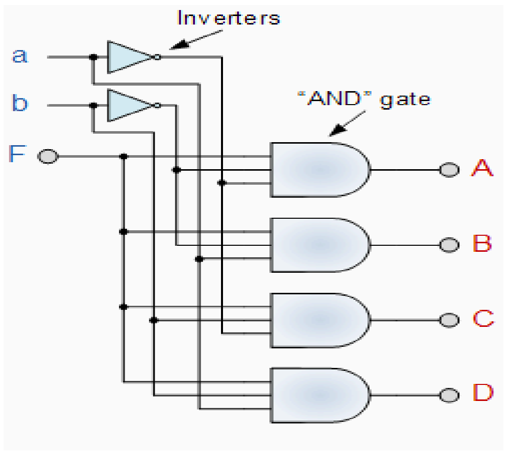
Self assessment Questions

What do you understand as a Demultiplexer?

Self Assessment Answers

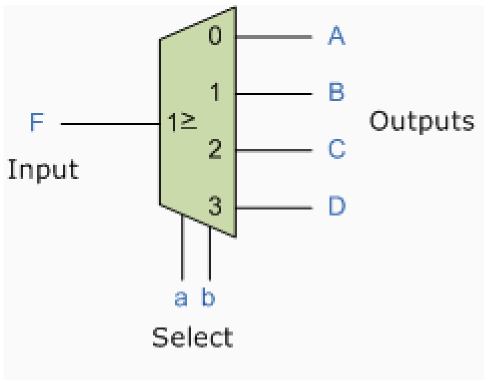
Basically the demultiplexer is the opposite of the Multiplexer which takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines

4 Channel Demultiplexer using Logic Gates



The symbol used in logic diagrams to identify a demultiplexer is as follows.

Demultiplexer Symbol



Standard **Demultiplexer** IC packages available are the TTL 74LS138 1 to 8-output demultiplexer, the TTL 74LS139 Dual 1-to-4 output demultiplexer or the CMOS CD4514 1-to-16 output demultiplexer. Another type of demultiplexer is the 24-pin, 74LS154 which is a 4-bit to 16-line demultiplexer/decoder. Here the individual output positions are selected using a 4-bit binary coded input. Like multiplexers, demultiplexers can also be cascaded together to form higher order demultiplexers.

Unlike multiplexers which convert data from a single data line to multiple lines and demultiplexers which convert multiple lines to a single data line, there are devices available which convert data to and from multiple lines and in the next tutorial about combinational logic devices, we will look at **Encoders** which convert multiple input lines into multiple output lines, converting the data from one form to another.

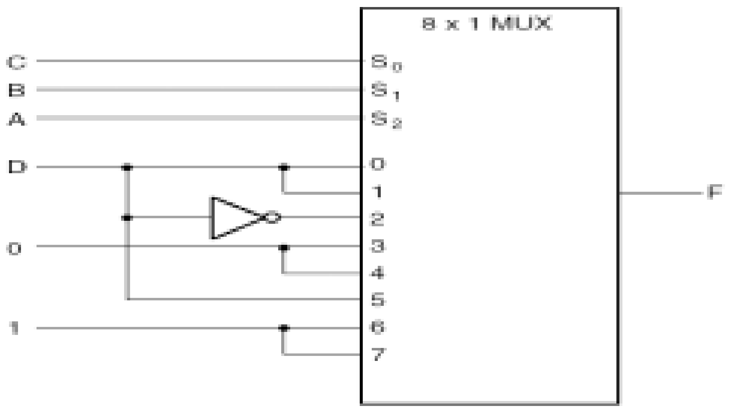
4.0 Conclusion

This unit takes a look at multiplexer and demultiplexer with the various types.

5.0 Summary

You have learnt that:

A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination. The basic multiplexer has severa] data-input lines and a single output line. Multiplexers are also known as data selectors.
A demultiplexer (DEMUX) basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. the demultiplexer is also known as a data distributor



6.0 Tutor Marked Assignment

1.Using a suitable logic diagram explain the working of a 1-to-16 de multiplexer.

2. What is a digital multiplexer? Illustrate its functional diagram. Write the scheme of a 4-input multiplexer using basic gates (AND/OR/NOT) and explain its operation.

3. Differentiate between multiplexer and de multiplexer

7.0 References/Further Reading

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
www.doc.ic.ac.uk/~dfg/hardware/HardwareLecture03.pd
www.doc.ic.ac.uk/~dfg/hardware/HardwareHandout02.pdf
www.itee.uq.edu.au/~engg1030/lectures/1perpage/lect14.pdf
http://www.indiabix.com/digital-electronics/combinational-logic-circuits/116006

Unit 2

Digital Encoder, Decoder and Comparator

Content

1.0 Introduction

Decoder is a digital circuit that detects the presence of a specified combination of bits (code) on its inputs and indicates the presence of that code by a specified output level. In its general form, a decoder has n input lines to handle n bits and from one to 2" output lines to indicate the presence of one or more n-bit combinations. An encoder is a combinational logic circuit that essentially performs a "reverse" decoder function. An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. Encoders can also be devised to encode var-ious symbols and alphabetic characters. This chapter takes a look at both including comparator.

2.0 Learning Outcomes

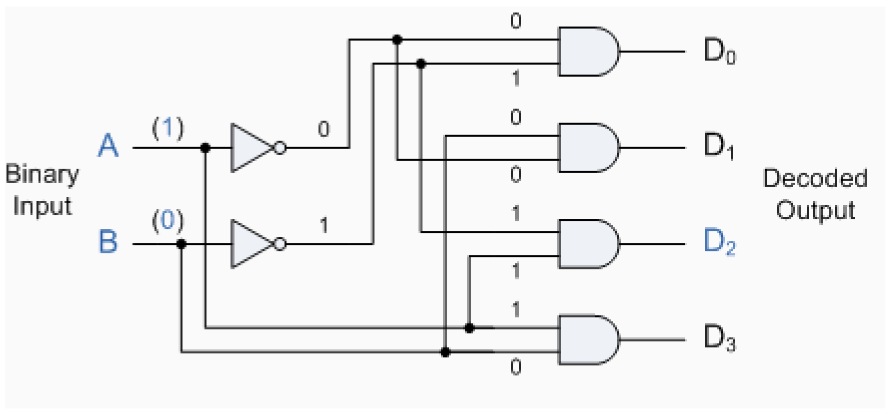
At the end of this unit you should be able to
Define decoder
Design a logic circuit to decode any combination of bits
Determine the logic for a decimal encoder .
Explain the purpose of the priority feature in decoder

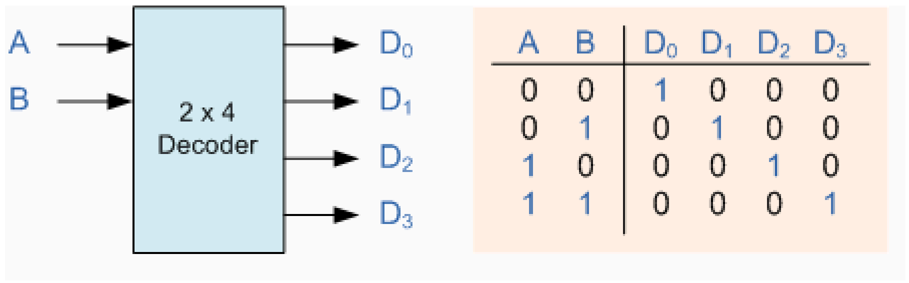
3.1 Digital Decoder

A **Decoder** is basically a combinational type logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output. **Binary Decoders** have inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, and a n-bit decoder has 2^n output lines. Therefore, if it receives n inputs (usually grouped as a binary or Boolean number) it activates one and only one of its 2^n outputs based on that input with all other outputs deactivated. A decoders output code normally has more bits than its input code and practical binary decoder circuits include, 2-to-4, 3-to-8 and 4-to-16 line configurations.

A binary decoder converts coded inputs into coded outputs, where the input and output codes are different and decoders are available to "decode" either a Binary or BCD (8421 code) input pattern to typically a Decimal output code. Commonly available BCD-to-Decimal decoders include the TTL 7442 or the CMOS 4028. An example of a 2-to-4 line decoder along with its truth table is given below. It consists of an array of four NAND gates, one of which is selected for each combination of the input signals A and B.

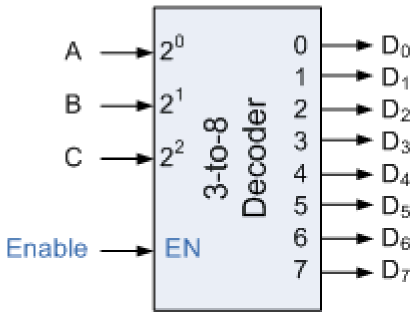
A 2 line -to-4line Binary Decoders.





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

In this simple example of a 2-to-4 line binary decoder, the binary inputs A and B determine which output line from D0 to D3 is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0" so only one output can be active (HIGH) at any one time. Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input and these types of binary decoders are commonly used as **Address Decoders** in microprocessor memory applications.



74LS138 Binary Decoder

Some binary decoders have an additional input labelled "Enable" that controls the outputs from the device. This allows the decoders outputs to be turned "ON" or "OFF" and we can see that the logic diagram of the basic decoder is identical to that of the basic demultiplexer. Therefore, we say that a demultiplexer is a decoder with an additional data line that is used to enable the decoder. An alternative way of looking at the decoder circuit is to regard inputs A, B and C as address signals. Each combination of A, B or C defines a unique address which can access a location having that address.

Self Assessment Questions

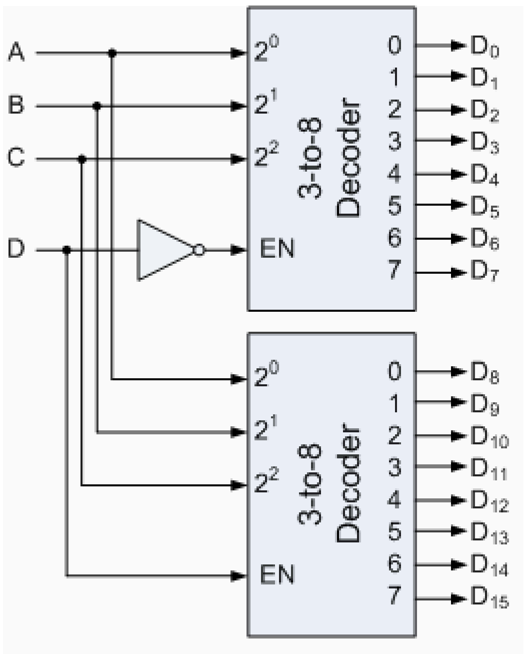
What is a decoder?

Self Assessment Answers

A Decoder is basically a combinational type logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output.

Sometimes it is required to have a **Binary Decoder** with a number of outputs greater than is available, or if we only have small devices available, we can combine multiple decoders together to form larger decoder networks as shown. Here a much larger 4-to-16 line binary decoder has been implemented using two smaller 3-to-8 decoders.

A 4 line -to-16 line Binary Decoder Configuration.


A
B
C
D
2^0
2^1
2^2
EN
3-to-8 Decoder
0 1 2 3 4 5 6 7
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
D_8 D_9 D_{10} D_{11} D_{12} D_{13} D_{14} D_{15}

4-to-16 Line Decoder Implemented with two 3-to-8 Decoders

Inputs A, B, C are used to select which output on either decoder will be at logic "1" (HIGH) and input D is used with the enable input to select which encoder either the first or second will output the "1".

Memory Address Decoder.

Binary Decoders are most often used in more complex digital systems to access a particular memory location based on an "address" produced by a computing device. In modern microprocessor systems the amount of memory required can be quite high and is generally more than one single memory chip alone. One method of overcoming this problem is to connect lots of individual memory chips together and to read the data on a common "Data Bus". In order to prevent the data being "read" from each memory chip at the same time, each memory chip is selected individually one at time and this process is known as **Address Decoding**.

Binary Decoders are very useful devices for converting one digital format to another, such as binary or BCD type data into decimal or octal etc and commonly available decoder IC's are the TTL 74LS138 3-to-8 line binary decoder or the 74ALS154 4-to-16 line decoder. They are also very useful for interfacing to 7-segment displays such as the TTL 74LS47 which we will look at in the next tutorial.

Decoders are available in two different types of output forms:

(1) Active high output type decoders: These types of decoders are constructed with AND gates and will give the output high for given input combination and all other output are low .

(2) Active low output type of decoders. These types of decoders will give the output low for given input combination and all other outputs are high. They are constructed with with NAND gates.

Self Assessment Questions

Identify the two types of Decoder

Self Assessment Answers

The Active high output and the Active low output

BCD to 7-Segment Display Decoder

As we saw in the previous lesson, a **Decoder**IC is a device which converts one digital format into another and the most commonly used device for doing this is the Binary Coded Decimal (BCD) to 7-Segment Display Decoder. 7-segment **LED** (Light Emitting Diode) or **LCD** (Liquid Crystal) displays, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numerical characters and they consist of 7 individual LED's (the segments), within one single display package.

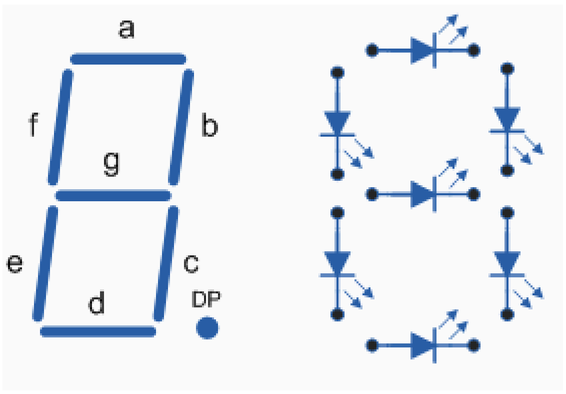
In order to produce the required numbers or HEX characters from 0 to 9 and A to F respectively, on the display the correct combination of LED segments need to be illuminated and **BCD to 7-segment Display Decoders** such as the 74LS47 do just that. A standard 7-segment LED display generally has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal segments. Some single displays have an additional input pin for the decimal point in their lower right or left hand corner.

There are two important types of 7-segment LED digital display.

The Common Cathode Display (CCD) - In the common cathode display, all the cathode connections of the LED's are joined together to logic "0" and the individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.

The Common Anode Display (CAD) - In the common anode display, all the anode connections of the LED's are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.

7-Segment Display Format



Truth Table for a 7-segment display

Individual Segments							Display
A	b	c	d	E	f	G	
×	×	×	×	×	×		0
	×	×					1
×	×		×	×		×	2
×	×	×	×			×	3
	×	×			×	×	4
×		×	×		×	×	5
×		×	×	×	×	×	6
×	×	×					7

Individual Segments							Display
a	b	c	D	e	f	g	
×	×	×	×	×	×	×	8
×	×	×			×	×	9
×	×	×		×	×	×	A
		×	×	×	×	×	B
×			×	×	×		C
	×	×	×	×		×	D
×			×	×	×	×	E
×				×	×	×	F

7-Segment Display Elements for all Numbers.

What are the two types of 7 segment display?

The two types of 7 segment display are the Common Cathode Display and the common Anode Display

It can be seen that to display any single digit number from 0 to 9 or letter from A to F, we would need 7 separate segment connections plus one additional connection for the LED's "common" connection. Also as the segments are basically a standard light emitting diode, the driving circuit would need to produce up to 20mA of current to illuminate each individual segment and to display the number 8, all 7 segments would need to be lit resulting a total current of nearly 140mA, (8 x 20mA). Obviously, the use of so many connections and power consumption is impractical for some electronic or microprocessor based circuits and so in order to reduce the number of signal lines required to drive just one single display, display decoders such as the BCD to 7-Segment Display Decoder and Driver IC's are used instead.

What are the two types of 7 segment display?

The two types of 7 segment display are the Common Cathode Display and the common Anode Display

SELF ASSESSMENT QUESTION

1 Draw a 3line by 8 line decoder

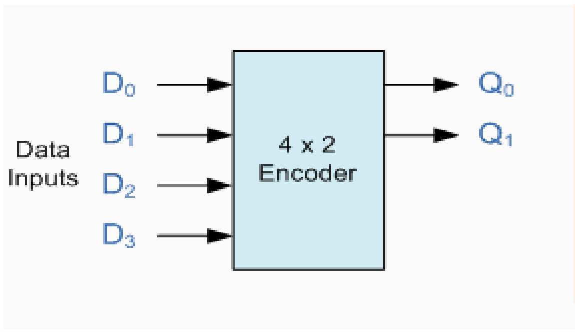
2. Distinguish between active high and active low output decoders

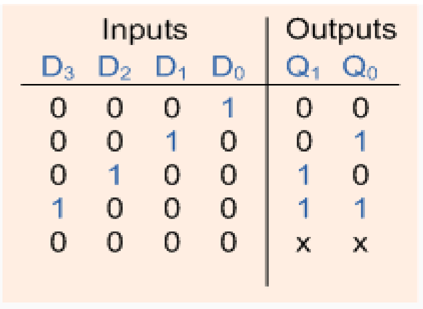
3.2 The Digital Encoder

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, a **Digital Encoder** more commonly called a **Binary Encoder** takes *ALL* its data inputs one at a time and then converts them into a single encoded output. So we can say that a binary encoder, is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output. Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An encoder is a digital function that produces a reverse operation from that of a decoder

An "n-bit" binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations. The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to "1" and are available to encode either a decimal or hexadecimal input pattern to typically a binary or B.C.D. output code.

4-to-2 Bit Binary Encoder





Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D_1 and D_2 HIGH at logic "1" at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output binary number that is different to the actual input present. Also, an output code of all logic "0"s can be generated when all of its inputs are at "0" OR when input D_0 is equal to one.

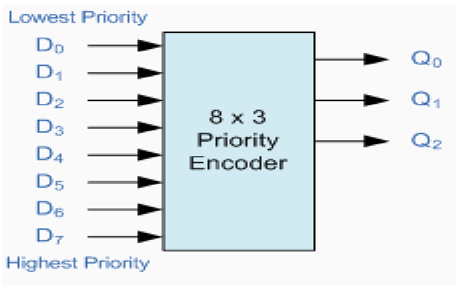
One simple way to overcome this problem is to "Prioritise" the level of each input pin and if there was more than one input at logic level "1" the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a **Priority Encoder** or **P-encoder** for short.

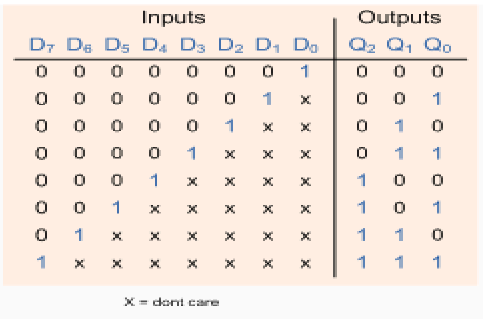
Differentiate between the Encoder and the Multiplexer

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, a Digital Encoder more commonly called a Binary Encoder takes *ALL* its data inputs one at a time and then converts them into a single encoded outputPriority Encoder

The **Priority Encoder** solves the problems mentioned above by allocating a priority level to each input. The *priority encoders* output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored. The priority encoder comes in many different forms with an example of an 8-input priority encoder along with its truth table shown below.

8-to-3 Bit Priority Encoder





Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic "0") inputs and provides a 3-bit code of the highest ranked input at its output. Priority encoders output the highest order input first for example, if input lines "D2", "D3" and "D5" are applied simultaneously the output code would be for input "D5" ("101") as this has the highest order out of the 3 inputs. Once input "D5" had been removed the next highest output code would be for input "D3" ("011"), and so on.

The truth table for a 8-to-3 bit priority encoder is given as:

Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0

0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

From this truth table, the Boolean expression for the encoder above with inputs D_0 to D_7 and outputs Q_0, Q_1, Q_2 is given as:

Output Q_0

$$Q_0 = \sum\left(1, 3, 5, 7\right)$$
$$Q_0 = \sum\left(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3\bar{D}_2D_1 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4D_3 + \bar{D}_7\bar{D}_6D_5 + D_7\right)$$
$$Q_0 = \sum\left(\bar{D}_6\bar{D}_4\bar{D}_2D_1 + \bar{D}_6\bar{D}_4D_3 + \bar{D}_6D_5 + D_7\right)$$
$$Q_0 = \sum\left(\bar{D}_6\left(\bar{D}_4\bar{D}_2D_1 + \bar{D}_4D_3 + D_5\right) + D_7\right)$$

Output Q_1

$$Q_1 = \sum\left(2, 3, 6, 7\right)$$
$$Q_1 = \sum\left(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3D_2 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4D_3 + \bar{D}_7D_6 + D_7\right)$$
$$Q_1 = \sum\left(\bar{D}_5\bar{D}_4D_2 + \bar{D}_5\bar{D}_4D_3 + D_6 + D_7\right)$$
$$Q_1 = \sum\left(\bar{D}_5\bar{D}_4\left(D_2 + D_3\right) + D_6 + D_7\right)$$

Output Q_2

$$Q_2 = \sum\left(4, 5, 6, 7\right)$$
$$Q_2 = \sum\left(\bar{D}_7\bar{D}_6\bar{D}_5D_4 + \bar{D}_7\bar{D}_6D_5 + \bar{D}_7D_6 + D_7\right)$$
$$Q_2 = \sum\left(D_4 + D_5 + D_6 + D_7\right)$$

Then the final Boolean expression for the priority encoder including the zero inputs is defined as:

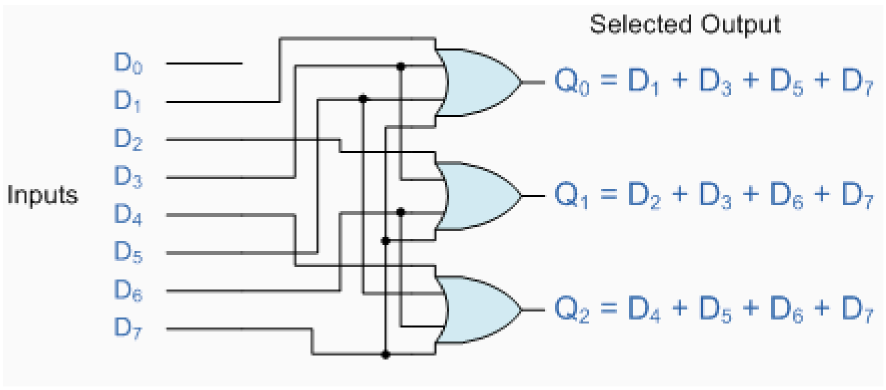
$$Q_0 = \sum\left(\overline{D}_6\left(\overline{D}_4\overline{D}_2 D_1 + \overline{D}_4 D_3 + D_5\right) + D_7\right)$$

$$Q_1 = \sum\left(\overline{D}_5\overline{D}_4(D_2 + D_3) + D_6 + D_7\right)$$

$$Q_2 = \sum(D_4 + D_5 + D_6 + D_7)$$

In practice these zero inputs would be ignored allowing the implementation of the final Boolean expression for the outputs of the 8-to-3 **priority encoder** above to be constructed using individual OR gates as follows.

Digital Encoder using Logic Gates


Selected Output
Inputs
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$Q_0 = D_1 + D_3 + D_5 + D_7$
$Q_1 = D_2 + D_3 + D_6 + D_7$
$Q_2 = D_4 + D_5 + D_6 + D_7$

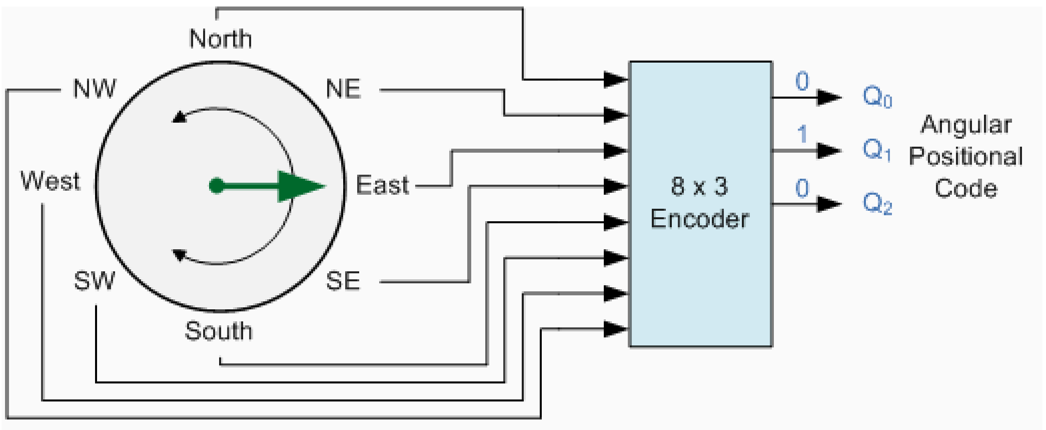
Encoder Applications

Keyboard Encoder

Priority encoders can be used to reduce the number of wires needed in a particular circuits or application that have multiple inputs. For example, assume that a microcomputer needs to read the 104 keys of a standard QWERTY keyboard where only one key would be pressed either "HIGH" or "LOW" at any one time. One way would be to connect all 104 wires from the keys directly to the computer but this would be impractical for a small home PC, but another better way would be to use a priority encoder. The 104 individual buttons or keys could be encoded into a standard ASCII code of only 7-bits (0 to 127 decimal) to represent each key or character of the keyboard and then inputted as a much smaller 7-bit B.C.D code directly to the computer. Keypad encoders such as the 74C923 20-key encoder are available to do just that.

Positional Encoders

Another more common application is in magnetic positional control as used on ships or robots etc. Here the angular or rotary position of a compass is converted into a digital code by an encoder and inputted to the systems computer to provide navigational data and an example of a simple 8 position to 3-bit output compass encoder is shown below. Magnets and reed switches could be used to indicate the compasses angular position.



Compass Direction	Binary Output		
	Q_0	Q_1	Q_2
North	0	0	0
North-East	0	0	1
East	0	1	0
South-East	0	1	1
South	1	0	0
South-West	1	0	1
West	1	1	0
North-West	1	1	1

Interrupt Requests

Other applications especially for **Priority Encoders** may include detecting interrupts in microprocessor applications. Here the microprocessor uses interrupts to allow peripheral devices such as the disk drive, scanner, mouse, or printer etc, to communicate with it, but the microprocessor can only "talk" to one peripheral device at a time. The processor uses "Interrupt Requests" or "IRQ" signals to assign priority to the devices to ensure that the most important peripheral device is serviced first. The order of importance of the devices will depend upon their connection to the priority encoder.

IRQ Number	Typical Use	Description
IRQ 0	System timer	Internal System Timer.
IRQ 1	Keyboard	Keyboard Controller.

IRQ 3	COM2 & COM4	Second and Fourth Serial Port.
IRQ 4	COM1 & COM3	First and Third Serial Port.
IRQ 5	Sound	Sound Card.
IRQ 6	Floppy disk	Floppy Disk Controller.
IRQ 7	Parallel port	Parallel Printer.
IRQ 12	Mouse	PS/2 Mouse.
IRQ 14	Primary IDE	Primary Hard Disk Controller.
IRQ 15	Secondary IDE	Secondary Hard Disk Controller.

Because implementing such a system using priority encoders such as the standard 74LS148 priority encoder IC involves additional logic circuits, purpose built integrated circuits such as the 8259 Programmable Priority Interrupt Controller is available.

SELF ASSESSMENT QUESTION

Identify two applications of the Encoder

The applications include: positional encoder as found in the mouse and also the keyboard encoder as found in the keyboard

3.3 The Digital Comparator

Another common and very useful combinational logic circuit is that of the **Digital Comparator** circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs. For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean algebra. There are two main types of digital comparator available and these are.

Identity Comparator - an *Identity Comparator* is a digital comparator that has only one output terminal for when A = B either "HIGH" A = B = 1 or "LOW" A = B = 0

Magnitude Comparator - a *Magnitude Comparator* is a type of digital comparator that has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

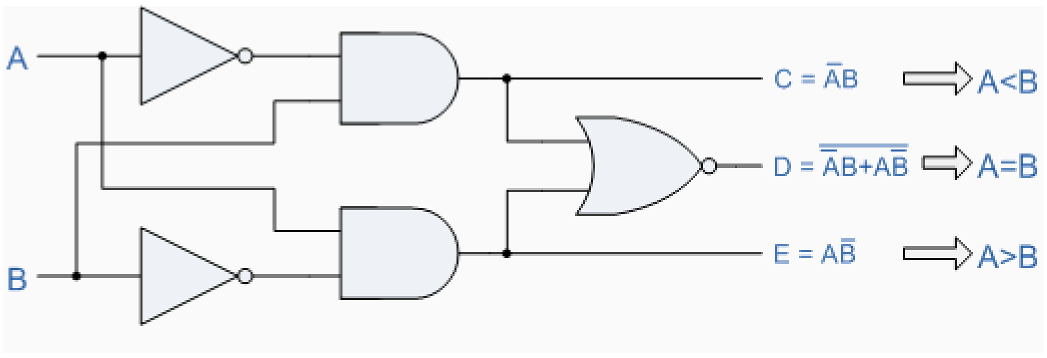
The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, ... An, etc) against that of a constant or unknown value such as B (B1, B2, B3, ...Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$A > B,\quad A = B,\quad A < B$

Which means: A is greater than B, A is equal to B, and A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

1-bit Comparator



Then the operation of a 1-bit digital comparator is given in the following Truth Table.

Truth Table

Inputs		Outputs		
B	A	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

You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the n-bits giving: $Q = A \oplus B$

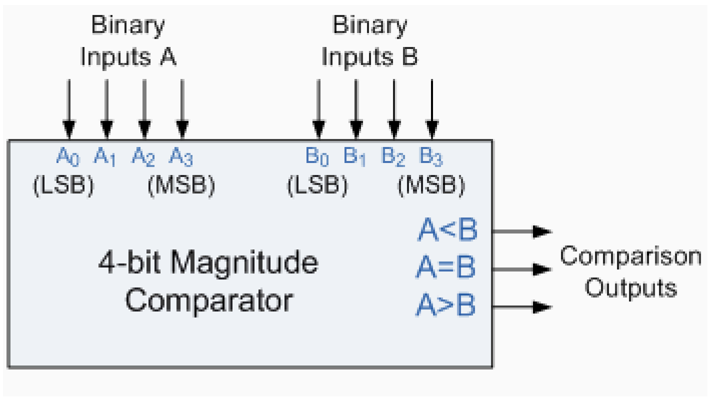
Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the "magnitude" of these values, a logic "0" against a logic "1" which is where the term **Magnitude Comparator** comes from.

How many outputs does the identity Comparator have?

The comparator has only one output

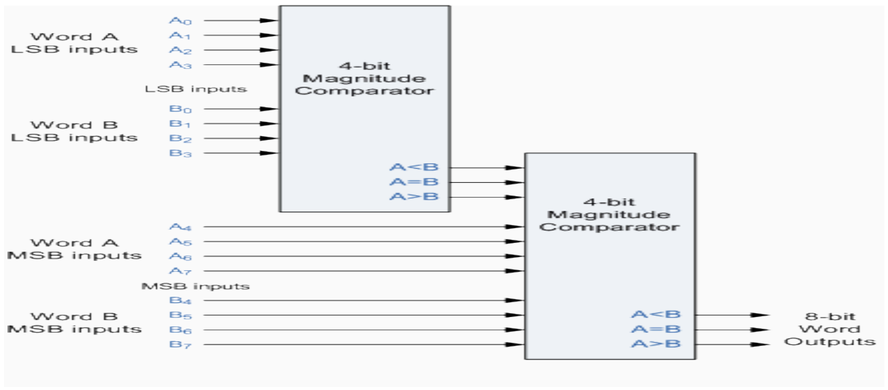
As well as comparing individual bits, we can design larger bit comparators by cascading together n of these and produce a n-bit comparator just as we did for the n-bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other. A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

3.3.2 4-bit Magnitude Comparator



Some commercially available digital comparators such as the TTL 7485 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

8-bit Word Comparator



When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = B then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal. If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops.

Digital Comparatorare used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.

5.0 CONCLUSION

This unit takes a look at binary decoder, encoder, and comparator with their types.

6.0 SUMMARY

You have learnt:

(i). Decoders are widely used in the memory system of computer, where they respond to the address code input from the CPU to activate the memory storage location specified by the address code.
(ii). Decoders are also used to convert binary data to a form suitable for displaying on decimal read outs.
(iii). Decoders can be used to implement combinational circuits, Boolean functions etc.
(iv). decoders can also be used as demultiplexers.

Digital Encoder is a combinational circuit that generates a specific code at its outputs such as binary or BCD in response to one or more active inputs

There are two main types of digital encoder. The **Binary Encoder** and the **Priority Encoder**.

The **Binary Encoder** converts one of 2^n inputs into an n-bit output. Then a binary encoder has fewer output bits than the input code.

Binary encoders are useful for compressing data and can be constructed from simple AND or OR gates. One of the main disadvantages of a standard binary encoder is that it would produce an error at its outputs if more than one input were active at the same time. To overcome this problem priority encoders were developed.

The **Priority Encoder** is another type of combinational circuit similar to a binary encoder, except that it generates an output code based on the highest prioritised input.

Priority encoders are used extensively in digital and computer systems as microprocessor interrupt controllers where they detect the highest priority input.

A magnitude comparator is a combinational circuit that compares two numbers A &B to determine whether:A □□B, orA = B, or A□□□B

TUTOR MARK ASSIGNMENT

1. Implement full adder with a decoder.

2. What is an encoder? Draw the logic circuit of Decimal to BCD encoder and explain its working.

3. Design a BCD to seven segment decoder that accepts a decimal digit in BCS and generates the appropriate output for segments in display indicator

7.0 REFERENCES/FURTHER READING

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
http://www.circuitstoday.com/half-adder-and-full-adder
http://www.electronics-tutorials.ws/combination/comb_5.html
http://www.electronics-tutorials.ws/combination/comb_8.html

MODULE 4

UNIT 1

SEQUENTIAL LOGIC BASICS: LATCHES AND FLIP FLOP

CONTENT

1.0 INTRODUCTION

This module begins a study of the fundamentals of sequential logic. Bistable, monostable, and astable logic devices called multivibrators are covered. Two categories of bistable devices are the latch and the flip-flop. Bistable devices have two stable states, called SET and RESET; they can retain either of these states indefinitely, making them useful as storage devices. The basic difference between latches and flip-flops is the way in which they are changed from one state to the other. The flip-flop is a basic building block for counters, registers, and other sequential control logic and is used in certain types of memories. The monostablemultivibrator, commonly known as the one-shot, has only one stable state. A one-shot produces a single controlled-width pulse when activated or triggered. The astablemultivibrator has no stable state and is used primarily as an oscillator, which is a self-sustained waveform generator. Pulse oscillators are used as the sources for timing waveforms in digital systems.

2.0 OBJECTIVES

At the end of this unit you should be able to

Understand the meaning of sequential logic circuits
Be able to differentiate between sequential and combibatioal logic circuits
Use logic gates to construct basic latches
Explain the difference between an S-R latch and a D latch

Recognize the difference between a latch and a flip-flop
Explain how S-R. D, and J-K flip-flops differ
Understand the significance of propagation delays, set-up time, hold time, maximum operating frequency, minimum clock pulse widths, and power dissipation in the application of flip-flops
Apply flip-flops in basic applications

3.0 MAIN CONTENT

3.1 Sequential Logic Circuits

Unlike **Combinational Logic** circuits that change state depending upon the actual signals being applied to their inputs at that time, **Sequential Logic** circuits have some form of inherent "Memory" built in to them as they are able to take into account their previous input state as well as those actually present, a sort of "before" and "after" is involved with sequential circuits.

In other words, the output state of a sequential logic circuit is a function of the following three states, the "present input", the "past input" and/or the "past output". *Sequential Logic circuits* remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits "Memory".

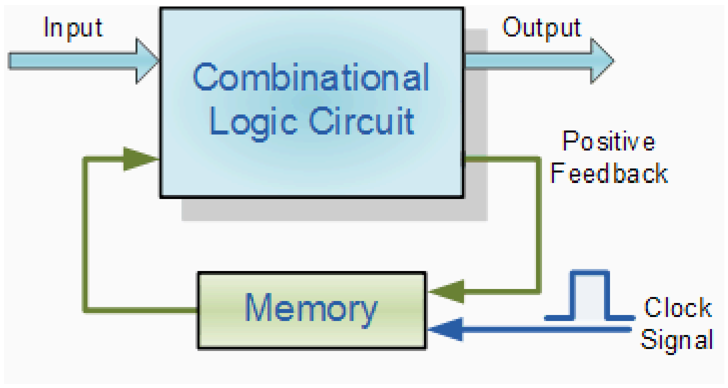
Sequential logic circuits are generally termed as *two state* or **Bistable** devices which can have their output or outputs set in one of two basic states, a logic level "1" or a logic level "0" and will remain "latched" (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause the bistable to change its state once again.

What is the basic difference between the Combinational and the Sequential Logic Circuit?

The fundamental difference is the fact that the sequential Logic circuit has some form of inherent memory while the combinational Logic circuit does not have.

Sequential Logic Representation

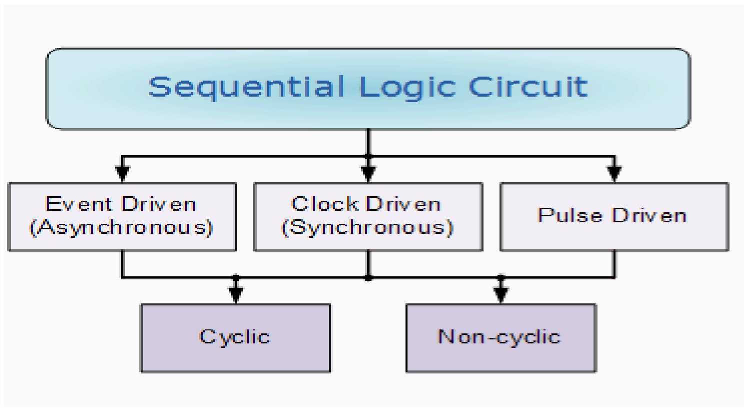
The word "Sequential" means that things happen in a "sequence", one after another and in **Sequential Logic** circuits, the actual clock signal determines when things will happen next. Simple sequential logic circuits can be constructed from standard **Bistable** circuits such as Flip-flops, Latches and Counters and which themselves can be made by simply connecting together universal **NAND Gates** and/or **NOR Gates** in a particular combinational way to produce the required sequential circuit.



Classification of Sequential Logic

As standard logic gates are the building blocks of combinational circuits, bistable latches and flip-flops are the building blocks of Sequential Logic Circuits. Sequential logic circuits can be constructed to produce either simple edge-triggered flip-flops or more complex sequential circuits such as storage registers, shift registers, memory devices or counters. Either way sequential logic circuits can be divided into the following three main categories:

1. Event Driven - asynchronous circuits that change state immediately when enabled.
2. Clock Driven - synchronous circuits that are synchronised to a specific clock signal.
3. Pulse Driven - which is a combination of the two that responds to triggering pulses.



As well as the two logic states mentioned above logic level "1" and logic level "0", a third element is introduced that separates **sequential logic** circuits from their **combinational logic** counterparts, namely *TIME*. Sequential logic circuits that return back to their original state once reset, i.e. circuits with loops or feedback paths are said to be "cyclic" in nature.

We now know that in sequential circuits changes occur only on the application of a clock signal making it synchronous, otherwise the circuit is asynchronous and depends upon an external input. To retain their current state, sequential circuits rely on feedback and this occurs when a fraction of the output is fed back to the input and this is demonstrated as

Identify the various classes of the sequential Logic Circuit.

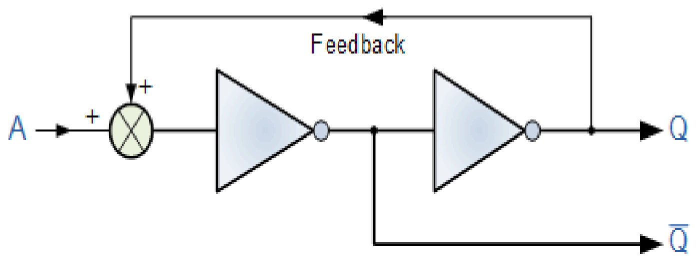
The various classes includes;

1. Event Driven - asynchronous circuits that change state immediately when enabled.

2. Clock Driven - synchronous circuits that are synchronised to a specific clock signal.

3. Pulse Driven - which is a combination of the two that responds to triggering pulses.

Sequential Feedback Loop



The two inverters or NOT gates are connected in series with the output at Q fed back to the input. Unfortunately, this configuration never changes state because the output will always be the same, either a "1" or a "0", it is permanently set. However, we can see how feedback works by examining the most basic sequential logic components, called the SR flip-flop.

LATCHES

The latch is a type of temporary storage device that has two stable states (bistable) and is normally placed in a category separate from that of flip-flops. Latches are similar to flip-flops because they are bistable devices that can reside in either of two states using a feedback arrangement, in which the outputs are connected back to the opposite inputs. The main difference between latches and flip-flops is in the method used for changing their state.

3.2 SR Flip-Flop

The **SR flip-flop**, also known as a *SR Latch*, can be considered as one of the most basic sequential logic circuit possible. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will "SET" the device (meaning the output = "1"), and is labelled S and another which will "RESET" the device (meaning the output = "0"), labelled R. Then the SR description stands for "Set-Reset". The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level "1" or logic "0" depending upon this set/reset condition.

A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it's current state or history. The term "Flip-flop" relates to the actual operation of the device, as it can be "flipped" into one logic Set state or "flopped" back into the opposing logic Reset state.

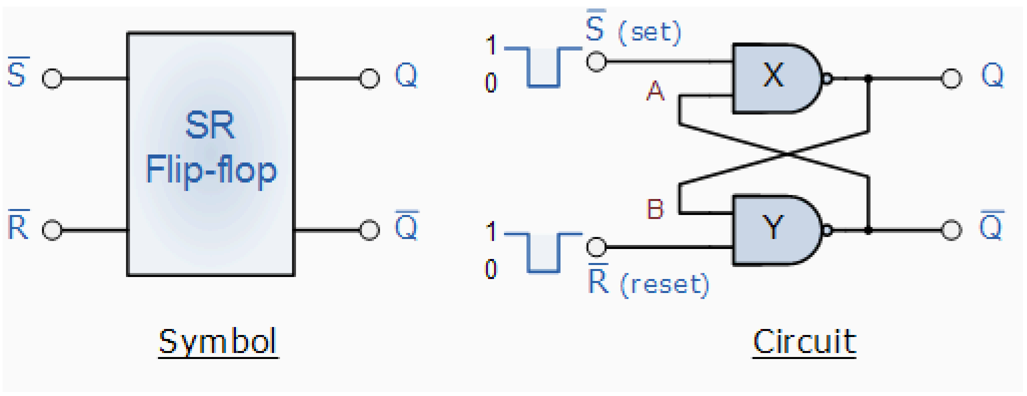
What is the main difference between Latches and Flip Flops

The main difference between latches and flip-flops is in the method used for changing their state.

The NAND Gate SR Flip-Flop

The simplest way to make any basic single bit set-reset SR flip-flop is to connect together a pair of cross-coupled 2-input NAND gates as shown, to form a Set-Reset Bistable also known as an active LOW SR NAND Gate Latch, so that there is feedback from each output to one of the other NAND gate inputs. This device consists of two inputs, one called the *Set*, S and the other called the *Reset*, R with two corresponding outputs Q and its inverse or complement Q (not-Q) as shown below.

The Basic SR Flip-flop



The Set State

Consider the circuit shown above. If the input R is at logic level "0" (R = 0) and input S is at logic level "1" (S = 1), the NAND gate *Y* has at least one of its inputs at logic "0" therefore, its output Q must be at a logic level "1" (NAND Gate principles). Output Q is also fed back to input "A" and so both inputs to NAND gate *X* are at logic level "1", and therefore its output Q must be at logic level "0". Again NAND gate principals. If the reset input R changes state, and goes HIGH to logic "1" with S remaining HIGH also at logic level "1", NAND gate *Y* inputs are now R = "1" and B = "0". Since one of its inputs is still at logic level "0" the output at Q still remains HIGH at logic level "1" and there is no change of state. Therefore, the flip-flop circuit is said to be "Latched" or "Set" with Q = "1" and Q = "0".

Reset State

In this second stable state, Q is at logic level "0", (not Q = "0") its inverse output at Q is at logic level "1", (Q = "1"), and is given by R = "1" and S = "0". As gate *X* has one of its inputs at logic "0" its output Q must equal logic level "1" (again NAND gate principles). Output Q is fed back to input "B", so both inputs to NAND gate *Y* are at logic "1", therefore, Q = "0". If the set input, S now changes state to logic "1" with input R remaining at logic "1", output Q still remains LOW at logic level "0" and there is no change of state. Therefore, the flip-flop circuits "Reset" state has also been latched and we can define this "set/reset" action in the following truth table.

Truth Table for this Set-Reset Function

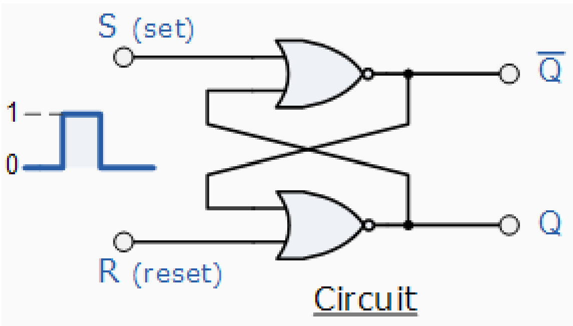
State	S	R	Q	Q	Description
Set	1	0	1	0	Set Q » 1
	1	1	1	0	no change
Reset	0	1	0	1	Reset Q » 0
	1	1	0	1	no change
Invalid	0	0	0	1	memory with Q = 0
	0	0	1	0	memory with Q = 1

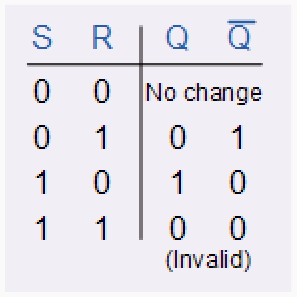
It can be seen that when both inputs S = "1" and R = "1" the outputs Q and Q can be at either logic level "1" or "0", depending upon the state of inputs S or R BEFORE this input condition existed. However, input state R = "0" and S = "0" is an undesirable or invalid condition and must be avoided because this will give both outputs Q and Q to be at logic level "1" at the same time and we would normally want Q to be the inverse of Q. However, if the two inputs are now switched HIGH again after this condition to logic "1", both the outputs will go LOW resulting in the flip-flop becoming unstable and switch to an unknown data state based upon the unbalance. This unbalance can cause one of the outputs to switch faster than the other resulting in the flip-flop switching to one state or the other which may not be the required state and data corruption will exist. This unstable condition is known as its **Meta-stable** state.

Then, a bistable SR flip-flop or SR latch is activated or set by a logic "1" applied to its S input and deactivated or reset by a logic "1" applied to its R. The SR flip-flop is said to be in an "invalid" condition (Meta-stable) if both the set and reset inputs are activated simultaneously.

The NOR gate SR Flip-flop

As well as using NAND gates, it is also possible to construct simple one-bit **SR Flip-flops** using two cross-coupled NOR gates connected in the same configuration. The circuit will work in a similar way to the NAND gate circuit above, except that the inputs are active HIGH and the invalid condition exists when both its inputs are at logic level "1", and this is shown below.



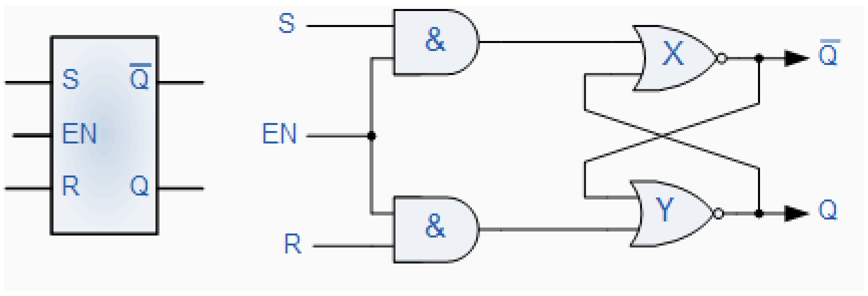


S	R	Q	$\overline{Q}$
0	0	No change	
0	1	0	1
1	0	1	0
1	1	0 (Invalid)	0

Gated or Clocked SR Flip-Flop

It is sometimes desirable in sequential logic circuits to have a bistable SR flip-flop that only changes state when certain conditions are met regardless of the condition of either the Set or the Reset inputs. By connecting a 2-input AND gate in series with each input terminal of the SR Flip-flop a Gated SR Flip-flop can be created. This extra conditional input is called an "Enable" input and is given the prefix of "EN". The addition of this input means that the output at Q only changes state when it is HIGH and can therefore be used as a clock (CLK) input making it level-sensitive as shown below.

Gated SR Flip-flop



When the Enable input "EN" is at logic level "0", the outputs of the two AND gates are also at logic level "0", (AND Gate principles) regardless of the condition of the two inputs S and R, latching the two outputs Q and Q into their last known state. When the enable input "EN" changes to logic level "1" the circuit responds as a normal SR bistable flip-flop with the two AND gates becoming transparent to the Set and Reset signals. This enable input can also be connected to a clock timing signal adding clock synchronisation to the flip-flop creating what is sometimes called a "Clocked SR Flip-flop". So a **Gated Bistable SR Flip-flop** operates as a standard bistable latch but the outputs are only activated when a logic "1" is applied to its EN input and deactivated by a logic "0".

Identify the basic Problems with the SR Flip Flops

The basic problem of the SR Flip Flops are number one, the S = 0 and R = 0 condition or S = R = 0 must always be avoided, and number two, if S or R change state while the enable input is high the correct latching action may not occur

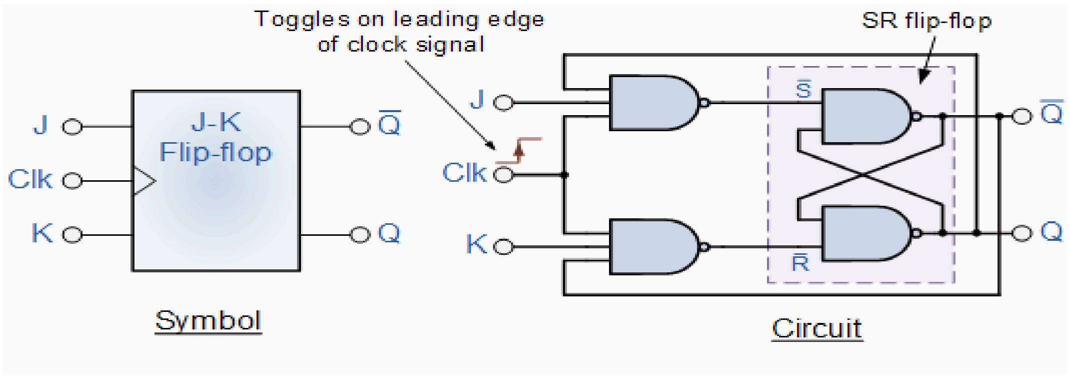
The JK Flip-flop

JK Flip-flopis named after its inventor, Jack Kilby. The JK flip-flop is the most widely used of all the flip-flop designs as it is considered to be a universal device.From the previous lessonsl we now know that the basic gated SR NAND flip-flop suffers from two basic problems: number one, the S = 0 and R = 0 condition or S = R = 0 must always be avoided, and number two, if S or R change state while the enable input is high the correct latching action may not occur. Then to overcome these two fundamental design problems with the SR flip-flop, the **JK flip-Flop** was developed.

This simple **JK flip-Flop** is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit. The sequential operation of the JK flip-flop is exactly the same as for the previous SR flip-flop with the same "set" and "reset" inputs. The difference this time is that the JK flip-flop has no invalid or forbidden input states of the SR Latch (when S and R are both 1).

The **JK flip-flop** is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1". Due to this additional clocked input, a JK flip-flop has four possible input combinations, "logic 1", "logic 0", "no change" and "toggle". The symbol for a JK flip-flop is similar to that of an **SR Bistable Latch** as seen in the previous tutorial except for the addition of a clock input.

The Basic JK Flip-flop



Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack Kilby. Then this equates to: J = S and K = R.

The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked. If the circuit is "SET" the J input is inhibited by the "0" status of Q through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q are always different we can use them to control the input. When both inputs J and K are equal to logic "1", the JK flip-flop toggles as shown in the following truth table.

The Truth Table for the JK Function

	Input		Output		Description
	J	K	Q	Q	
same as for the SR Latch	0	0	0	0	Memory no change
	0	0	0	1	
	0	1	1	0	Reset Q » 0
	0	1	0	1	
	1	0	0	1	Set Q » 1
	1	0	1	0	
toggle action	1	1	0	1	Toggle
	1	1	1	0	

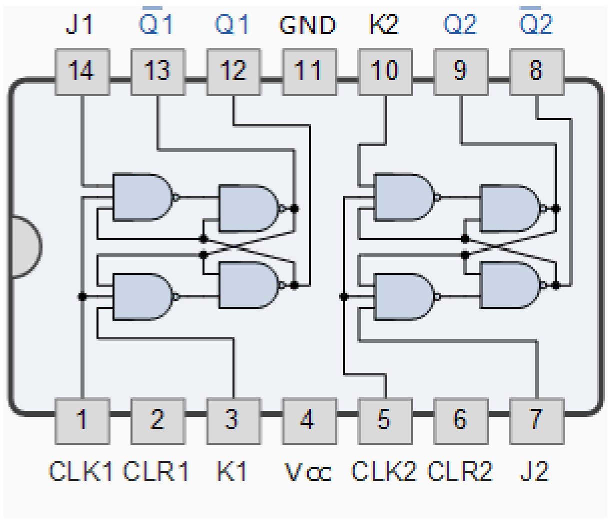
Then the JK flip-flop is basically an SR flip-flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip-flop circuit. Also when both the J and the K inputs are at logic level "1" at the same time, and the clock input is pulsed either "HIGH", the circuit will "toggle" from its SET state to a RESET state, or visa-versa. This results in the JK flip-flop acting more like a T-type toggle flip-flop when both terminals are "HIGH".

Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called "race" if the output Q changes state before the timing pulse of the clock input has time to go "OFF". To avoid this the timing pulse period (T) must be kept as short as possible (high frequency).

As this is sometimes not possible with modern TTL IC's the much improved **Master-Slave JK Flip-flop** was developed. This eliminates all the timing problems by using two SR flip-flops connected together in series, one for the "Master" circuit, which triggers on the leading edge of the clock pulse and the other, the "Slave" circuit, which triggers on the falling edge of the clock pulse. This results in the two sections, the master section and the slave section being enabled during opposite half-cycles of the clock signal.

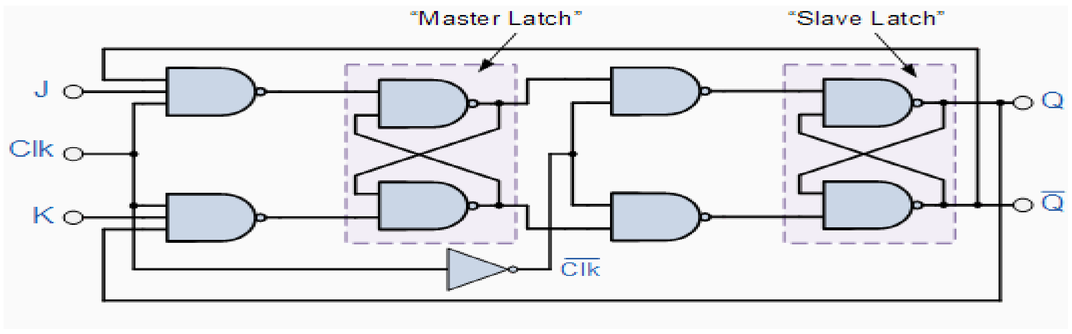
The 74LS73 is a Dual JK flip-flop IC, which contains two individual JK type bistable's within a single chip enabling single or master-slave toggle flip-flops to be made. Other JK flip-flop IC's include the 74LS107 Dual JK flip-flop with clear, the 74LS109 Dual positive-edge triggered JK flip-flop and the 74LS112 Dual negative-edge triggered flip-flop with both preset and clear inputs.

Dual JK Flip-flop 74LS73



The Master-Slave JK Flip-flop

The **Master-Slave Flip-Flop** is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip-flop being connected to the two inputs of the "Slave" flip-flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip-flop as shown below.



The input signals J and K are connected to the gated "master" SR flip-flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip-flop is the inverse (complement) of the "master" clock input, the "slave" SR flip-flop does not toggle. The outputs from the "master" flip-flop are only "seen" by the gated "slave" flip-flop when the clock input goes "LOW" to logic level "0". When the clock is "LOW", the outputs from the "master" flip-flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip-flop now responds to the state of its inputs passed over by the "master" section. Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip-flop are fed through to the gated inputs of the "slave" flip-flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip-flop edge or pulse-triggered.

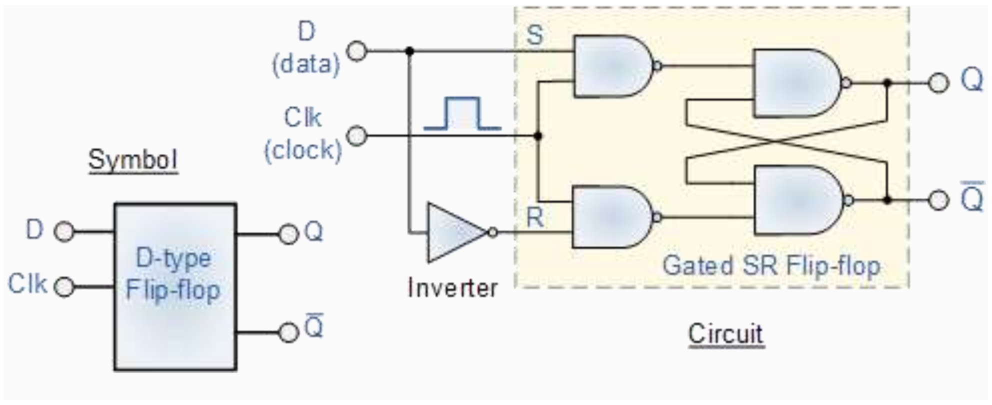
Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal. In other words, the **Master-Slave JK Flip-flop** is a "Synchronous" device as it only passes data with the timing of the clock signal.

The D flip-flop

One of the main disadvantages of the basic **SR NAND Gate**bistable circuit is that the indeterminate input condition of "SET" = logic "0" and "RESET" = logic "0" is forbidden. This state will force both outputs to be at logic "1", over-riding the feedback latching action and whichever input goes to logic level "1" first will lose control, while the other input still at logic "0" controls the resulting state of the latch. In order to prevent this from happening an inverter can be connected between the "SET" and the "RESET" inputs to produce another type of flip-flop circuit called a **Data Latch**, **Delay flip-flop**, **D-type Bistable** or simply a **D-type flip-flop** as it is more generally called.

The **D flip-flop** is by far the most important of the clocked flip-flops as it ensures that ensures that inputs S and R are never equal to one at the same time. D-type flip-flops are constructed from a gated SR flip-flop with an inverter added between the S and the R inputs to allow for a single D (data) input. This single data input D is used in place of the "set" signal, and the inverter is used to generate the complementary "reset" input thereby making a level-sensitive D-type flip-flop from a level-sensitive RS-latch as now S = D and R = not D as shown.

D flip-flop Circuit



We remember that a simple SR flip-flop requires two inputs, one to "SET" the output and one to "RESET" the output. By connecting an inverter (NOT gate) to the SR flip-flop we can "SET" and "RESET" the flip-flop using just one input as now the two input signals are complements of each other. This complement avoids the ambiguity inherent in the SR latch when both inputs are LOW, since that state is no longer possible.

Thus the single input is called the "DATA" input. If this data input is HIGH the flip-flop would be "SET" and when it is LOW the flip-flop would be "RESET". However, this would be rather pointless since the flip-flop's output would always change on every data input. To avoid this an additional input called the "CLOCK" or "ENABLE" input is used to isolate the data input from the flip-flop after the desired data has been stored. The effect is that D is only copied to the output Q when the clock is active. This then forms the basis of a **D flip-flop**.

The **D flip-flop** will store and output whatever logic level is applied to its data terminal so long as the clock input is HIGH. Once the clock input goes LOW the "set" and "reset" inputs of the flip-flop are both held at logic level "1" so it will not change state and store whatever data was present on its output before the clock transition occurred. In other words the output is "latched" at either logic "0" or logic "1".

Truth Table for the D Flip-flop

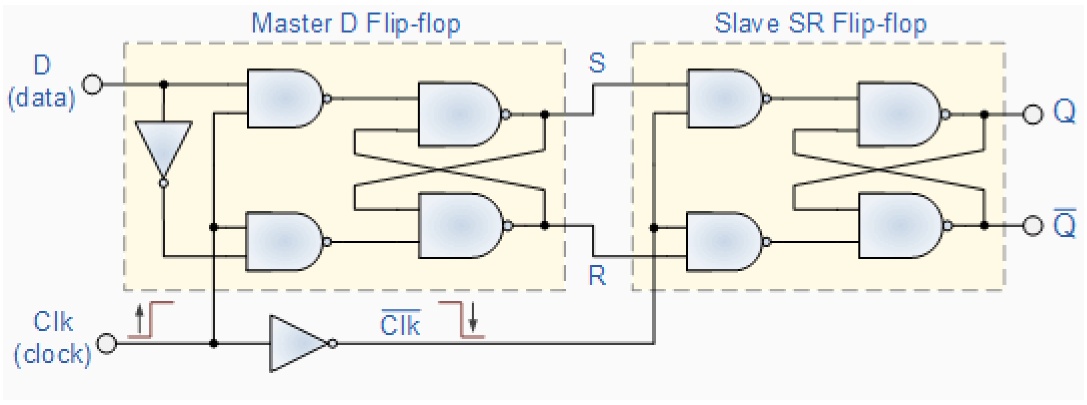
Clk	D	Q	Q	Description
↓ » 0	X	Q	Q	Memory no change
↑ » 1	0	0	1	Reset Q » 0
↑ » 1	1	1	0	Set Q » 1

Note: ↓ and ↑ indicates direction of clock pulse as it is assumed D flip-flops are edge triggered

The Master-Slave D Flip-flop

The basic **D flip-flop** can be improved further by adding a second SR flip-flop to its output that is activated on the complementary clock signal to produce a "Master-Slave D flip-flop". On the leading edge of the clock signal (LOW-to-HIGH) the first stage, the "master" latches the input condition at D, while the output stage is deactivated. On the trailing edge of the clock signal (HIGH-to-LOW) the second "slave" stage is now activated, latching on to the output from the first master circuit. Then the output stage appears to be triggered on the negative edge of the clock pulse. "Master-Slave D flip-flops" can be constructed by the cascading together of two latches with opposite clock phases as shown.

Master-Slave D flip-flop Circuit



We can see from above that on the leading edge of the clock pulse the master flip-flop will be loading data from the data D input, therefore the master is "ON". With the trailing edge of the clock pulse the slave flip-flop is loading data, i.e. the slave is "ON". Then there will always be one flip-flop "ON" and the other "OFF" but never both the master and slave "ON" at the same time. Therefore, the output Q acquires the value of D, only when one complete pulse, i.e. 0-1-0 is applied to the clock input.

There are many different D flip-flop IC's available in both TTL and CMOS packages with the more common being the 74LS74 which is a Dual D flip-flop IC, which contains two individual D type bistable's within a single chip enabling single or master-slave toggle flip-flops to be made. Other D flip-flop IC's include the 74LS174 HEX D flip-flop with direct clear input, the 74LS175 Quad D flip-flop with complementary outputs and the 74LS273 Octal D flip-flop containing eight D flip-flops with a clear input in one single package.

4.0 CONCLUSION

In this unit various types of sequential logic circuits were discussed. These include flip flop and latches.

5.0 SUMMARY

You have learnt that:

Latches are bistable devices whose state normally depends on asynchronous inputs and is a bistable digital circuit used for storing a bit
A latch is a level sensitive device. Because of this the state of the latch may keep changing in circuits with feedback as long as the clock pulse remains active.
Thus, instead of having output change once in a clock cycle, the output may change a number of times resulting in latching of unwanted input to the output. Due to this uncertainty, latches can not be reliably used as storage elements.
To overcome this problem of undesired toggling, we need to have a mechanism in which we have higher degree of control on the output of the memory element when the clock pulse changes. This is achieved by introducing a special clock-edge detection logic, such that the state of the memory element is switched by a momentary change in the clock pulse (i.e. an edge). This is effective because the clock changes

only once during a clock period. Such a memory element is "edge-sensitive", i.e., it changes its state at the rising or falling edge of a clock. Edge-sensitive memory elements are called Flip-Flops

Edge-triggered flip-flops are bistable devices with synchronous inputs whose state depends on the inputs only at the triggering transition of a clock pulse. Changes in the outputs occur at the triggering transition of the clock.

D flip-flop is a type of bistablemultivibrator in which the output assumes the state of the D input on the triggering edge of a clock pulse.

Edge-triggered flip-flop is a type of flip-flop in which the data are entered and appear on the output on the same clock edge.

Setup time (T_s) refers to a constant duration for which the inputs must be held prior to the arrival of the clock transition

(i). Hold time(T_h)is the time interval required for the control levels to remain on the inputs to a flip-flop after the triggering edge of the clock in order to reliably activate the device.

J-K flip-flop is a type of flip-flop that can operate in the SET. RESET. no-change, and toggle modes.

SELF ASSESSMENT QUESTION

1. What is a flip-flop? What is the difference between a latch and a flip-flop? List out the application of flip-flop

6.0 TUTOR-MARKED ASSIGNMENT

1. Give the truth table of S-R and D-flipflops. Convert the given S-R flipflop to a D-flipflop

2. What is a flip-flop? Write the truth table for a clocked J-K flip-flop that is triggered by the positive-going edge of the clock signal.

3. With relevant diagram explain the working of master-slave JK flip flop

7.0 REFERENCES/FURTHER READING

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,

Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.

Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall

Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

www.doc.ic.ac.uk/~dfg/hardware/HardwareLecture03.pd

www.doc.ic.ac.uk/~dfg/hardware/HardwareHandout02.pdf

www.itee.uq.edu.au/~engg1030/lectures/1perpage/lect14.pdf
http://www.indiabix.com/digital-electronics/combinational-logic-circuits/116006

UNIT 2

MULTIVIBRATOR

CONTENT

1.0 INTRODUCTION

This unit introduces multivibrator which is a class of digital circuits.

2.0 OBJECTIVES

At the end of this unit you should be able to:

Expalin the meaning of multivibrators
List the different types of multivibrators
Describe each type and give example

3.0 MAIN CONTENT

3.1 Multivibrators

Individual **Sequential Logic** circuits can be used to build more complex circuits such as Multivibrators, Counters, Shift Registers, Latches and Memories etc, but for these types of circuits to operate in a "sequential" way, they require the addition of a clock pulse or timing signal to cause them to change their state. Clock pulses are generally continuous square or rectangular shaped waveform that is produced by a single pulse generator circuit such as a **Multivibrator**.

Multivibrator is a class of digital circuits in which the output is connected back to the input (an arrangement called feedback) to produce either two stable states, one stable state, or no stable states, depending on the configuration.

They are used as waveform generators to produce the clock signals to switch sequential circuits.Thismultivibrator circuit oscillates between a "HIGH" state and a "LOW" state producing a continuous output. Astablemultivibrators generally have an even 50% duty cycle, that is that 50% of the cycle time the output is "HIGH" and the remaining 50% of the cycle time the output is "OFF". In other words, the duty cycle for an astable timing pulse is 1:1.

What are Multivibrators used to produce?

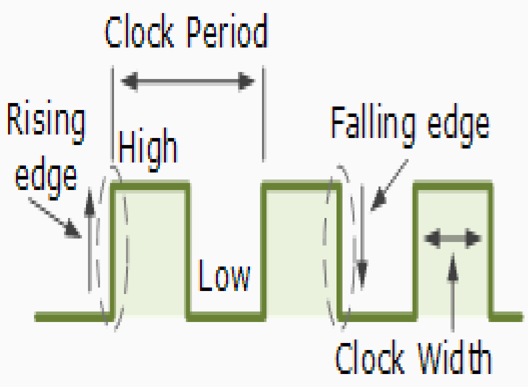
Multivibrators are used to as pulse generators to produce continuous square or rectangular shaped waveform.

Sequential logic circuits that use the clock signal for synchronization are dependant upon the frequency and and clock pulse width to activate there switching action. Sequential circuits may also change their state on either the rising or falling edge, or both of the actual clock signal as we have seen previously with the basic flip-flop circuits. The following lists are terms associated with a timing pulse or waveform.

Active HIGH - if the state changes occur at the clock's rising edge or during the clock width.

Active LOW - if the state changes occur at the clock's falling edge.

Duty Cycle - is the ratio of clock width and clock period.



Clock Signal Waveform

Clock Width - this is the time during which the value of the clock signal is equal to one.

Clock Period - this is the time between successive transitions in the same direction, i.e., between two rising or two falling edges.

Clock Frequency - the clock frequency is the reciprocal of the clock period, frequency = 1/clock period

Clock pulse generation circuits can be a combination of analogue and digital circuits that produce a continuous series of pulses (these are called astablemultivibrators) or a pulse of a specific duration (these are called monostablemultivibrators). Combining two or more of multivibrators provides generation of a desired pattern of pulses (including pulse width, time between pulses and frequency of pulses).

There are basically three types of clock pulse generation circuits:

Astable - A *free-running multivibrator* that has **NO** stable states but switches continuously between two states this action produces a train of square wave pulses at a fixed frequency.

Monostable - A *one-shot multivibrator* that has only **ONE** stable state and is triggered externally with it returning back to its first stable state.

Bistable - A *flip-flop* that has **TWO** stable states that produces a single pulse either positive or negative in value.

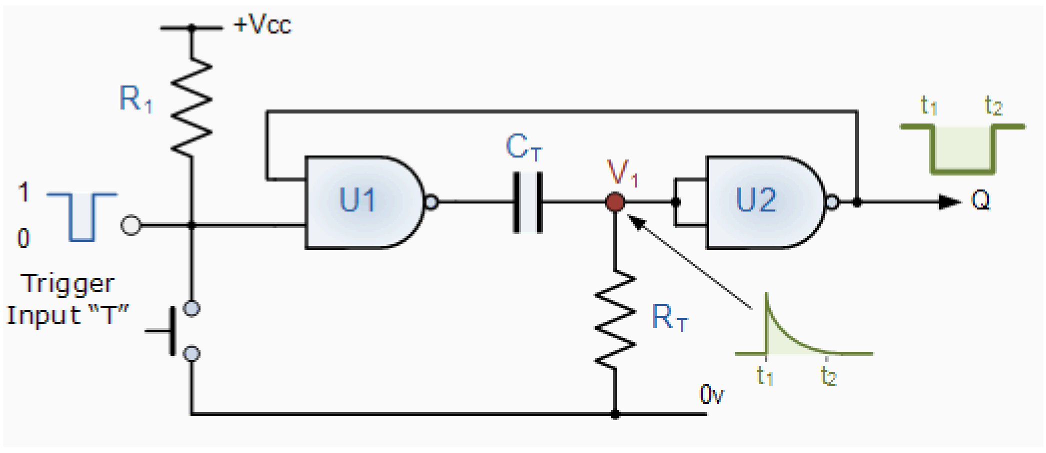
One way of producing a very simple clock signal is by the interconnection of logic gates. As NAND gates contains amplification, they can also be used to provide a clock signal or timing pulse with the aid of a single **Capacitor, C** and **Resistor, R** which provide the feedback and timing function. These timing circuits are often used because of there simplicity and are also useful if a logic circuit is designed that has un-used gates which can be utilised to create the monostable or astable oscillator. This simple type of RC Oscillator network is sometimes called a "Relaxation Oscillator".

3.2 Monostable Circuits.

MonostableMultivibrators or "one-shot" pulse generators are used to convert short sharp pulses into wider ones for timing applications. Monostablemultivibrators generate a single output pulse, either "high" or "low", when a suitable external trigger signal or pulse T is applied. This trigger pulse signal initiates a timing cycle which causes the output of the monostable to change state at the start of the timing cycle, (t_1) and remain in this second state until the end of the timing period, (t_1) which is determined by the time constant of the timing capacitor, C_T and the resistor, R_T.

The monostablemultivibrator now stays in this second timing state until the end of the RC time constant and automatically resets or returns itself back to its original (stable) state. Then, a monostable circuit has only one stable state. A more common name for this type of circuit is simply a "Flip-Flop" as it can be made from two cross-coupled NAND gates (or NOR gates) as we have seen previously. Consider the circuit below.

Simple NAND Gate Monostable Circuit



Suppose that initially the trigger input T is held HIGH at logic level "1" by the resistor R_1 so that the output from the first NAND gate U1 is LOW at logic level "0", (NAND gate principals). The timing resistor, R_T is connected to a voltage level equal to logic level "0", which will cause the capacitor, C_T to be discharged. The output of U1 is LOW, timing capacitor C_T is completely discharged therefore junction V1 is also equal to "0" resulting in the output from the second NAND gate U2, which is connected as an inverting NOT gate will therefore be HIGH.

Monostablemultivibrators are also known as ________?

MonostableMultivibrators are also known as one shot Timer

The output from the second NAND gate, (U2) is fed back to one input of U1 to provide the necessary positive feedback. Since the junction V1 and the output of U1 are both at logic "0" no current flows in the capacitor C_T. This results in the circuit being **Stable** and it will remain in this state until the trigger input T changes.

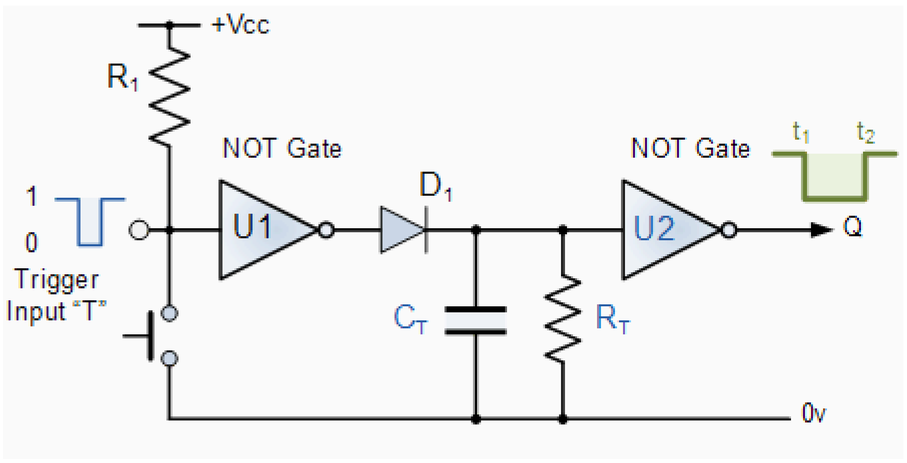
If a negative pulse is now applied either externally or by the action of the push-button to the trigger input of the NAND gate U1, the output of U1 will go HIGH to logic "1" (NAND gate principles). Since the voltage across the capacitor cannot change instantaneously (capacitor charging principals) this will cause the junction at V1 and also the input to U2 to also go HIGH, which inturn will make the output of the NAND gate U2 change LOW to logic "0" The circuit will now remain in this second state even if the trigger input pulse T is removed. This is known as the **Meta-stable** state.

The voltage across the capacitor will now increase as the capacitor C_T starts to charge up from the output of U1 at a time constant determined by the resistor/capacitor combination. This charging process continues until the charging current is unable to hold the input of U2 and therefore junction V1 HIGH. When this happens, the output of U2 switches HIGH again, logic "1", which inturn causes the output of U1 to go LOW and the capacitor discharges into the output of U1 under the influence of resistor R_T. The circuit has now switched back to its original stable state.

Thus for each negative going trigger pulse, the monostablemultivibrator circuit produces a LOW going output pulse. The length of the output time period is determined by the capacitor/resistor combination (**RC Network**) and is given as the **Time Constant** T = 0.69RC of the circuit in seconds. Since the input impedance of the NAND gates is very high, large timing periods can be achieved.

As well as the NAND gate monostable type circuit above, it is also possible to build simple monostable timing circuits that start their timing sequence from the rising-edge of the trigger pulse using NOT gates, NAND gates and NOR gates connected as inverters as shown below.

NOT Gate Monostable Circuit



As with the NAND gate circuit above, initially the trigger input T is HIGH at a logic level "1" so that the output from the first NOT gate U1 is LOW at logic level "0". The timing resistor, R_T and the capacitor, C_T are connected together in parallel and also to the input of the second NOT gate U2. As the input to U2 is LOW at logic "0" its output at Q is HIGH at logic "1".

When a logic level "0" pulse is applied to the trigger input T of the first NOT gate it changes state and produces a logic level "1" output. The diode D1 passes this logic "1" voltage level to the RC timing network. The voltage across the capacitor, C_T increases rapidly to this new voltage level, which is also connected to the input of the second NOT gate. This inturn outputs a logic "0" at Q and the circuit stays in this **Meta-stable** state as long as the trigger input T applied to the circuit remains LOW.

When the trigger signal returns HIGH, the output from the first NOT gate goes LOW to logic "0" (NOT gate principals) and the fully charged capacitor, C_T starts to discharge itself through the parallel resistor, R_T connected across it. When the voltage across the capacitor drops below the lower threshold value of the input to the second NOT gate, its output switches back again producing a logic level "1" at Q. The diode D1 prevents the timing capacitor from discharging itself back through the first NOT gates output.

Then, the **Time Constant** for a NOT gate **MonostableMultivibrator** is given as T = 0.8RC + Trigger in seconds.

One main disadvantage of **MonostableMultivibrators** is that the time between the application of the next trigger pulse T has to be greater than the RC time constant of the circuit.

What are the basic three types of clock pulse generation circuits?

Astable - A *free-running multivibrator* that has **NO** stable states but switches continuously between two states this action produces a train of square wave pulses at a fixed frequency.

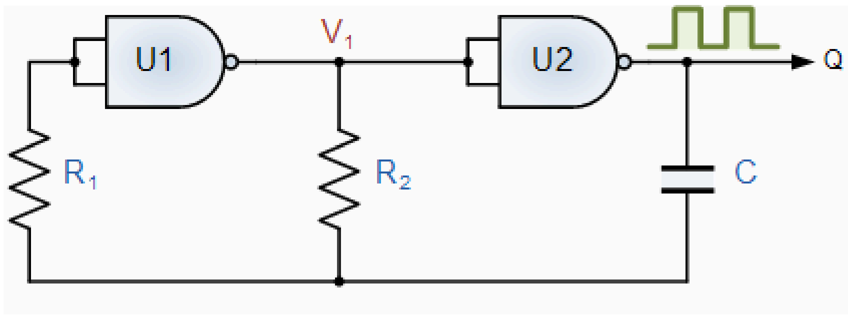
Monostable - A *one-shot multivibrator* that has only **ONE** stable state and is triggered externally with it returning back to its first stable state.

Bistable - A *flip-flop* that has **TWO** stable states that produces a single pulse either positive or negative in value.

3.3 Astable Circuits.

AstableMultivibrators are a type of free running oscillator that have no permanent "meta" or "steady" state but are continually changing there output from one state ("LOW") to the other state ("HIGH") and then back again. This continual switching action from "HIGH" to "LOW" and "LOW" to "HIGH" produces a continuous and stable square wave output that switches abruptly between the two logic levels making it ideal for timing and clock pulse applications. As with the monostablemultivibrator circuit above, the timing cycle is determined by the time constant of the resistor-capacitor, **RC Network**. Then the output frequency can be varied by changing the value(s) of the resistors and capacitor in the circuit.

NAND Gate AstableMultivibrators



The **astablemultivibrator** circuit uses two CMOS NOT gates such as the CD4069 or the 74HC04 hex inverter ICs, or as in our simple circuit below a pair of CMOS NAND such as the CD4011 or the 74LS132 and an RC timing network. The two NAND gates are connected as inverting NOT gates.

Suppose that initially the output from the NAND gate U2 is HIGH at logic level "1", then the input must therefore be LOW at logic level "0" (NAND gate principles) as will be the output from the first NAND gate U1. Capacitor, C is connected between the output of the second NAND gate U2 and its input via the timing resistor, R_2. The capacitor now charges up at a rate determined by the time constant of R_2 and C.

As the capacitor, C charges up, the junction between the resistor R_2 and the capacitor, C, which is also connected to the input of the NAND gate U1 via the stabilizing resistor, R_2 decreases until the lower threshold value of U1 is reached at which point U1 changes state and the output of U1 now becomes HIGH. This causes NAND gate U2 to also change state as its input has now changed from logic "0" to logic "1" resulting in the output of NAND gate U2 becoming LOW, logic level "0".

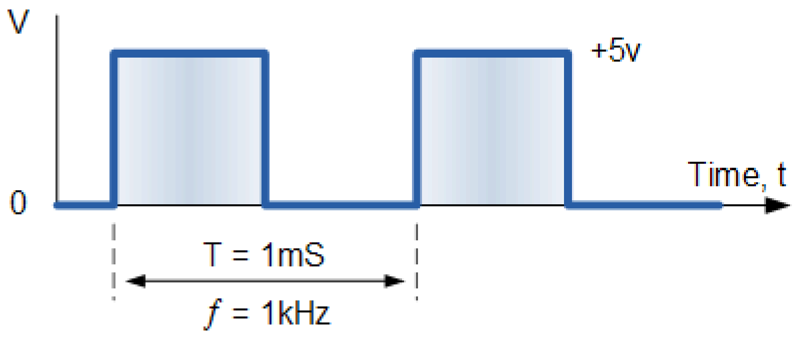
Capacitor C is now reverse biased and discharges itself through the input of NAND gate U1. Capacitor, C charges up again in the opposite direction determined by the time constant of both R_2 and C as before until it reaches the upper threshold value of NAND gate U1. This causes U1 to change state and the cycle repeats itself over again.

Then, the time constant for a NAND gate **AstableMultivibrator** is given as T = 2.2RC in seconds with the output frequency given as f = 1/T.

For example: if resistor R_2 = 10kΩ and the capacitor C = 45nF, then the oscillation frequency will be given as:

$$f = \frac{1}{T} = \frac{1}{2.2RC} = \frac{1}{2.2 \times 10k\Omega \times 45nF} = 1kHz$$

then the output frequency is calculated as being 1kHz, which equates to a time constant of 1mS so the output waveform would look like:



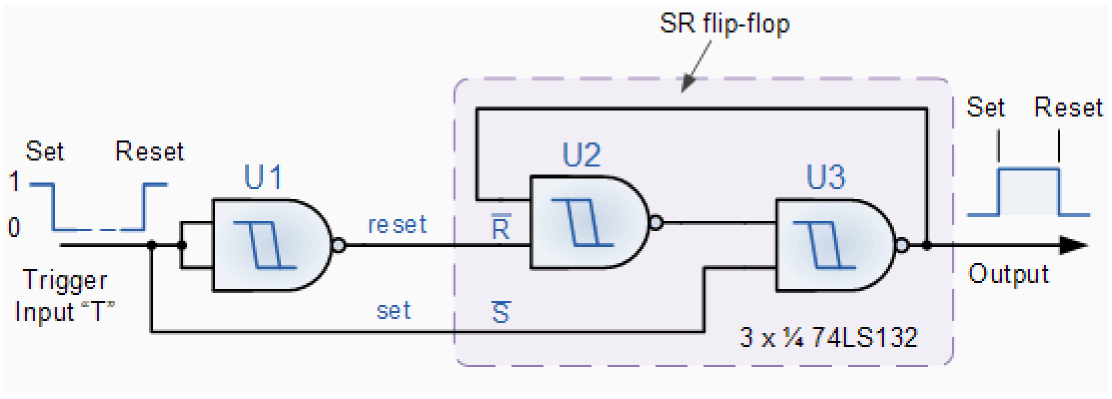
What is the basic difference monostable and Astable multivaibrators?

The bsic difference is that the monostable as a permanent state and a one shot timer while the Astable Monostable as no permanent state, but continually changes states.

3.4 Bistable Circuits.

The **BistableMultivibrators** circuit is basically a SR flip-flop that we look at in the previous tutorials with the addition of an inverter or NOT gate to provide the necessary switching function. As with flip-flops, both states of a bistablemultivibrator are stable, and the circuit will remain in either state indefinitely. This type of multivibrator circuit passes from one state to the other "only" when a suitable external trigger pulse T is applied and to go through a full "SET-RESET" cycle **two** triggering pulses are required. This type of circuit is also known as a "**Bistable Latch**", "**Toggle Latch**" or simply "**T-latch**".

NAND Gate BistableMultivibrator



The simplest way to make a **Bistable Latch** is to connect together a pair of Schmitt NAND gates to form a SR latch as shown above. The two NAND gates, U2 and U3 form the bistable which is triggered by the input NAND gate, U1. This U1 NAND gate can be omitted and replaced by a single toggle switch to make a switch debounce circuit as seen previously in the **SR Flip-flop** tutorial. When the input pulse goes "LOW" the bistable latches into its "SET" state, with its output at logic level "1", until the input goes "HIGH" causing the bistable to latch into its "RESET" state, with its output at logic level "0". The output of a bistablemultivibrator will stay in this "RESET" state until another input pulse is applied and the whole sequence will start again.

Then a **Bistable Latch** or "Toggle Latch" is a two-state device in which both states either positive or negative, (logic "1" or logic "0") are stable.

BistableMultivibrators have many applications such as frequency dividers, counters or as a storage device in computer memories but they are best used in circuits such as **Latches** and **Counters**.

SELF ASSESSMENT QUESTION

What are the three types of multivibrator?

4.0 CONCLUSION

This unit describesmultivibrator, its type and functions.

5.0 SUMMARY

In this unit the following aspects have been discussed:

Monostablemultivibrarors (one-shots) have one stable state. When the one-shot is triggered, the output goes to its unstable state for a time determined by an RC circuit. Astablemultivibrators have no stable states and are used as oscillators to generate timing waveforns in digital systems. It oscillates between two quasi-stable states. Bistable Having two stable states. Flip-flops and latches are bistablemultivibrators.

6.0 TUTOR-MARKED ASSIGNMENT

Describe the mode of operation of the monostablemultivibrator

7.0 REFERENCES/FURTHER READING

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,
Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.
Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall
Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice
http://www.tech.mtu.edu/~alaraje/.../EET2141_Outline_Fall2009.pdf
http://www.courses.cs.tamu.edu/rabi/.../Lecture_2_DD_vahid_ch1.p
http://www.en.wikibooks.org/wiki/Microprocessor_Design/Introduction
http://www.engr.sjsu.edu/tle/120syl.pdf
http://www.cs.ucla.edu/Logic_Design
http://www.allaboutcircuits.com

STUDY UNIT 3

SHIFT REGISTER AND COUNTER

CONTENT

1.0 INTRODUCTION

Shift registers are a type of sequential logic circuit, mainly for storage ofdigital data. They are a group of flip-flops connected in a chain so thatthe output from one flip-flop becomes the input of the next flip-flop.Most of the registers possess no characteristic internal sequence of states.All flip-flopsis driven by a common clock, and all are set or resetsimultaneously.

In this unit, the basic types of shift registers are studied, such asSerial In - Serial Out, Serial In - Parallel Out, Parallel In – Serial Out,Parallel In - Parallel Out, and bidirectional shift registers. A special formof counter - the shift register counter, is also introduced.

2.0 OBJECTIVES

At the end of this unit you should be able to:

Identify the basic forms of data movement in shift registers
Explain how serial in/serial out, serial in/parallel out, parallel inlserial out, and parallel in/parallel out shift registers operate
Describe how a bidirectional shift register operates
Determine the sequence of a Johnson counter
Set up a ring counter to produce a specified sequence
Construct a ring counter from a shift register

3.1 The Shift Register

The **Shift Register** is another type of sequential logic circuit that is used for the storage or transfer of data in the form of binary numbers and then "shifts" the data out once every clock cycle, hence the name "shift register". It basically consists of several single bit "D-Type Data Latches", one for each bit (0 or 1) connected together in a serial or daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on. The data bits may be fed in or out of the register serially, i.e. one after the other from either the left or the right direction, or in parallel, i.e. all together. The number of individual data latches required to make up a single **Shift Register** is determined by the number of bits to be stored with the most common being 8-bits wide, i.e. eight individual data latches.

The Shift Register is used for data storage or data movement and are used in calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. The individual data latches that make up a single shift register are all driven by a common clock (Clk) signal making them synchronous devices. Shift register IC's are generally provided with a *clear* or *reset* connection so that they can be "SET" or "RESET" as required.

What type of Logic Circuit are the shift registers

The Shift registers are a type of sequential Logic circuit

Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register being:

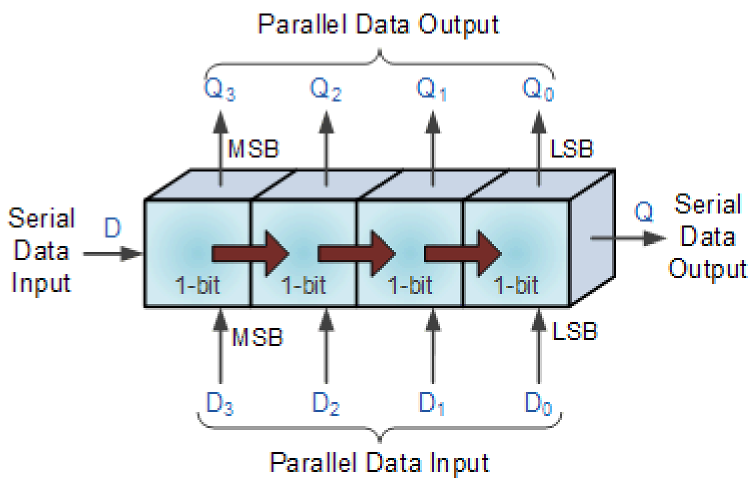
Serial-in to Parallel-out (SIPO) - the register is loaded with serial data, one bit at a time, with the stored data being available in parallel form.

Serial-in to Serial-out (SISO) - the data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.

Parallel-in to Serial-out (PISO) - the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.

Parallel-in to Parallel-out (PIPO) - the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.

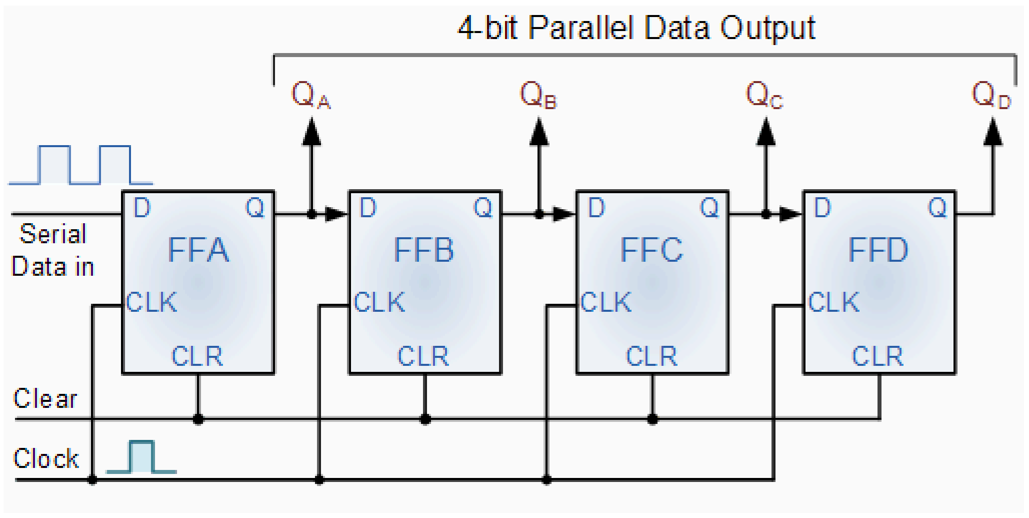
The effect of data movement from left to right through a shift register can be presented graphically as:



Also, the directional movement of the data through a shift register can be either to the left, (left shifting) to the right, (right shifting) left-in but right-out, (rotation) or both left and right shifting within the same register thereby making it *bidirectional*. In this tutorial it is assumed that all the data shifts to the right, (right shifting).

3.2 Serial-in to Parallel-out (SIPO)

4-bit Serial-in to Parallel-out Shift Register



The operation is as follows. Lets assume that all the flip-flops (FFA to FFD) have just been RESET (CLEAR input) and that all the outputs Q_A to Q_D are at logic level "0" i.e, no parallel data output. If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting Q_A will be set HIGH to logic "1" with all the other outputs still remaining LOW at logic "0". Assume now that the DATA input pin of FFA has returned LOW again to logic "0" giving us one data pulse or 0-1-0.

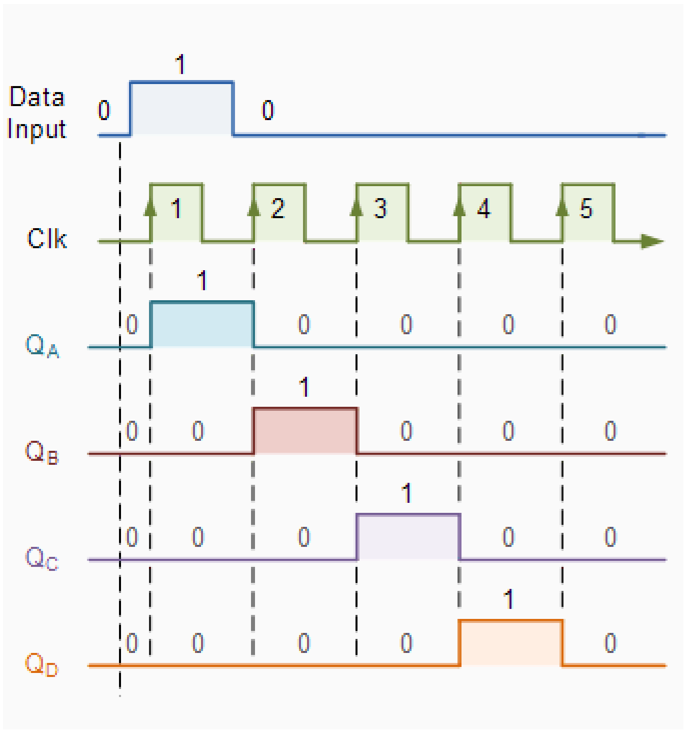
The second clock pulse will change the output of FFA to logic "0" and the output of FFB and Q_B HIGH to logic "1" as its input D has the logic "1" level on it from Q_A. The logic "1" has now moved or been "shifted" one place along the register to the right as it is now at Q_A.

When the third clock pulse arrives this logic "1" value moves to the output of FFC (Q_C) and so on until the arrival of the fifth clock pulse which sets all the outputs Q_A to Q_D back again to logic level "0" because the input to FFA has remained constant at logic level "0".

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the following table until the complete data value of 0-0-0-1 is stored in the register. This data value can now be read directly from the outputs of Q_A to Q_D. Then the data has been converted from a serial data input signal to a parallel data output. The truth table and following waveforms show the propagation of the logic "1" through the register from left to right as follows.

Basic Movement of Data through a Shift Register

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0



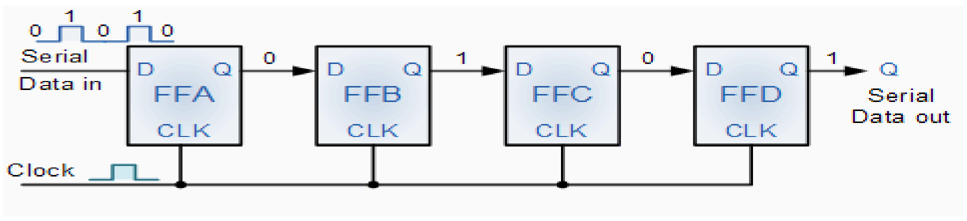
Note that after the fourth clock pulse has ended the 4-bits of data (0-0-0-1) are stored in the register and will remain there provided clocking of the register has stopped. In practice the input data to the register may consist of various combinations of logic "1" and "0". Commonly available SIPO IC's include the standard 8-bit 74LS164 or the 74LS594.

3.3 Serial-in to Serial-out (SISO)

This **shift register** is very similar to the SIPO above, except were before the data was read directly in a parallel form from the outputs Q_A to Q_D, this time the data is allowed to flow straight through the register and out of the other end. Since there is only one output, the DATA leaves the shift register one bit at a time in a serial pattern, hence the name **Serial-in to Serial-Out Shift Register** or **SISO**.

The SISO shift register is one of the simplest of the four configurations as it has only three connections, the serial input (SI) which determines what enters the left hand flip-flop, the serial output (SO) which is taken from the output of the right hand flip-flop and the sequencing clock signal (Clk). The logic circuit diagram below shows a generalized serial-in serial-out shift register.

4-bit Serial-in to Serial-out Shift Register

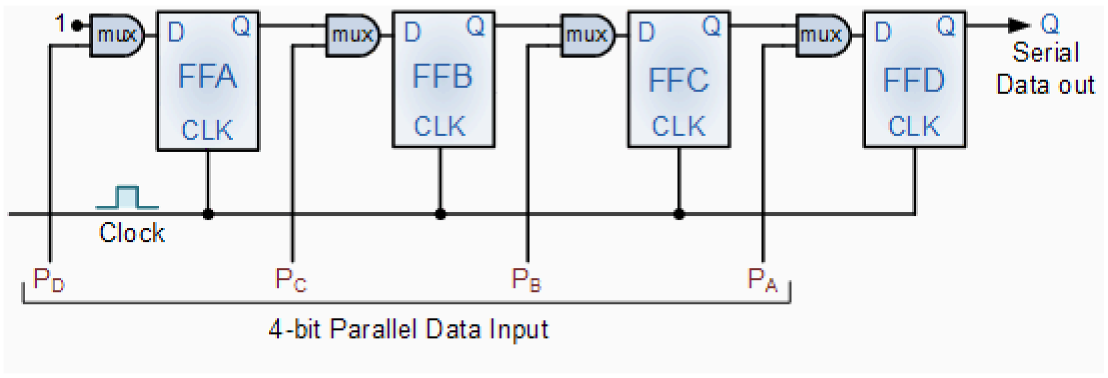


You may think what's the point of a SISO shift register if the output data is exactly the same as the input data. Well this type of **Shift Register** also acts as a temporary storage device or as a time delay device for the data, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc or by varying the application of the clock pulses. Commonly available IC's include the 74HC595 8-bit Serial-in/Serial-out Shift Register all with 3-state outputs.

3.4 Parallel-in to Serial-out (PISO)

The Parallel-in to Serial-out shift register acts in the opposite way to the serial-in to parallel-out one above. The data is loaded into the register in a parallel format i.e. all the data bits enter their inputs simultaneously, to the parallel input pins P_A to P_D of the register. The data is then read out sequentially in the normal shift-right mode from the register at Q representing the data present at P_A to P_D. This data is outputted one bit at a time on each clock cycle in a serial format. It is important to note that with this system a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data.

4-bit Parallel-in to Serial-out Shift Register

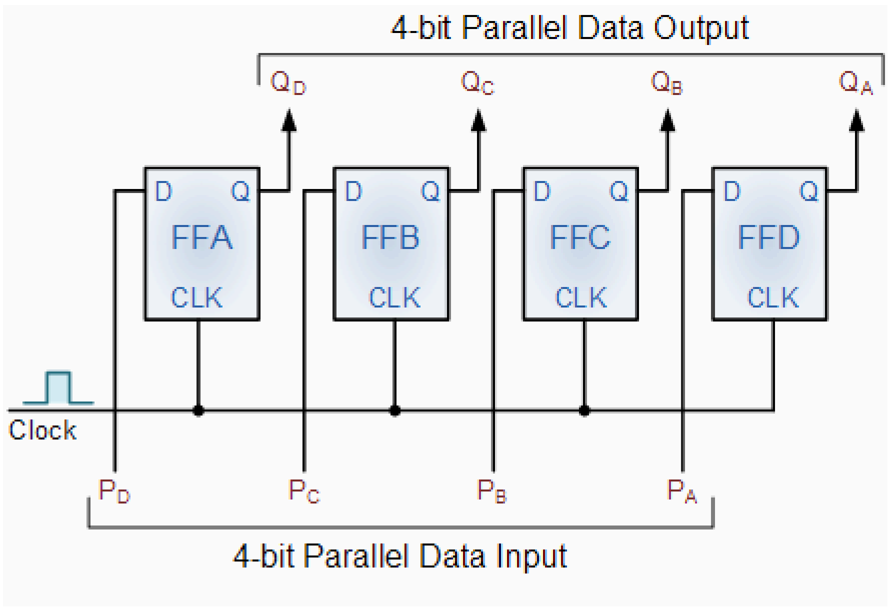


As this type of shift register converts parallel data, such as an 8-bit data word into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line. Commonly available IC's include the 74HC166 8-bit Parallel-in/Serial-out Shift Registers.

3.5 Parallel-in to Parallel-out (PIPO)

The final mode of operation is the Parallel-in to Parallel-out Shift Register. This type of register also acts as a temporary storage device or as a time delay device similar to the SISO configuration above. The data is presented in a parallel format to the parallel input pins P_A to P_D and then transferred together directly to their respective output pins Q_A to Q_A by the same clock pulse. Then one clock pulse loads and unloads the register. This arrangement for parallel loading and unloading is shown below.

4-bit Parallel-in to Parallel-out Shift Register



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

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

What are the various modes of operation of the shift registers.

The various modes of operation includes

Serial-in to Parallel-out (SIPO) - the register is loaded with serial data, one bit at a time, with the stored data being available in parallel form.

Serial-in to Serial-out (SISO) - the data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.

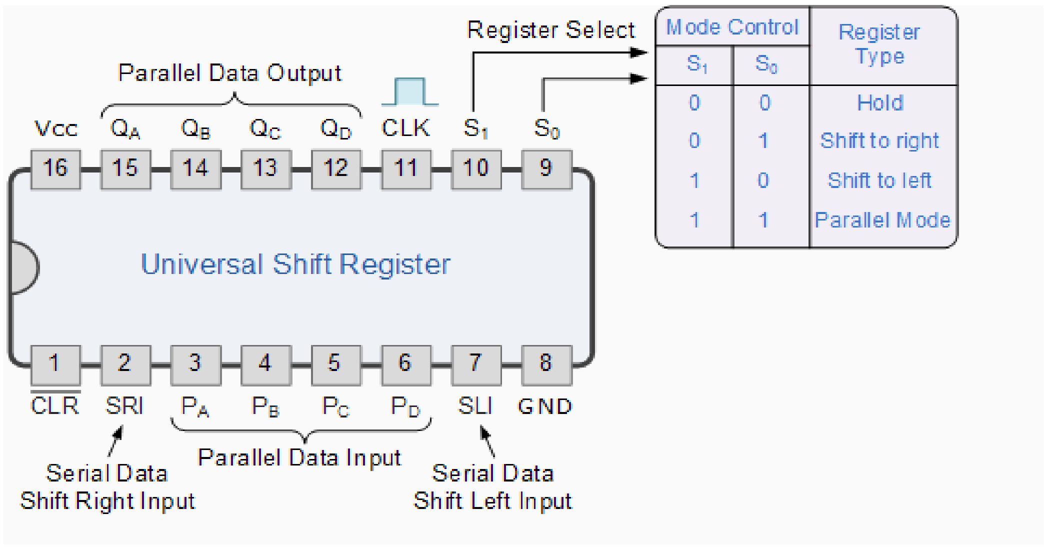
Parallel-in to Serial-out (PISO) - the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.

Parallel-in to Parallel-out (PIPO) - the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.

3.6 Universal Shift Register

The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations.A universal bidirectional, or reversible, shift register is one in which the data can be shift either left or right.Today, high speed bi-directional "universal" type **Shift Registers** such as the TTL 74LS194, 74LS195 or the CMOS 4035 are available as a 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, and as a parallel-to-parallel multifunction data register, hence the name "Universal". These devices can perform any combination of parallel and serial input to output operations but require additional inputs to specify desired function and to pre-load and reset the device.

4-bit Universal Shift Register 74LS194



Mode Control		Register Type
S_1	S_0	
0	0	Hold
0	1	Shift to right
1	0	Shift to left
1	1	Parallel Mode

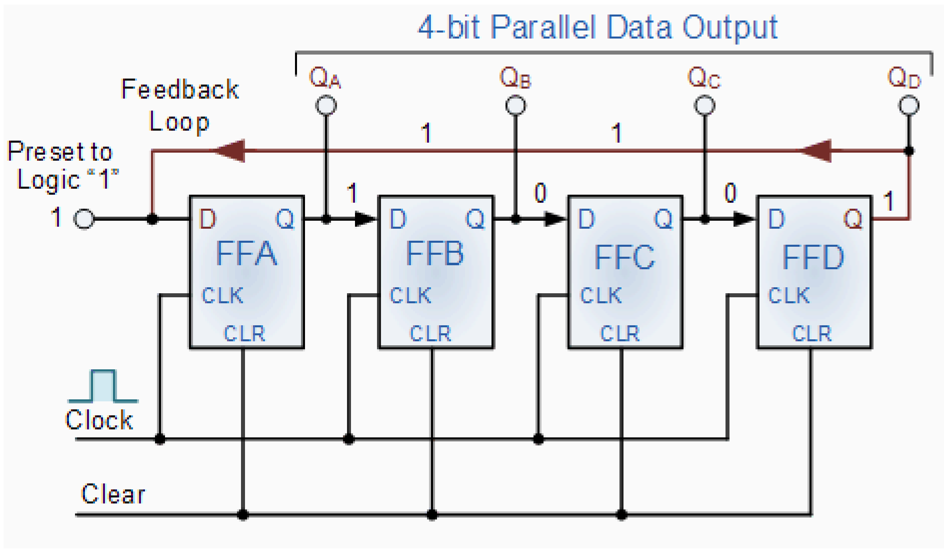
Universal shift registers are very useful digital devices. They can be configured to respond to operations that require some form of temporary memory, delay information such as the SISO or PIPO configuration modes or transfer data from one point to another in either a serial or parallel format. Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.

3.7 The Ring Counter

In the previous lesson on **Shift Register** we saw that if we apply a serial data signal to the input of a *serial-in to serial-out shift register*, the same sequence of data will exit from the last flip-flip in the register chain after a preset number of clock cycles thereby acting as a sort of time delay circuit to the original signal.

But what if we were to connect the output of this shift register back to its input so that the output from the last flip-flop, Q_D becomes the input of the first flip-flop, D_A. We would then have a closed loop circuit that "recirculates" the DATA around a continuous loop for every state of its sequence, and this is the principal operation of a **Ring Counter**. Then by looping the output back to the input, we can convert a standard shift register into a ring counter. Consider the circuit below.

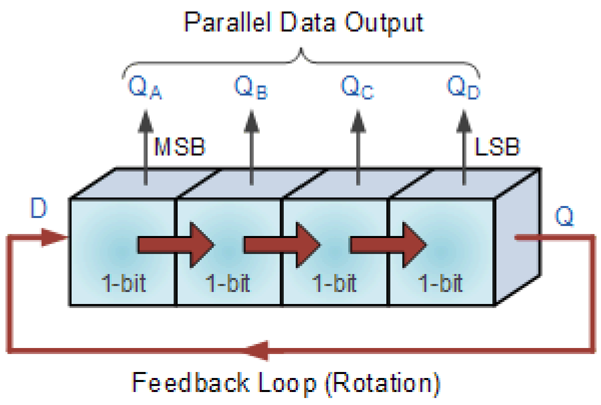
4-bit Ring Counter



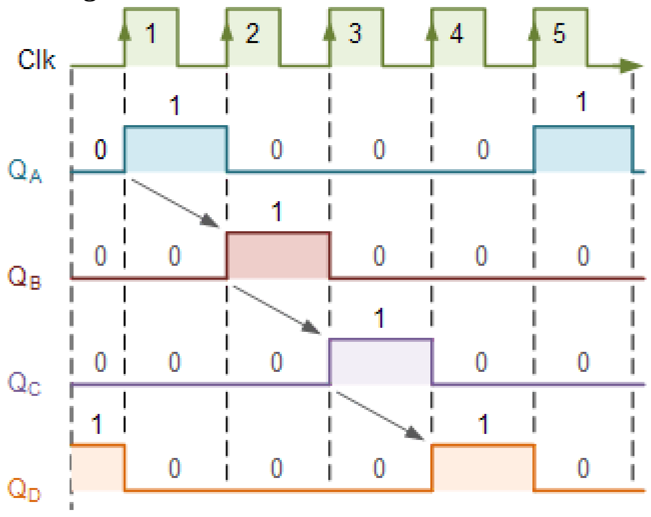
The synchronous **Ring Counter** example above, is preset so that exactly one data bit in the register is set to logic "1" with all the other bits reset to "0". To achieve this, a "CLEAR" signal is firstly applied to all the flip-flops together in order to "RESET" their outputs to a logic "0" level and then a "PRESET" pulse is applied to the input of the first flip-flop (FFA) before the clock pulses are applied. This then places a single logic "1" value into the circuit of the ring counter . On each successive clock pulse, the counter circulates the same data bit between the four flip-flops over and over again around the "ring" every fourth clock cycle. But in order to cycle the data correctly around the counter we must first "load" the counter with a suitable data pattern as all logic "0"'s or all logic "1"'s outputted at each clock cycle would make the ring counter invalid.

This type of data movement is called "rotation", and like the previous shift register, the effect of the movement of the data bit from left to right through a ring counter can be presented graphically as follows along with its timing diagram:

Rotational Movement of a Ring Counter



Timing diagram



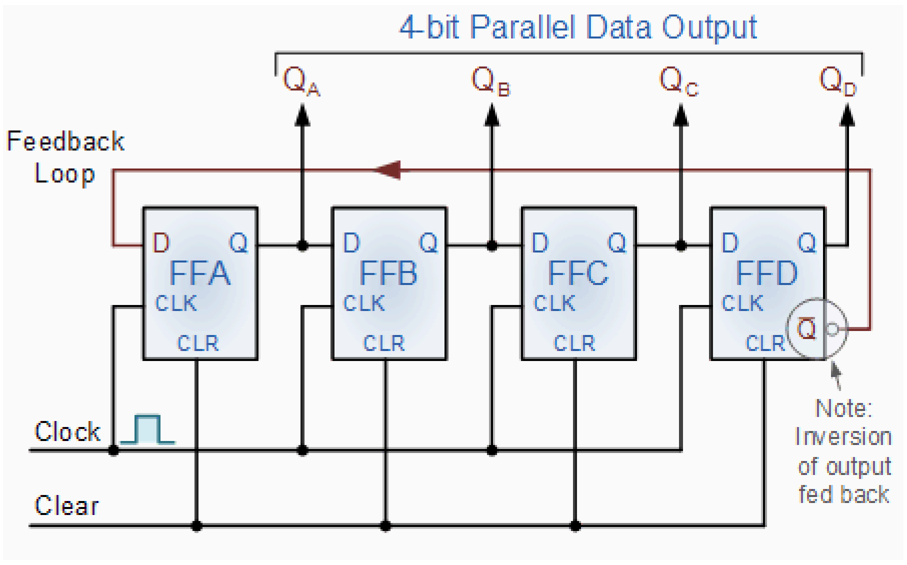
Since the ring counter example shown above has four distinct states, it is also known as a "modulo-4" or "mod-4" counter with each flip-flop output having a frequency value equal to one-fourth or a quarter (1/4) that of the main clock frequency.

The "MODULO" or "MODULUS" of a counter is the number of states the counter counts or sequences through before repeating itself and a ring counter can be made to output any modulo number. A "mod-n" ring counter will require "n" number of flip-flops connected together to circulate a single data bit providing "n" different output states. For example, a mod-8 ring counter requires eight flip-flops and a mod-16 ring counter would require sixteen flip-flops. However, as in our example above, only four of the possible sixteen states are used, making ring counters very inefficient in terms of their output state usage.

Johnson Ring Counter

The **Johnson Ring Counter** or "Twisted Ring Counters", is another shift register with feedback exactly the same as the standard *Ring Counter* above, except that this time the inverted output Q of the last flip-flop is now connected back to the input D of the first flip-flop as shown below. The main advantage of this type of ring counter is that it only needs half the number of flip-flops compared to the standard ring counter then its modulo number is halved. So a "n-stage" Johnson counter will circulate a single data bit giving sequence of 2n different states and can therefore be considered as a "mod-2n counter".

4-bit Johnson Ring Counter



This inversion of Q before it is fed back to input D causes the counter to "count" in a different way. Instead of counting through a fixed set of patterns like the normal ring counter such as for a 4-bit counter, "0001"(1), "0010"(2), "0100"(4), "1000"(8) and repeat, the Johnson counter counts up and then down as the initial logic "1" passes through it to the right replacing the preceding logic "0". A 4-bit Johnson ring counter passes blocks of four logic "0" and then four logic "1" thereby producing an 8-bit pattern. As the inverted output Q is connected to the input D this 8-bit pattern continually repeats. For example, "1000", "1100", "1110", "1111", "0111", "0011", "0001", "0000" and this is demonstrated in the following table below.

Truth Table for a 4-bit Johnson Ring Counter

Clock Pulse No	FFA	FFB	FFC	FFD
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

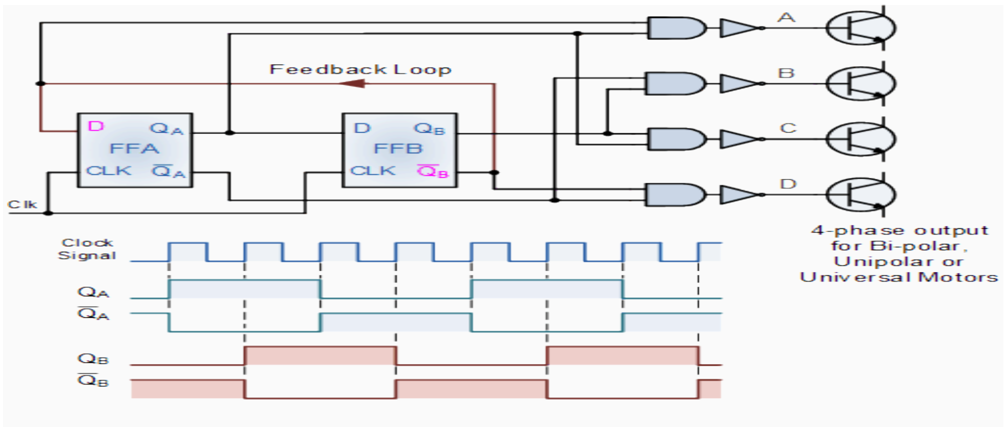
How different is the Johnson Ring Counter from the standard Ring Counter

The Johnson Ring Counter or "Twisted Ring Counters", is another shift register with feedback exactly the same as the standard *Ring Counter* above, except that this time the inverted output Q of the last flip-flop is now connected back to the input D of the first flip-flop as shown below. The main advantage of this type of ring counter is that it only needs half the number of flip-flops compared to the standard ring counter then its modulo number is halved.

As well as counting or rotating data around a continuous loop, ring counters can also be used to detect or recognise various patterns or number values within a set of data. By connecting simple logic gates such as the **AND** or the **OR** gates to the outputs of the flip-flops the circuit can be made to detect a set number or value. Standard 2, 3 or 4-stage Johnson ring counters can also be used to divide the frequency of the clock signal by varying their feedback connections and divide-by-3 or divide-by-5 outputs are also available.

A 3-stage Johnson Ring Counter can also be used as a 3-phase, 120 degree phase shift square wave generator by connecting to the data outputs at A, B and NOT-B. The standard 5-stage Johnson counter such as the commonly available CD4017 is generally used as a synchronous decade counter/divider circuit. The smaller 2-stage circuit is also called a "Quadrature" (sine/cosine) Oscillator/Generator and is used to produce four individual outputs that are each "phase shifted" by 90 degrees with respect to each other, and this is shown below.

2-bit Quadrature Generator

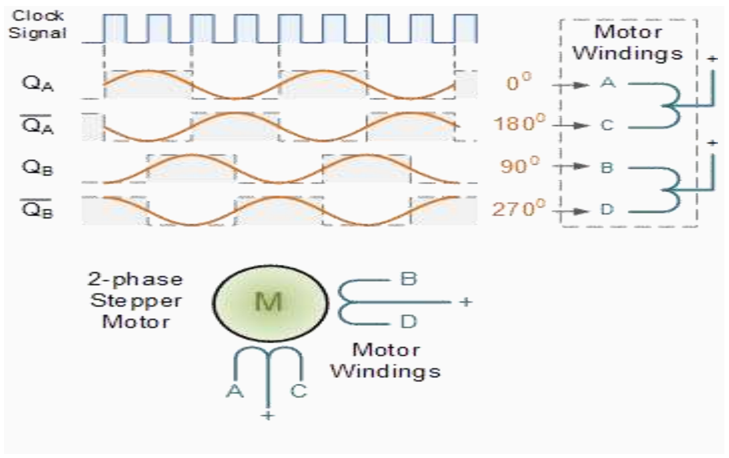


Output	A	B	C	D
Q_A+Q_B	1	0	0	0
Q_A+Q_B	0	1	0	0
Q_A+Q_B	0	0	1	0
Q_A+Q_B	0	0	0	1

2-bit Quadrature Oscillator, Count Sequence

As the four outputs, A to D are phase shifted by 90 degrees with regards to each other, they can be used with additional circuitry, to drive a 2-phase full-step stepper motor for position control or the ability to rotate a motor to a particular location as shown below.

Stepper Motor Control



2-phase (unipolar) Full-Step Stepper Motor Circuit

The speed of rotation of the **Stepper Motor** will depend mainly upon the clock frequency and additional circuitry would be required to drive the "power" requirements of the motor. As this section is only intended to give the reader a basic understanding of **Johnson Ring Counters** and its applications, other good websites explain in more detail the types and drive requirements of stepper motors.

Johnson Ring Counters are available in standard TTL or CMOS IC form, such as the CD4017 5-Stage, decade Johnson ring counter with 10 active HIGH decoded outputs or the CD4022 4-stage, divide-by-8 Johnson counter with 8 active HIGH decoded outputs.

SELFASSESSMENT QUESTION

1 What is a shift register? Can a shift register be used as a counter? If yes, explain how?

2. What is a Shift Register? What are its various types? List out some applications of Shift Register.

5.0 CONCLUSION

This unit discusses shift register and shift register counter which are a type of sequential logic circuit mainly for storage of digital data.

5.0 SUMMARY

You have learnt that::

A register is a digital circuit with two basic functions: data storage and data move ment.

Shift Registeris used to convert parallel data into serial data and vice versa.

A simple **Shift Register** can be made using only D-type flip-Flops, one flip-Flop for each data bit.

Shift registers hold the data in their memory which is moved or "shifted" to their required positions on each clock pulse.

Each clock pulse shifts the contents of the register one bit position to either the left or the right.

The SISO shift register accepts data serially-that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

SIPO Data bits are entered serially (right-most bit first) into this type of register in the same manner as in SISO. The difference is the way in which the data bits are taken out of the register; in the parallel output register. the output of each stage is availahle. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output.

PISO For a register with parallel data inputs, the bits are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis on one line as with serial data inputs. The serial output is the same as SISO, once the data are completely stored in the register.

PIPO the parallel in/parallel out register employs both methods for parallel in and parallel out. Immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs.

The data bits can be loaded one bit at a time in a series input (SI) configuration or be loaded simultaneously in a parallel configuration (PI).

Data may be removed from the register one bit at a time for a series output (SO) or removed all at the same time from a parallel output (PO).

One application of shift registers is converting between serial and parallel data.

Johnson counter the complement of the output of the last flip-flop is connected back to the D input of the first flip-flop (it can be implemented with other types of flip-flops as well).

The ring counter utilizes one flip-flop for each state in its sequence. It has the advantage that decoding gates are not required.

6.0 TUTOR-MARKED ASSIGNMENT

1. Describe the operation of parallel in parallel out (PIPO) shift register.
2. Using D-Flip flops and waveforms explain the working of a 4-bit SISO shift register
3. Design a 3-bit shift register which has 4 operating modes.
4. Define a register. Construct a shift register from S-R flip-flops. Explain its working.

7.0 REFERENCES/FURTHER READING

Ronald J. T. & Neal S., (2001). Widmer Digital Systems: Principle and Applications (8th Ed.) Prentice Hall,

Thomas L F., (2006). Digital Fundamentals (9th Ed.). Prentice Hall.

Morris M. & Charles R. K. (2004) Logic and Computer Design Fundamentals. (2004) NJPrentice Hall

Wakerly J.F. (2000). Digital Design: Principles and Practices (3rd Ed.) Upper Saddle River NJ; Prentice

http://www.cs.ucla.edu/Logic_**Design**

http://www.allaboutcircuits.com

https://maxwell.ict.griffith.edu.au/yg/teaching/.../dns_module3_p3.pdf..

http://www.ce.rit.edu/studentresources/reference.../341/.../EECC341-08.pdf

http://www.techterms.com/definition/integratedcircuit

ANSWERS TO SELF ASSESSMENT QUESTIONS

MODULE 1

UNIT 1

SAQ 1: Blaise Pascal invents an adding machine to relieve the tedium of adding up long columns of tax figures.

Gottfried Leibniz invents the first mechanical calculator capable of multiplication.

Charles Babbagedesigns a complex, clockwork calculator capable of solving equations and printing the results

UNIT 2

1. 8 247

8 30 R 7 $\uparrow$

8 3 R 6

0 3 R 3

= 367_8

2. (95.5)10 = (5F.8)16

Integer part Fractional part

16 95 0.5x16=8.0

16 5 R 15 $\uparrow$

0 5 R 5

=$5F.8_{16}$

3. $(10001011.011)_2$=2^7+2^3+2^1+2^0+2^{-2}+2^{7-3}

=139.375

4. 567_8= 101110111_2

= 0001/0111/0111

= 177_{16}

5. $FA_{16=}$ 11111010_2

UNIT 3

1 (a) A byte is 8 bits

1 1 1 1 1 0 1 1
1 1 1 1 0 1 1 0
1 1 1 0 1 1 0 0

1 (b), By using 2's compliment

1 0 1 0 1 0 1 1=

-128 64 32 16 8 4 2 1

1 0 1 0 1 0 1 1

= -128 +32+8+2+1

= -85

By using signed magnitude

0 1 0 1 0 1 1
64 32 16 8 4 2 1

1 0 1 0 1 0 1 1

= -(32+8+2+1)

= -43

2

The gray code is 101 00 10 1
The binary equivalence is 11001010

MODULE 2

UNIT 1:

SAQ 1

(XYZ)' = X'+Y'+Z'

X	Y	Z	XYZ	(XYZ)'	X'+Y'+Z'
0	0	0	0	1	1
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	0	0

X+YZ =(X+Y).(X+Z)

X	Y	Z	YZ	X+YZ	X+Y	X+Z	(X+Y)(X+Z)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

SAQ: 2

The Identify and label variables are as follows

p=1: person in front of door

h=1: held open manually

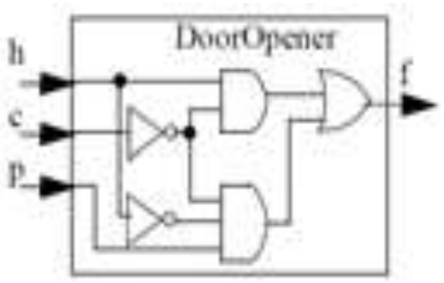
c=1: force door to stay closed

f=1: open sliding door

Write Boolean Equation expressing functionality described: not forced close and manually held open, or not forced closed and not manually held open and person detected

f = hc’ + h’pc’

The circuit is shown below



UNIT 2

Solutions to SAQ

1. $\overline{\overline{A + BC} + \overline{A\overline{B}}}$

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

$\downarrow$ Breaking longest bar

$$\overline{\overline{(A + BC)}}\;\;\overline{\overline{(A\overline{B})}}$$

$\downarrow$ Applying identity $\overline{\overline{A}} = A$ wherever double bars of equal length are found

$$(A + BC)\,(A\overline{B})$$

$\downarrow$ Distributive property

$$AA\overline{B} + BCA\overline{B}$$

$\downarrow$ Applying identity $AA = A$ to left term; applying identity $A\overline{A} = 0$ to B and $\overline{B}$ in right term

$$A\overline{B} + 0$$

$\downarrow$ Applying identity $A + 0 = A$

$$A\overline{B}$$

2. $\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

$\overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

$\downarrow$ Factoring BC out of 1st and 4th terms

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

$\downarrow$ Applying identity $A + \overline{A} = 1$

$BC(1) + A\overline{B}C + AB\overline{C}$

$\downarrow$ Applying identity $1A = A$

$BC + A\overline{B}C + AB\overline{C}$

$\downarrow$ Factoring B out of 1st and 3rd terms

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

$\downarrow$ Applying rule $A + \overline{A}B = A + B$ to the $C + A\overline{C}$ term

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

$\downarrow$ Distributing terms

$BC + AB + A\overline{B}C$

$\downarrow$ Factoring A out of 2nd and 3rd terms

$BC + A(B + \overline{B}C)$

$\downarrow$ Applying rule $A + \overline{A}B = A + B$ to the $B + \overline{B}C$ term

$BC + A(B + C)$

$\downarrow$ Distributing terms

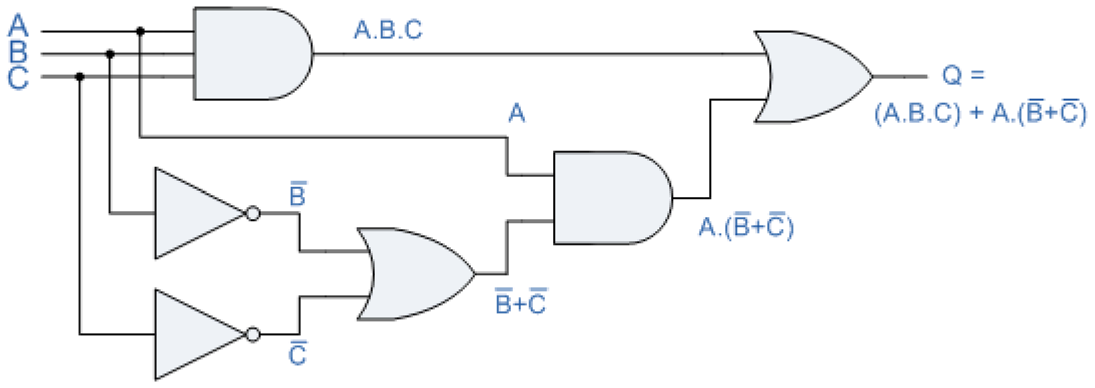
$BC + AB + AC$ Simplified result

or

$AB + BC + AC$

.

SAQ2


A
B
C
A.B.C
A
$\overline{B}$
$\overline{C}$
$\overline{B}+\overline{C}$
$A.(\overline{B}+\overline{C})$
Q =
$(A.B.C) + A.(\overline{B}+\overline{C})$

UNIT 3

1

(i) Each individual term in standard Sum Of Products form is called as minterm whereas each individual term in standard Product Of Sums form is called maxterm.

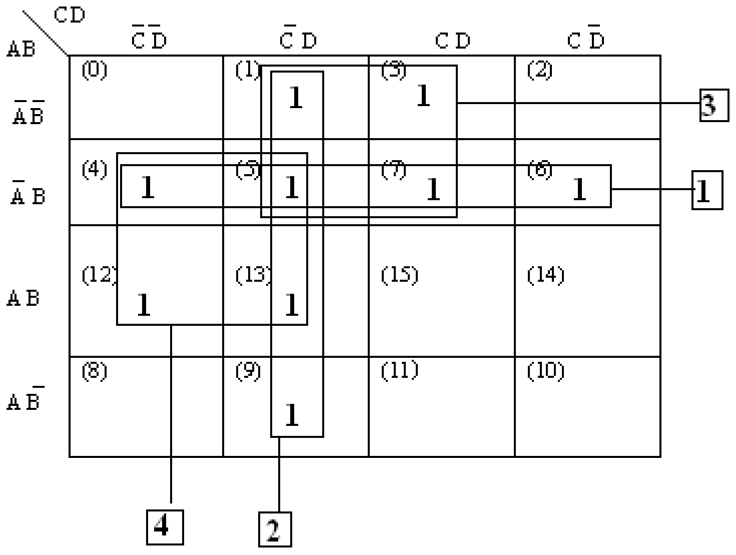
(ii) The unbarred letter represent 1’s and the barred letter represent 0’s in min terms,whereas the unbarred letter represent 0’s and the barred represent 1’s in maxterms.

(iii) If a system has variables A, B, C then the minterms would be in the form ABC,whereas the maxterm would be in the form A+B+C.

(iv) The minterm designation for three variable expression beY=Σm (1, 3, 5, 7) Where the capital Σ represents the product and m stands for minterms Whereas the Maxterm designation for three variable expression be Y=∏M (0, 1, 3, 4) Where the capital ∏ represents the product and M stands for maxterms

ANSWERS TO SAQ

Karnaugh Map for the expression F(A,B,C,D) = S (1,3,4,5,6,7,9,12,13) is shown The grouping of cells is also shown in the Figure.



The equations for (1) is *A*’B; (2) is *C’* D; (3) is *A*’D; (4) is B*C’*

Hence, the Simplified Expression for the above Karnaugh map is

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

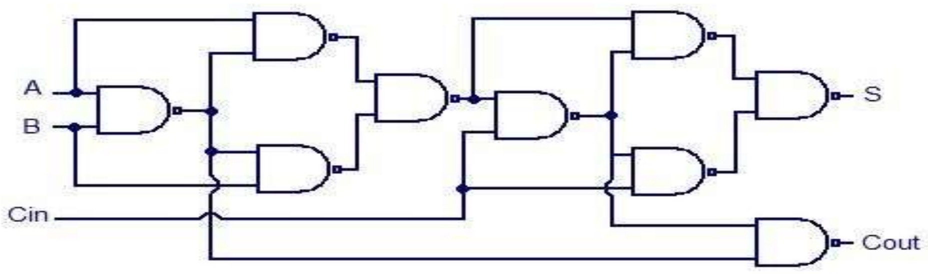
= *A'* (B + D) +*C'* (B + D)

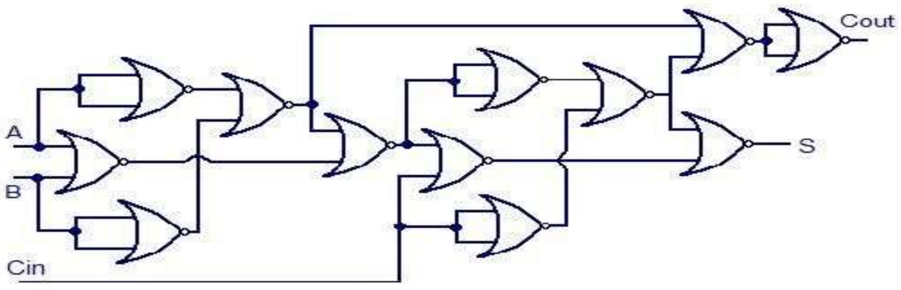
MODULE 3

UNIT 1

ANSWERS TO SAQ

Full adder using NAND and NOR gates





UNIT 2

Answers to SAQ

This function can be implemented with an 8-to-1 line MUX

A, B, and C are applied to the select inputs as follows:

A ⇒S2 , B ⇒S1, C ⇒S0

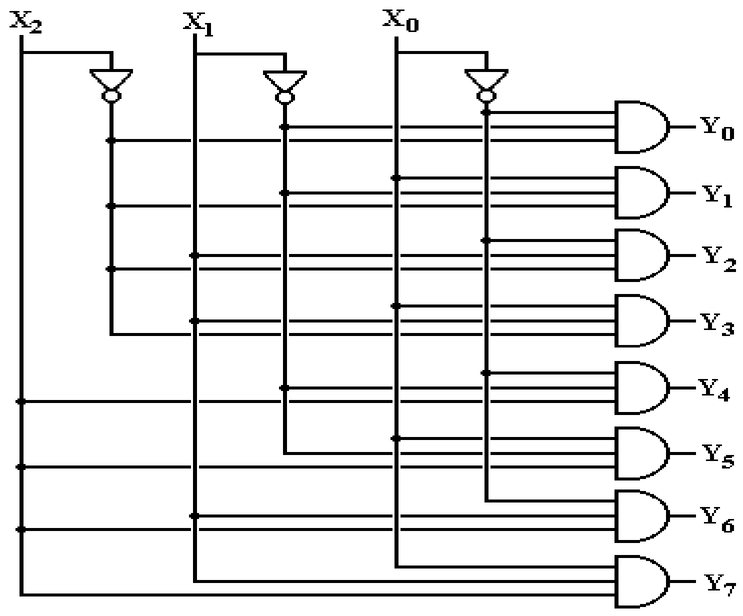
The truth for its implementation and its implementation are shown below

A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = $\overline{D}$
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	

UNIT 3

SAQ 1

1. a 3 to 8 decoder



2. (1) Active high output type decoders are constructed constructed with AND gates and will give the output high for given input combination and all other output are low . While Active low output type of decoders will give the output low for given input combination and all other outputs are high. They are constructed with NAND gates

SAQ2

Octal to binary encoder consists of eight inputs, one for each of eight digits and three outputs that generate the corresponding binary number. For example: low order output bit Z is if the input octal digit is odd. Here DO input is not connected to any O R gate; the bin

ary output must be all zeroes in this case and all 0's output is also obtained, when all inputs are zeroes. This discrepancy can be resolved by providing one more output to indicate the fact that all inputs are not zeroes.

MODULE 4

UNIT 1

Flip-Flop: A flip-flop is a basic memory element used to store one bit of information. Both Flip-flops and latches are bistable logic circuits and can reside in any of the two stable states due to a feedback arrangement. The main difference between them is in the method used for changing the state

Applications of Flop-Flops:

(1) Bounce elimination switch

(2) Parallel Data Storage in Registers

(3) Transfer of Data from one bit to another.

(4) Counters

(5) Frequency Division

UNIT 2

Monostable, Astable, and bistable

UNIT 3

ANSWER

1. A register in which data gets shifted towards left or right when clock pulses are applied is known as a Shift Register. A shift register can be used as a counter. If the output of a shift register is fed back to serial input, then the shift register can be used as a Ring Counter.

2. **Shift Register:** A register in which data gets shifted towards left or right when clockpulses are applied is known as a Shift Register.

Types of Shift Registers:

(i) Serial-In Serial-Out (SISO) Shift Register

(ii) Serial-In Parallel Out (SIPO) Shift Register

(iii) Parallel-In Serial Out (PISO) Shift Register

(iv) Parallel-In Parallel Out (PIPO) Shift Register

Applications of Shift Registers:

(i) Serial to Parallel Converter

(ii) Parallel to Serial Converter

(iii)Delay line

(iv) Ring Counter

(v) Twisted-ring Counter

(vi) Sequence Generator