COURSE GUIDE

IFT 211
DIGITAL AND LOGIC DESIGN

Course Team Dr. Kehinde Adebola Sotonwa (Course Developer/Writer) – Lagos State University, Ojo, Lagos State
Prof Joshua Abah (Course Editor) – Nile University, Abuja

NATIONAL OPEN UNIVERSITY OF NIGERIA

© 2025 by NOUN Press
National Open University of Nigeria
Headquarters
University Village
Plot 91, Cadastral Zone
Nnamdi Azikiwe Expressway
Jabi, Abuja

Lagos Office
14/16 Ahmadu Bello Way
Victoria Island, Lagos

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

First Printed 2008

Reprinted 2025

ISBN: 978-058-470-6

CONTENTS **PAGE**

COURSE INTRODUCTION

IFT 211 - Digital and Logic Design: Welcome to Digital and Logic Design, This course provides a comprehensive overview of digital logic fundamentals, covering essential topics ranging from number systems and logic gates to sequential circuits and programmable logic devices. Through a combination of theoretical lectures, hands-on activities, and practical assignments, you will go into the world of digital systems design and gain the necessary skills to analyse, design, and implement digital circuits.

WHAT YOU WILL LEARN IN THIS COURSE

In this course, you will learn the foundational principles of digital and logic design, including:

- Understanding different number systems and base conversions.
- Exploring complement systems and codes used in digital systems.
- Analysing and designing digital logic gates circuits.
- Mastering Boolean algebra and its application in digital circuit design.
- Learning about canonical and standard forms for Boolean expressions.
- Utilizing minimization techniques such as Karnaugh maps.
- Understanding the physical properties of logic gates.
- Designing combinational circuits and implementing design procedures.
- Exploring sequential circuits, including flip-flops, latches, registers, and counters.
- Studying memory devices classification and their applications.
- Learning about programmable logic devices such as PLAs, PLDs, and FPGAs.

COURSE AIM

The aim of this course is to provide you with a solid foundation in digital and logic design principles, equipping you with the knowledge and skills necessary to analyse, design, and implement digital circuits for a variety of applications.

COURSE OBJECTIVES

By the end of this course, you will be able to:

1. Understand the fundamentals of number systems and perform base conversions.
2. Analyse and design digital logic gates circuits.

3. Apply Boolean algebra techniques to simplify logic expressions.
4. Utilize minimization techniques, including Karnaugh maps, to optimize digital circuits.
5. Design combinational and sequential circuits, including flip-flops, latches, registers, and counters.
6. Explore memory devices classification and their applications in digital systems.
7. Program and configure programmable logic devices such as PLAs, PLDs, and FPGAs for various digital logic applications.

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 tutor - marked assignments, and at some point in this course, you are required to submit the tutor marked assignments.

COURSE JUSTIFICATION

A comprehensive study of digital logic design is essential for understanding the foundational principles of computer engineering and electronics. This course covers key areas such as number systems, digital logic gates, Boolean algebra, minimization techniques, and the design and analysis of combinational and sequential circuits. Starting with the basics of number systems and Boolean algebra, students will learn to work with digital logic gates and understand canonical and standard forms. Minimization techniques like the Karnaugh Map method and the study of the physical properties of gates will help optimize digital circuits.

The course then explores combinational circuits, including binary subtractors, multiplexers, decoders, and encoders, as well as sequential circuit elements like latches and flip-flops. Further, it covers the design and analysis of more complex sequential circuits, including flip-flop conversion, registers, and counters. Finally, students will explore various memory devices and programmable logic, including programmable logic arrays, devices, and field-programmable gate arrays. The theoretical and practical knowledge gained from this course will provide a solid foundation, enabling students to appreciate the relevance and interrelationships of different digital logic concepts, preparing them for advanced studies and practical applications in digital technology.

COURSE MATERIALS

The major components of the course are:

1. Course Guide
2. Study Units

3. Text Books
4. Assignment Files
5. Presentation Schedule

COURSE REQUIREMENTS

This is a compulsory course for all computer science students in the University. In view of this, students are expected to participate in all the course activities and have minimum of 75% attendance to be able to write the final examination.

STUDY UNITS

There are 5 modules and 24 study units in this course. They are:

Module 1: Introduction to Digital Logic Design

Unit 1: Introduction to Number Systems and Base Conversion
Unit 2: Complement Systems and Codes
Unit 3: Digital Logic Gates
Unit 4: Boolean Algebra
Unit 5: Canonical and Standard Forms

Module 2: Minimization Techniques

Unit 1: Karnaugh Map Method
Unit 2: Manipulation and Minimisation
Unit 3: Physical Properties of Gates

Module 3: Combinational and Sequential Circuits

Unit 1: Combinational Circuits and Design Procedure
Unit 2: Binary Subtractor
Unit 3: Multiplexers
Unit 4: De-multiplexers
Unit 5: Decoders
Unit 6: Encoders
Unit 7: Latches
Unit 8: Flip-Flops

Module 4: Sequential Circuits

Unit 1: Sequential Circuits
Unit 2: Conversion of Flip-Flops
Unit 3: Registers

Unit 4 Counters

Module 5: Memory Devices and Programmable Logic

Unit 1: Memory Devices and Classification
Unit 2: Programmable Logic Array (PLAs)
Unit 3: Programmable Logic Devices (PLDs)
Unit 4: Field-Programmable Gate Arrays (FGPAs)

TEXTBOOKS AND REFERENCES

M. Morris Mano. Digital Logic and Computer Design.

Dr. Muhamed Mudawar. Digital Logic Design

Department of Information Technology (2018). Digital Notes on Digital Logic Design.

Digital Circuits Number Systems https://www.tutorialspoint.com/digital_circuits/digital_circuits_number_systems.htm

Digital Circuits Base Conversions

https://www.tutorialspoint.com/digital_circuits/digital_circuits_base_conversions.htm
Digital Circuits Codes

https://www.tutorialspoint.com/digital_circuits/digital_circuits_codes.htm

Digital Circuits Error Detection Correction Codes

https://www.tutorialspoint.com/digital_circuits/digital_circuits_error_detection_correction_codes.htm

Digital Circuits Boolean Algebra

https://www.tutorialspoint.com/digital_circuits/digital_circuits_boolean_algebra.htm

Digital Circuits Canonical Standard Forms

https://www.tutorialspoint.com/digital_circuits/digital_circuits_canonical_standard_forms.htm

Digital Circuits K Map Method

https://www.tutorialspoint.com/digital_circuits/digital_circuits_k_map_method.htm

Digital Circuits Quine-McCluskey Tabular Method

https://www.tutorialspoint.com/digital_circuits/digital_circuits_quine_mccluskey_tabular_method.htm

Digital Circuits Logic Gates

https://www.tutorialspoint.com/digital_circuits/digital_circuits_logic_gates.htm

Digital Combinational Circuits

https://www.tutorialspoint.com/digital_circuits/digital_combinational_circuits.htm

Digital Arithmetic Circuits

https://www.tutorialspoint.com/digital_circuits/digital_arithmetic_circuits.htm

Digital Circuits Decoders
https://www.tutorialspoint.com/digital_circuits/digital_circuits_decoders.htm

Digital Circuits Encoders

https://www.tutorialspoint.com/digital_circuits/digital_circuits_encoders.htm

Digital Circuits Multiplexers
https://www.tutorialspoint.com/digital_circuits/digital_circuits_multiplexers.htm

Digital Circuits De-Multiplexers
https://www.tutorialspoint.com/digital_circuits/digital_circuits_demultiplexers.htm

Digital Circuits Programmable Logic Devices
https://www.tutorialspoint.com/digital_circuits/digital_circuits_programmable_logic_devices.htm

Digital Circuits Sequential Circuits
https://www.tutorialspoint.com/digital_circuits/digital_circuits_sequential_circuits.htm

Digital Circuits Latches
https://www.tutorialspoint.com/digital_circuits/digital_circuits_latches.htm

Digital Circuits Flip-Flops
https://www.tutorialspoint.com/digital_circuits/digital_circuits_flip_flops.htm

Digital Circuits Conversion of Flip-Flops
https://www.tutorialspoint.com/digital_circuits/digital_circuits_conversion_of_flip_flops.htm

Digital Circuits Shift Registers
https://www.tutorialspoint.com/digital_circuits/digital_circuits_shift_registers.htm

Digital Circuits Counters
https://www.tutorialspoint.com/digital_circuits/digital_circuits_counters.htm

Minimization of Boolean Functions
https://www.slideshare.net/blaircomp2003/minimization-of-boolean-functions-39058948

PLDs vs. FPGAs
https://www.ampheo.com/blog/plds-vs-fpgas-whats-the-difference-between-them.html

TMA on Memory Device and Programmable Logic
https://engweb.eng.wayne.edu/~ad5781/ECECourses/ECE2610/Lecture Notes/Lecture13.pdf
TMA on K Map

https://www.gatevidyalay.com/tag/karnaugh-map-questions-and-answers-pdf/

TMA on Digital Logic Design
https://www.geeksforgeeks.org/digital-logic-design-mcqs/

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 5 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 5 TMAs in this course. You need to submit all the TMAs. The best 3 will therefore be counted. The total marks for the best five (3) assignments will be 40% of your total course mark.

FINAL EXAMINATION AND GRADING

The final examination for the course will carry 60% percentage of the total marks available for this course. The examination will cover every aspect of the course, so you are advised to revise all your corrected assignments before the examination.

This course endows you with the status of a teacher and that of a learner. This means that you teach yourself and that you learn, as your learning capabilities would allow. It also means that you are in a better position to

determine and to ascertain the what, the how, and the when of your language learning. No teacher imposes any method of learning on you.

The course units are similarly designed with the introduction following the contents, then a set of objectives and then the dialogue and so on. The objectives guide you as you go through the units to ascertain your knowledge of the required terms and expressions.

COURSE MARKING SCHEME

The following table includes the course marking scheme

Table 1: Course Marking Scheme

Assessment	**Marks**
Assignments 1-5	5 assignments, 40% for the best 3 total = 8% X 5 = 40%
Final Examination	60% of overall course marks
Total	100% of Course Marks

COURSE OVERVIEW

This table indicates the units, the number of weeks required to complete them and the assignments.

Unit	**Title of the Work**	**Weeks**	**Assessment (End of Unit)**
	Course Guide		
	Module 1 Introduction to Digital Logic Design		
1	Introduction to Number Systems and Base Conversion	Week 1 and Week 2	Assessment 1
2	Complement Systems and Codes		
3	Digital Logic Gates		
4	Boolean Algebra		
5	Canonical and Standard Forms		
	Module 2 Minimization Techniques		
1	Karnaugh Map Method	Week 3 and Week 4	Assessment 2
2	Manipulation and Minimisation		
3	Physical Properties of Gates		
	Module 3 Combinational and Sequential Circuits		
1	Combinational Circuits and Design Procedure	Week 5 to Week 8	Assessment 3
2	Binary Subtractor		

3	Multiplexers		
4	De-multiplexers		
5	Decoders		
6	Encoders		
7	Latches		
8	Flip-Flops		
	Module 4 Sequential Circuits		
1	Sequential Circuits	Week 9 and Week 10	Assessment 4
2	Conversion of Flip-Flops		
3	Registers		
4	Counters		
	Module 5 Memory Devices and Programmable Logic		
1	Memory Devices and Classification	Week 11 and Week 12	Assessment 5
2	Programmable Logic Array (PLAs)		
3	Programmable Logic Devices (PLDs)		
4	Field-Programmable Gate Arrays (FGPAs)		

HOW TO GET THE BEST FROM THIS COURSE

In distance learning the study units replace the university lecturer. This is one of the great advantages of distance learning; you can read and work through specially designed study materials at your own pace, and at a time and place that suit you best. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some reading to do, the study units tell you when to read your set books or other material. Just as a lecturer might give you an in-class exercise, your study units provide exercises for you to do at appropriate points.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next is a set of learning objectives. These objectives enable you know what you should be able to do by the time you have completed the unit. You should use these objectives to guide your study. When you have finished the units you must go back and check whether you have achieved the objectives. If you make a habit of doing this, you will significantly improve your chances of passing the course.

Remember that your tutor's job is to assist you. When you need help, don't hesitate to call and ask your tutor to provide it.

1. Read this 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 assignments relate to the units. Whatever method you

chose to use, you should decide on it 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 lag behind in their course work.
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 Overview at the beginning of each unit. You will almost always need both the study unit you are working on and one of your set of books 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 instructed to read sections from your set books or other articles. Use the unit to guide your reading.
7. Review the objectives for each study unit to confirm that you have achieved them. If you feel unsure about any of the objectives, review the study material or consult your tutor.
8. When you are confident that you have achieved a unit's objectives, you can then start on the next unit. Proceed unit by unit through the course and try to pace your study so that you keep yourself on schedule.
9. 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 your tutor as soon as possible if you have any questions or problems.
10. 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 12 hours of tutorials provided in support of this course. You will be notified of the dates, times and location of these tutorials, 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 or submit your tutor-marked assignments to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, or e-mail 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-tests or exercises
- You have a question or problem with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

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

Course Structure and Specification

Modules and Units	ILOs: By the end of this unit, you will be able to:	Learning Activities	Teaching Technique	Required Hours for Study
Module 1				
Unit 1: Introduction to Number Systems and Base Conversion	• Understand the fundamentals of number systems including binary, decimal, octal, and hexadecimal. • Ability to convert numbers between different number systems.	• Lecture on the basics of number systems and their significance in digital logic. • Practice exercises on converting numbers between binary, decimal, octal, and hexadecimal systems.	• Visual aids illustrating the concept of different number systems. • Step-by-step demonstrations of base conversion methods.	Approximately 2 hours.
Unit 2: Complement Systems and Codes	• Understand the concepts of complement systems including ones' complement and twos' complement. • Familiarity with different codes such as binary coded	• Lecture on complement systems and their applications in digital logic. • Practice exercises on converting numbers to ones' complement and twos'	• Examples illustrating the concept of complement systems and their relevance. • Interactive activities for practicing code conversion.	Approximately 1.5 hours.

	decimal (BCD).	complement.		
Unit 3: Digital Logic Gates	• Understand the basic types of digital logic gates including AND, OR, NOT, NAND, NOR, and XOR. • Familiarity with truth tables and logic gate operation.	• Lecture on the different types of digital logic gates and their behaviour. • Hands-on practice constructing truth tables and analysing gate operations.	• Visual demonstrations of gate behavior using circuit diagrams. • Group activities for analyzing and designing logic circuits.	Approximately 2 hours.
Unit 4: Boolean Algebra	• Understand the fundamental concepts of Boolean algebra including basic theorems and laws. • Ability to simplify Boolean expressions using algebraic manipulation.	• Lecture on the principles of Boolean algebra and its application in digital logic. • Practice exercises on simplifying Boolean expressions using algebraic laws.	• Step-by-step examples of Boolean expression simplification. • Interactive exercises for practicing Boolean algebra manipulation.	Approximately 2.5 hours.
Unit 5: Canonical and Standard Forms	• Understand the concepts of canonical and standard forms of Boolean expressions.	• Lecture on canonical and standard forms and their significance in digital	• Visual aids illustrating the concept of canonical and standard forms. • Step-by-step demonstrations of converting	Approximately 2 hours.

		logic design. • Practice exercises on converting Boolean expressions to canonical and standard forms.	Boolean expressions.	
Module 2				
Unit 1: Karnaugh Map Method	• Understand the concept of Karnaugh maps as a graphical method for simplifying Boolean expressions. • Ability to use Karnaugh maps to minimize logic functions.	• Lecture on the principles of Karnaugh maps and their application in logic minimization. • Practice exercises on using Karnaugh maps to simplify logic expressions.	• Step-by-step examples of Karnaugh map simplification. • Guided practice sessions for solving Karnaugh map problems.	Approximately 2.5 hours.
Unit 2: Manipulation and Minimization	• Understand various methods for manipulating and minimizing Boolean expressions. • Ability to apply algebraic laws and theorems for logic	• Lecture on different techniques for manipulating and minimizing Boolean expressions. • Practice exercises on applying algebraic laws and	• Demonstrations of algebraic manipulation techniques for logic simplification. • Interactive sessions for practicing problem-solving skills.	Approximately 2.5 hours.

	simplification.	theorems for simplification.		
Unit 3: Physical Properties of Gates	• Understand the physical properties of logic gates including fan-in, fan-out, and propagation delay. • Familiarity with timing diagrams	• Lecture on the physical characteristics of logic gates and their impact on circuit performance. • Discussion on timing diagrams	• Visual demonstrations of gate properties such as fan-in, fan-out, and propagation delay. • Examples illustrating timing diagrams	Approximately 2 hours.
Module 3				
Unit 1: Combinational Circuits and Design Procedure	• Understand the concept of combinational circuits and their role in digital logic design. • Ability to follow the design procedure for combinational circuits.	• Lecture on the basics of combinational circuits and their design methodology. • Practice exercises on designing combinational circuits using truth tables and logic gates.	• Step-by-step examples of combinational circuit design procedures. • Group activities for designing and analyzing combinational circuits.	Approximately 2.5 hours.
Unit 2: Binary Subtractor	• Understand the operation and design of binary subtractors in digital circuits.	• Lecture on the principles of binary subtraction and binary subtractor circuits.	• Visual demonstrations of binary subtraction and subtractor circuit operation. • Guided practice	Approximately 2 hours.

	• Ability to design binary subtractors using logic gates.	• Practice exercises on designing binary subtractors using logic gates.	sessions for designing and analyzing binary subtractors.	
Unit 3: Multiplexers	• Understand the concept and operation of multiplexers (MUX). • Ability to design and analyse multiplexer circuits.	• Lecture on the principles of multiplexers and their applications. • Practice exercises on designing and implementing multiplexer circuits.	• Step-by-step explanations of multiplexer operation and circuit design. • Interactive sessions for designing and testing multiplexer circuits.	Approximately 2 hours.
Unit 4: De-multiplexers	• Understand the concept and operation of de-multiplexers (DEMUX). • Ability to design and analyse de-multiplexer circuits.	• Lecture on the principles of de-multiplexers and their applications. • Practice exercises on designing and implementing de-multiplexer circuits.	• Detailed explanations of de-multiplexer operation and circuit design. • Hands-on activities for designing and testing de-multiplexer circuits.	Approximately 2 hours.
Unit 5: Decoders	• Understand the concept and operation of decoders.	• Lecture on the principles of decoders and their	• Step-by-step explanations of decoder	Approximately 2 hours.

	• Ability to design and analyse decoder circuits for various applications.	role in digital systems. • Practice exercises on designing decoder circuits for different scenarios.	operation and circuit design. • Interactive sessions for designing and testing decoder circuits.	
Unit 6: Encoders	• Understand the concept and operation of encoders. • Ability to design and analyse encoder circuits for various applications.	• Lecture on the principles of encoders and their role in digital systems. • Practice exercises on designing encoder circuits for different scenarios.	• Lecture on the principles of encoders and their role in digital systems. • Practice exercises on designing encoder circuits for different scenarios.	Approximately 2 hours.
Unit 7: Latches	• Understand the principles and operation of latches in sequential circuits. • Ability to analyse and design latch circuits for various applications.	• Lecture on the fundamentals of latches and their significance in sequential circuit design. • Practice exercises on analyzing and designing latch circuits.	• Visual demonstrations of latch operation and circuit design. • Guided practice sessions for analyzing and designing latch circuits.	Approximately 2 hours.

Unit 8: Flip-Flops	• Understand the principles and operation of flip-flops in sequential circuits. • Ability to analyze and design flip-flop circuits for various applications.	• Lecture on the fundamentals of flip-flops and their role in sequential circuit design. • Practice exercises on analyzing and designing flip-flop circuits.	• Lecture on the fundamentals of flip-flops and their role in sequential circuit design. • Practice exercises on analyzing and designing flip-flop circuits.	Approximately 2 hours.
Module 4	.	.	.	
Unit 1: Sequential Circuits	• Understand the basic architectural differences between combinational and sequential circuits. • Familiarity with the operation and characteristics of sequential circuits.	• Lecture on the fundamentals of sequential circuits and their architectural distinctions. • Discussion on the operation and characteristics of sequential circuits.	• Visual aids illustrating the differences between combinational and sequential circuits. • Examples demonstrating the behavior and applications of sequential circuits.	Approximately 2.5 hours.
Unit 2: Conversion of Flip-Flops	• Understand the characteristics and operation of different types of flip-flops.	• Lecture on the characteristics and behavior of various flip-flops including	• Step-by-step explanations of flip-flop characteristics and conversion methods.	Approximately 2 hours.

	• Ability to convert flip-flops from one type to another.	SR, JK, D, and T flip-flops. • Practice exercises on converting flip-flops from one type to another.	• Guided practice sessions for converting flip-flops..	
Unit 3: Registers	• Understand the concept and operation of registers in digital systems. • Ability to design and analyze register circuits for various applications.	• Lecture on the principles of registers and their significance in digital systems. • Practice exercises on designing and implementing register circuits.	• Visual demonstrations of register operation and circuit design. • Hands-on activities for designing and testing register circuits.	Approximately 2 hours.
Unit 4: Counters	• Understand the principles and operation of counters in digital systems. • Ability to design and analyze counter circuits for various counting applications.	• Lecture on the fundamentals of counters and their applications. • Practice exercises on designing and implementing counter circuits.	• Detailed explanations of counter operation and circuit design. • Interactive sessions for analyzing and designing counter circuits.	Approximately 2 hours.
Module 5	.	.	.	

Unit 1: Memory Devices and Classification	• Understand the fundamentals of memory devices including ROM, RAM, and their classification. • Ability to classify different types of memory devices based on their characteristics and applications.	• Lecture on the principles of memory devices and their classification. • Discussion on the characteristics and applications of different types of memory.	• Visual aids illustrating the different types of memory devices and their classification. • Real-world examples of memory applications.	Approximately 2.5 hours.
Unit 2: Programmable Logic Array (PLAs)	• Understand the concept and operation of Programmable Logic Arrays (PLAs). • Ability to design and implement logic functions using PLAs.	• Lecture on the principles of PLAs and their architecture. • Practice exercises on designing and programming PLAs for specific logic functions.	• Step-by-step explanations of PLA architecture and operation. • Guided practice sessions for designing and programming PLAs.	Approximately 2 hours.
Unit 3: Programmable Logic Devices (PLDs)	• Understand the concept and operation of Programmable Logic Devices (PLDs).	• Lecture on the principles of PLDs and their architecture. • Practice exercises	• Visual demonstrations of PLD architecture and configuration methods. • Hands-on activities for	Approximately 2 hours.

	• Ability to program and configure PLDs for specific logic functions.	on programming and configuring PLDs using hardware description languages.	programming and testing PLDs.	
Unit 4: Field-Programmable Gate Arrays (FPGAs)	• Understand the concept and operation of Field-Programmable Gate Arrays (FPGAs). • Ability to program and configure FPGAs for complex digital systems.	• Lecture on the principles of FPGAs and their architecture. • Practice exercises on programming and configuring FPGAs for various applications.	• Detailed explanations of FPGA architecture and configuration methods. • Interactive sessions for programming and testing FPGAs.	Approximately 2.5 hours.

MAIN COURSE

CONTENT PAGE

Module 1 Introduction to Digital Logic Design

Unit 1 Introduction to Number Systems and Base Conversion

What is a Digital System?

Imagine digital systems as interconnected building blocks, handling information in a special way. These systems work with discrete chunks of data, represented in binary form - ones and zeros. You encounter digital systems in many everyday gadgets like calculators, computers, and even digital watches.

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Digital Systems Characteristics

Where do you find Digital System? You encounter digital systems in many everyday gadgets like calculators, computers, and even digital watches.

What makes digital systems ticks? Let's peek inside the digital world to see what sets it apart:

- Digital systems handle information bit by bit, much like assembling a puzzle piece by piece. Each piece, whether it's a number or a letter, fits snugly into the bigger picture, allowing the system to process and understand the information it receives.
- Digital systems communicate using signals, which act like messengers delivering important information. These signals travel through the system, ensuring that data reaches its destination accurately and efficiently. It's similar to how we use signals like Wi-Fi or Bluetooth to connect devices and share information.
- Picture digital systems as giant word puzzles, where numbers and letters are the building blocks. From counting numbers to alphabet letters, these systems can handle a wide range of data, making them versatile tools in our digital world.
- At the core of digital communication is the binary code, a language made up of just two symbols: 0 and 1. Much like a traffic light with

only two options - green for go and red for stop - digital signals use this binary language to convey information. It's a simple yet powerful system that forms the backbone of digital technology.

- Every signal in a digital system represents a single binary digit, known as a bit. Think of bits as the smallest units of information, like tiny switches that can be either on (1) or off (0). Just like how a single Lego brick contributes to a larger construction, each bit plays a crucial role in forming the digital landscape we interact with every day.

Comparison of Analog Systems and Digital Systems

- **Representation of Information:**

Analog Systems: Analog systems represent data using continuous signals that vary over time. These signals can take on any value within a range. Examples include analogue audio signals from a microphone, analogue temperature sensors, and analogue clocks.

Digital Systems: Digital systems represent data using discrete signals that have distinct, quantized values. These values are usually binary, represented by 0s and 1s. Examples include digital audio files, digital thermometers, and digital clocks.

- **Accuracy and Precision:**

Analog Systems: Analog systems can suffer from noise and distortion, which can reduce accuracy and precision, especially over long distances or time periods. However, analogue systems can often capture nuances that digital systems may miss.

Digital Systems: Digital systems are highly resistant to noise and distortion, providing high accuracy and precision. Digital signals can be transmitted over long distances without significant degradation.

- **Processing and Manipulation:**

Analog Systems: Analog signals are processed using analogue circuits, which often require specialized components such as operational amplifiers and filters. Manipulating analogue signals may involve techniques like amplification, filtering, and modulation.

Digital Systems: Digital signals are processed using digital circuits, typically implemented using digital logic gates and microprocessors. Manipulating digital signals involves operations like encoding, decoding, compression, and encryption.

Advantages of Digital System over Analog System

- **Ease of Storage and Transmission:** Digital signals can be easily stored, transmitted, and replicated without degradation using digital storage devices and communication protocols. Digital storage devices offer higher storage densities and faster access times compared to analogue storage mediums.
- **Robustness:** Digital systems are more robust against environmental factors such as temperature variations and electromagnetic interference. They can withstand harsh operating conditions better than analogue systems, making them suitable for use in challenging environments.
- **Integration and Compatibility:** Digital systems can integrate easily with other digital devices and systems, facilitating interoperability and compatibility. Digital communication protocols and standards ensure seamless connectivity between different digital devices and platforms.
- **Cost-Effectiveness:** While digital systems may have higher initial implementation costs due to the need for digital logic circuits and processors, they often offer more cost-effective solutions in the long run. Digital technology enables higher levels of automation, efficiency, and scalability, leading to reduced operational costs over time.

Number System

Understanding Number Systems: The Foundation of Counting and Computing;

Number systems serve as the backbone of counting and computation, providing a structured way to represent quantities. In the digital realm, modern computers communicate and operate using binary numbers, which consist of only two digits: 0 and 1. However, humans typically rely on the decimal number system in everyday life.

Decimal vs. Binary: Consider the decimal number 18. In binary, it is represented as 10010. Notice how binary numbers require more digits to represent the same value compared to decimal numbers. For larger numbers, dealing with lengthy binary strings becomes cumbersome. To address this issue, alternative number systems have emerged:

1. **Octal Number System (Base 8):** This system uses digits from 0 to 7, providing a more compact representation for large numbers compared to binary.
2. **Hexadecimal Number System (Base 16):** Hexadecimal numbers employ digits from 0 to 9, along with additional symbols from A to F. This system offers an even more concise representation, making it particularly useful in computer science and digital electronics.

3. **Binary Coded Decimal (BCD) System:** BCD is a binary-encoded representation of decimal numbers. Unlike pure binary, BCD uses groups of binary digits to represent each decimal digit, offering a compromise between binary and decimal systems.

Defining Number Systems: To understand any number system, we must specify its base, which determines the total number of available digits. For example, the binary system has a base of 2, while the decimal system has a base of 10. In any number system, the first digit is always zero, and the last digit is always one less than the base.

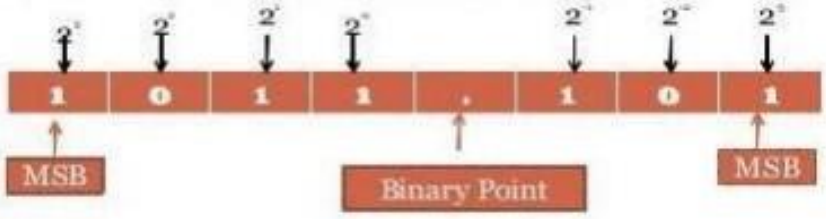
Binary number system:

Understanding the Binary Number System: Cracking the Code of 0s and 1s;

Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

The binary number system, with a radix of 2, is the language of computers, relying solely on two digits: 0 and 1. In this system, the weight of each digit is expressed as a power of 2.



Breaking Down Binary:

1. **Radix of 2:** In binary, we work with a base of 2, meaning there are only two available digits: 0 and 1. This simplicity allows for efficient digital communication and computation.

2. **Significant Bits:** Within a binary number, the leftmost bit holds the highest weight and is known as the Most Significant Bit (MSB). Conversely, the rightmost bit carries the lowest weight and is called the Least Significant Bit (LSB). These bits play crucial roles in determining the value of the binary number.

Putting Binary into Practice: For example, let's convert the binary number 1001.01_2 into its decimal equivalent:
$1001.01_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2})$
$1001.01_2 = (1 \times 8) + (0 \times 4) + (0 \times 2) + (1 \times 1) + (0 \times 0.5) + (1 \times 0.25)$
$1001.01_2 = 8 + 0 + 0 + 1 + 0 + 0.25$
$1001.01_2 = 9.25_{10}$

Decimal Number System: The decimal system, familiar to us all, use ten symbols: 0 through 9, creating a base of 10. Every time we count from 0 to 9 and start again, we're traversing one decimal place.
Octal Number System: Digital systems primarily work with binary numbers, which can get lengthy. To simplify, we use shorthand notations like octal and hexadecimal. Octal, with a base of 8, utilizes the first eight digits of the decimal system (0 through 7).
Hexadecimal Number System: Hexadecimal, with a base of 16, expands our numerical horizons even further. It embraces 16 symbols, including the decimal digits 0 through 9 and the letters A through F. These letters represent the values 10 through 15, offering a convenient way to express large numbers in a compact format.

Number Base Conversions
Navigating Number Base Conversions: Bridging the gap between Humans and Computers. Humans and computers each have their preferred numerical languages: decimal for us and binary for computers. However, translating between these systems is essential for communication between humans and machines.

i. **Conversion of Binary Numbers to Octal Numbers:**
- Divide the binary number into groups of three digits (2^3), starting from the right.
- Convert each group of three binary digits into its octal equivalent.
- Concatenate the octal equivalents of each group to get the final octal number.

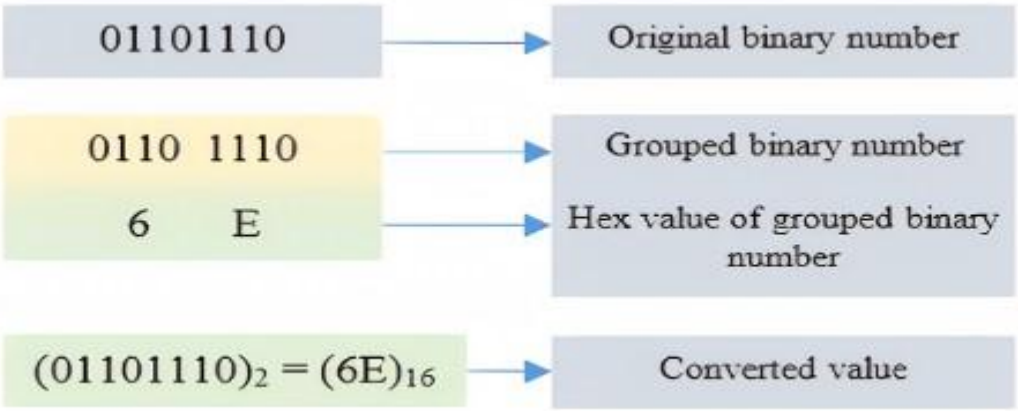
ii. **Conversion of Binary Numbers to Hexadecimal Numbers:**

$$\underbrace{010}_{2}\ \underbrace{010}_{2}\ \underbrace{111}_{7}\ \underbrace{.110}_{.6}\ \underbrace{010}_{2}$$

$$\text{So, } (10010111.11001)_2 = (227.62)_8$$

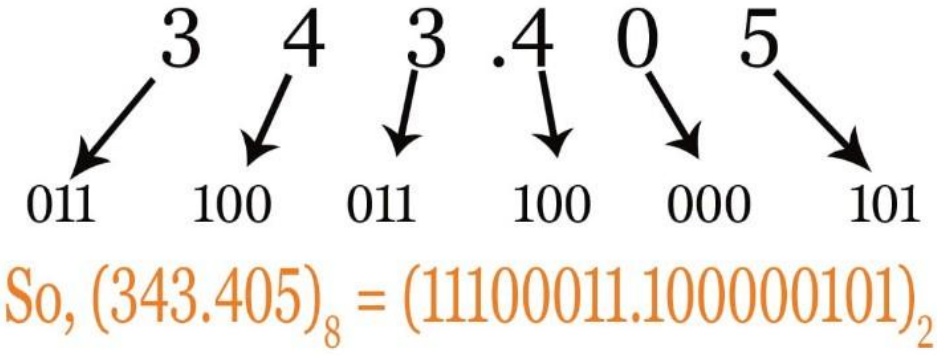
- Divide the binary number into groups of four digits (2^4), starting from the right.
- Convert each group of four binary digits into its hexadecimal equivalent.
- Concatenate the hexadecimal equivalents of each group to get the final hexadecimal number.

iii. **Octal to Binary Conversion:**



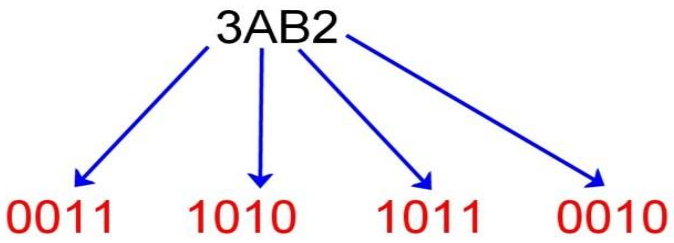
- Convert each octal digit into its three-bit binary equivalent.
- Concatenate the binary equivalents of each octal digit to get the final binary number.

iv. **Hexadecimal to Binary Conversion:**



$$\text{So, } (343.405)_8 = (11100011.100000101)_2$$

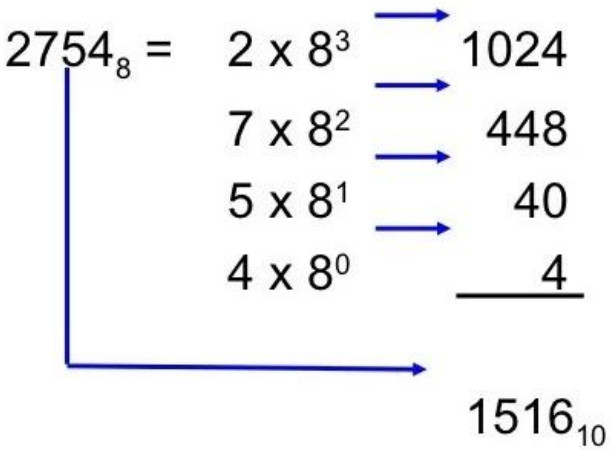
- Convert each hexadecimal digit into its four-bit binary equivalent.
- Concatenate the binary equivalents of each hexadecimal digit to get the final binary number.



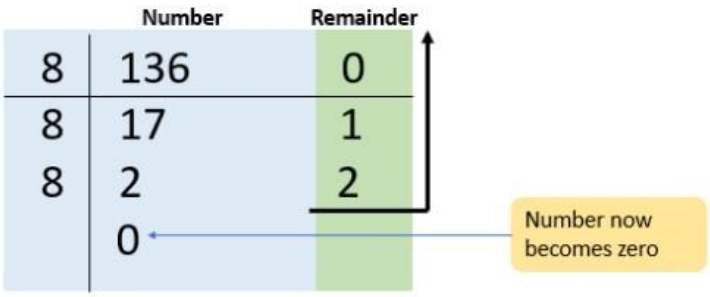
$3AB2_{16} = 11101010110010_2$

v. **Octal to Decimal Conversion:**

- Multiply each octal digit by 8^k, where *k* is the position of the digit (starting from the right).
- Sum up the products to obtain the decimal equivalent.



- **Decimal to Octal Conversion:**
- Divide the decimal number by 8 successively and record the remainders.
- The remainders, read from bottom to top, give the octal equivalent of the decimal number.



$(136)_{10} = (210)_8$

vi. **Hexadecimal to Decimal Conversion:**

- Multiply each hexadecimal digit by 16^k, where k is the position of the digit (starting from the right).
- Sum up the products to obtain the decimal equivalent.

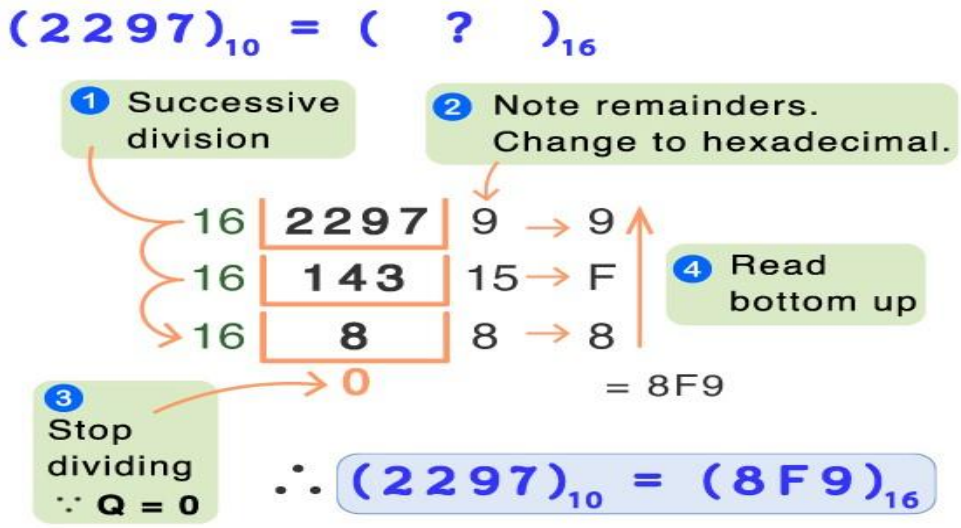
Digit	5	4	D	2
Place value	16^1	16^0	16^{-1}	16^{-2}

$$54.D2_{16}$$
$$= 5 \cdot 16^1 + 4 \cdot 16^0 + D \cdot 16^{-1} + 2 \cdot 16^{-2}$$
$$= 5 \cdot 16^1 + 4 \cdot 16^0 + 13 \cdot 16^{-1} + 2 \cdot 16^{-2}$$
$$= 80 + 4 + 0.8125 + 0.0078125$$
$$= 84.8203125$$

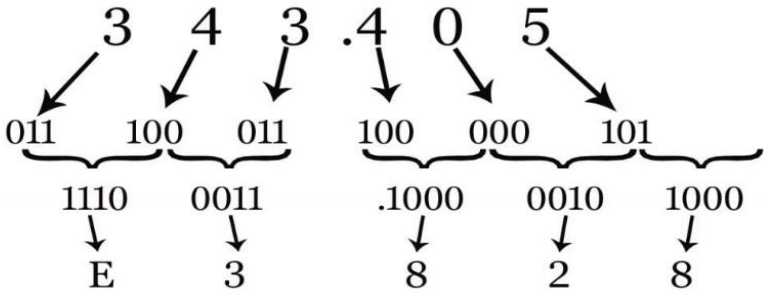
- **Decimal to Hexadecimal Conversion:**

- Divide the decimal number by 16 successively and record the remainders.
- Convert each remainder greater than 9 to its hexadecimal equivalent (A=10, B=11, ..., F=15).
- The remainders, read from bottom to top, give the hexadecimal equivalent of the decimal number.

vii. **Octal to Hexadecimal Conversion:**



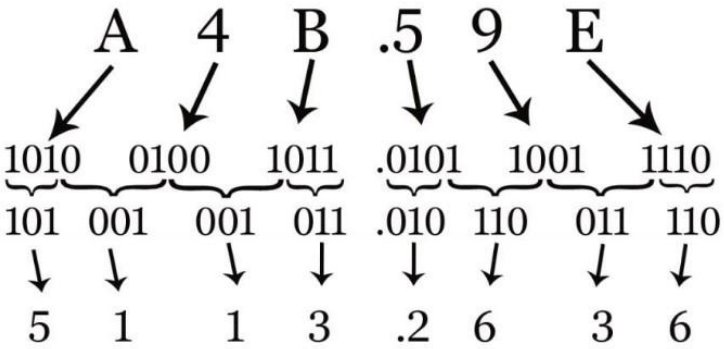
- Convert the octal number to binary.
- Convert the binary number to hexadecimal.



So, $(343.405)_8 = (E3.828)_{16}$

viii. **Hexadecimal to Octal Conversion:**

- Convert the hexadecimal number to binary.



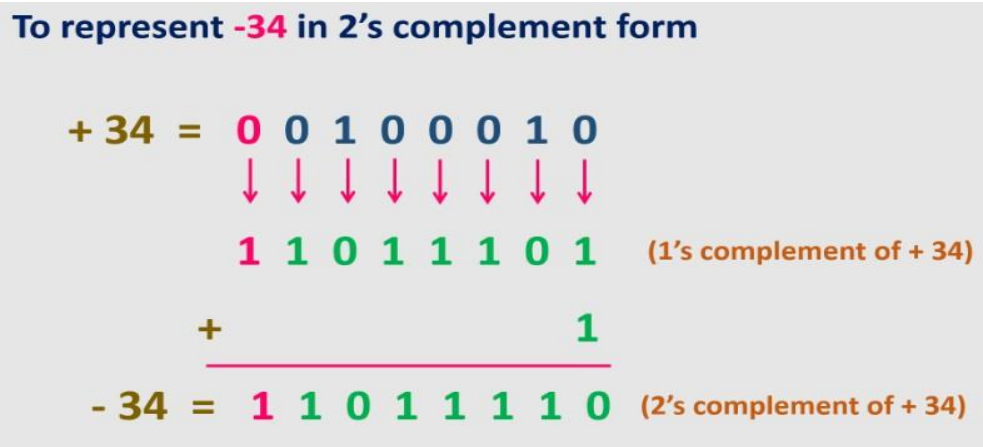
So, $(A4B.59E)_{16} = (5113.2636)_8$

- Convert the binary number to octal.

ix. **One's Complement and Two's Complement:**

- **One's Complement:** Invert all bits in the binary number (0s become 1s and vice versa) to get the one's complement.
- **Two's Complement:** Take the one's complement and add 1 to the result to obtain the two's complement.

SELF ASSESMENT EXERCISES



Multiple Choice Questions

1. What is the base of the binary number system?

A. 8
B. 10
C. 2 ✅
D. 16

Answer: C

2. Which number system uses digits 0–7?

A. Decimal
B. Binary
C. Hexadecimal
D. Octal ✅

Answer: D

3. What is the binary equivalent of decimal 18?

A. 10010 ✅
B. 11001
C. 10100
D. 11100

Answer: A

4. What is the most significant bit (MSB) in binary?

A. Rightmost bit
B. Leftmost bit ✅
C. Middle bit
D. Least significant bit

Answer: B

5. Which system uses digits and letters A–F?

A. Binary
B. Octal
C. Hexadecimal ✅
D. Decimal
Answer: C
6. What is the decimal equivalent of binary 1001.01?
A. 9.25 ✅
B. 10.5
C. 8.75
D. 9.5
Answer: A
7. What is the first step in converting binary to hexadecimal?
A. Divide by 8
B. Group digits in threes
C. Group digits in fours ✅
D. Multiply by 16
Answer: C
8. What does BCD stand for?
A. Binary Code Decimal ✅
B. Binary Conversion Data
C. Base Code Decimal
D. Binary Count Digit
Answer: A
9. What is the base of the decimal system?
A. 2
B. 8
C. 10 ✅
D. 16
Answer: C
10. Which conversion method involves dividing by 8 and reading remainders?
A. Decimal to Binary
B. Decimal to Octal ✅
C. Binary to Hex
D. Hex to Decimal
Answer: B

Fill in the Blank Questions

1. The binary system uses only _____ digits. → *two*
2. The hexadecimal system includes digits 0–9 and letters _____. → *A to F*
3. The leftmost bit in a binary number is called the _____. → *MSB*
4. BCD stands for _____. → *Binary Coded Decimal*
5. To convert decimal to octal, divide by _____. → *8*

Unit 2 Complement Systems and Codes

Binary codes are methods of representing data using binary digits (0s and 1s). They are widely used in digital systems, communication systems, and computing to encode information for storage, processing, and transmission. Various types of binary codes serve different purposes, each with its own rules and characteristics.

1. Gray Code (Reflected Binary Code):

Gray Code, also known as Reflected Binary Code, is a binary numeral system where two consecutive values differ by only one bit. It is designed to minimize errors in digital communication and analog-to-digital conversion systems.

Properties:

- Adjacent Gray code values differ by only one bit, reducing the likelihood of errors in transmission or signal conversion.
- The Gray code sequence is not unique; there are multiple possible sequences of Gray codes for any given number of bits.

Applications:

- Used in rotary encoders, where mechanical imperfections can cause signal jitter. Gray code ensures that only one bit changes at a time, minimizing misinterpretation.
- Employed in digital communication systems to reduce the effects of signal noise and errors during transmission.

Four-bit Gray code

Gray code	Decimal equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

- **Example:** The 3-bit Gray code sequence is 000,001,011,010,110,111,101,100. Note that adjacent codes differ by only one bit.

2. **Binary-Coded Decimal (BCD):**
Binary-Coded Decimal (BCD) is a binary encoding of decimal numbers, where each decimal digit is represented by its equivalent 4-bit binary code. It is commonly used in digital systems to facilitate arithmetic operations involving decimal numbers.

Properties:

- Each decimal digit is represented by a unique 4-bit binary code, allowing for direct conversion between decimal and binary representations.
- BCD codes 1010 through 1111 are invalid to avoid ambiguity, as they do not represent valid decimal digits.

Applications:

- Used in digital displays, such as LED or LCD displays in calculators, clocks, and electronic instruments, to accurately represent decimal numbers.
- Employed in financial systems and calculators for precise arithmetic calculations involving monetary values.
- **Example:** The BCD representation of the decimal number 25 is 0010 0101, where each group of four bits represents a decimal digit.

Excess-3 Code (XS-3):

Binary codes for the decimal digits

Decimal digit	(BCD) 8421	Excess-3	84-2-1	2421	(Biquinary) 5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

Excess-3 Code, also known as XS-3 or XS-3-2421 code, is a self-complementary binary code used to represent decimal digits. Each decimal digit is represented by its 4-bit binary equivalent, obtained by adding 3 to the digit and converting the result to binary.

Properties:

- XS-3 is self-complementary, meaning that the code for n is the complement of the code for $9 - n$.
- It simplifies arithmetic operations in digital systems by providing a direct representation of decimal digits in binary form.

Applications:

- Used in digital arithmetic circuits, such as adders and subtractors, to perform arithmetic operations on decimal numbers in binary form.
- Employed in calculators and digital counters for accurate arithmetic calculations and counting operations.

Example: The excess-3 code for the decimal number 7 is obtained by adding 3 to 7, resulting in 10, which in binary is 1010.

3. **ASCII (American Standard Code for Information Interchange):**

ASCII is a character encoding standard that assigns unique binary codes to characters, control characters, and symbols commonly used in computers and communication equipment.

Properties:

- Originally defined with 7 bits, ASCII codes have been extended to use 8 bits, allowing for the representation of 128 characters.
- ASCII encodes uppercase and lowercase letters, digits, punctuation marks, control characters (e.g., newline, carriage return), and special symbols.

Applications:

- Used in computing, telecommunications, and data transmission to represent text-based information in digital form.
- Employed in text-based communication protocols, file formats, and programming languages for character encoding and manipulation.

Example: The ASCII code for the uppercase letter 'A' is 01000001, while the ASCII code for the lowercase letter 'a' is 01100001.

American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

Error Detecting Codes

Error detecting codes are techniques used to detect errors that occur during the transmission or storage of data. These codes introduce redundancy into the data, allowing receivers to detect whether errors have occurred.

Properties:

- Error detecting codes add extra bits, called parity bits or checksums, to the data based on specific algorithms.
- If errors occur during transmission or storage, the received data will not match the expected data, indicating the presence of errors.
- Error detecting codes can identify the presence of errors but cannot correct them.

Types:

- **Parity Check:** Involves adding a single parity bit to the data, making the total number of bits (including the parity bit) either even or odd. Commonly used for detecting single-bit errors.
- **Checksums:** Calculated by summing all the bits or bytes of the data and appending the result to the data. Checksums can detect errors caused by multiple-bit errors or burst errors.

Example: Consider the data 1011010 with even parity. The parity bit is calculated as the XOR (exclusive OR) of all the data bits: $1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$. So, the transmitted data becomes 10110100. If a single bit error occurs during transmission, such as changing the last bit to 1, the parity check at the receiver will detect the error because the number of ones in the received data (odd) does not match the expected even parity.

Error Correcting Codes

Parity bit

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

Error correcting codes are techniques used to detect and correct errors that occur during the transmission or storage of data. These codes introduce redundancy into the data and provide mechanisms for detecting and correcting errors.

Properties:

- Error correcting codes add additional redundant bits to the data, allowing receivers to both detect and correct errors.
- These codes are more complex than error detecting codes and require additional computational overhead.
- Error correcting codes can correct a certain number of errors based on their design and the amount of redundancy introduced.

Types:

- **Hamming Codes:** A class of error-correcting codes capable of correcting single-bit errors and detecting double-bit errors. Hamming codes add parity bits at specific positions in the data to form code words that can detect and correct errors.
- **BCH Codes:** Bose-Chaudhuri-Hocquenghem (BCH) codes are a class of cyclic error-correcting codes capable of correcting multiple errors in data. They are widely used in applications where high reliability is required, such as digital communication and storage systems.

Example: Consider a Hamming code with parity bits. Suppose we have the data 1101001. Adding parity bits at positions 1, 2, and 4, we get the code word 01101001. If a single bit error occurs during transmission, the receiver can use the parity bits to identify and correct the error, ensuring that the received data matches the original transmitted data.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions

1. What is the one's complement of binary 1010?

A. 0101
B. 1111
C. 0101 ✅
D. 1001

2. What is the two's complement of binary 1010?

A. 0101
B. 0110 ✅
C. 1001
D. 1100

Answer: B

3. Which code changes only one bit between consecutive values?

A. ASCII

B. Gray Code ✅
C. BCD
D. Excess-3
Answer: B
4. What is the BCD representation of decimal 25?
A. 11001
B. 0010 0101 ✅
C. 1010 0101
D. 0100 0011
Answer: B
5. Excess-3 code adds _____ to each decimal digit.
A. 2
B. 3 ✅
C. 4
D. 5
Answer: B
6. ASCII uses how many bits originally?
A. 6
B. 7 ✅
C. 8
D. 10
Answer: B
7. Which code is self-complementary?
A. BCD
B. ASCII
C. Excess-3 ✅
D. Gray Code
Answer: C
8. What is the ASCII code for uppercase 'A'?
A. 01000001 ✅
B. 01100001
C. 00100001
D. 11000001
Answer: A
9. What is the main purpose of error detecting codes?
A. To compress data
B. To detect transmission errors ✅
C. To encrypt data
D. To store data
Answer: B
10. Which code can correct single-bit errors?
A. Parity
B. Hamming ✅
C. ASCII
D. Gray

Answer: B

Fill in the Blank Questions

1. One's complement is obtained by _____ all bits. → *inverting*
2. Two's complement is one's complement plus _____. → *1*
3. Gray code differs by only _____ bit between values. → *one*
4. ASCII originally used _____ bits. → *7*
5. Hamming code can correct _____-bit errors. → *single*

Unit 3 Digital Logic Gates

Digital electronic circuits operate with voltages of **two logic levels** namely Logic Low and Logic High. The range of voltages corresponding to Logic Low is represented with '0'. Similarly, the range of voltages corresponding to Logic High is represented with '1'.

The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate**. Hence, the Logic gates are the building blocks of any digital system. We can classify these Logic gates into the following three categories.

- Basic gates
- Universal gates
- Special gates

Now, let us discuss about the Logic gates come under each category one by one.

Basic Gates

In earlier chapters, we learnt that the Boolean functions can be represented either in sum of products form or in product of sums form based on the requirement. So, we can implement these Boolean functions by using basic gates. The basic gates are AND, OR & NOT gates.

AND gate

An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '.'.

The following table shows the **truth table** of 2-input AND gate.

A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

Here A, B are the inputs and Y is the output of two input AND gate. If both inputs are '1', then only the output, Y is '1'. For remaining combinations of inputs, the output, Y is '0'.

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

A.B

The following figure shows the **symbol** of an AND gate, which is having two inputs A, B and one output, Y.

This AND gate produces an output Y, which is the **logical AND** of two inputs A, B. Similarly, if there are 'n' inputs, then the AND gate produces an output, which is the logical AND of all those inputs. That means, the output of AND gate will be '1', when all the inputs are '1'.

OR gate

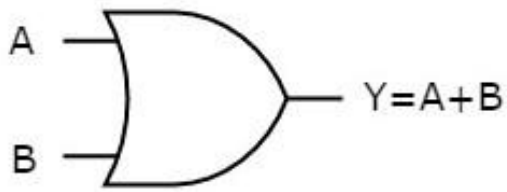
An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol '+'.

The following table shows the **truth table** of 2-input OR gate.

Here A, B are the inputs and Y is the output of two input OR gate. If both inputs are '0', then only the output, Y is '0'. For remaining combinations of inputs, the output, Y is '1'.

The following figure shows the **symbol** of an OR gate, which is having two inputs A, B and one output, Y.

This OR gate produces an output Y, which is the **logical OR** of two inputs

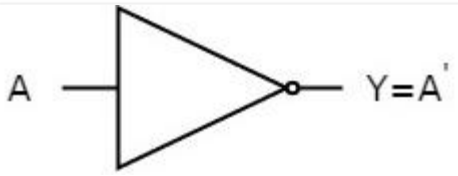


A, B. Similarly, if there are 'n' inputs, then the OR gate produces an output, which is the logical OR of all those inputs. That means, the output of an OR gate will be '1', when at least one of those inputs is '1'.

NOT gate

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

A	Y = A′
0	1
1	



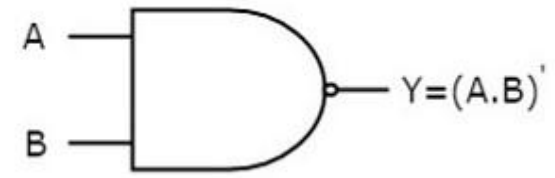
Here A and Y are the input and output of NOT gate respectively. If the input, A is ‘0’, then the output, Y is ‘1’. Similarly, if the input, A is ‘1’, then the output, Y is ‘0’.

The following figure shows the **symbol** of NOT gate, which is having one input, A and one output, Y. This NOT gate produces an output Y, which is the **complement** of input, A.

Universal gates
NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function, which is in sum of products form by using NAND gates alone. Similarly, we can implement any Boolean function, which is in product of sums form by using NOR gates alone.

NAND gate

NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs. The following table shows the **truth table** of 2-input NAND gate Here A, B are the inputs and Y is the output of two input NAND gate. When both inputs are '1', the output, Y is '0'. If at least one of the input is zero, then the output, Y is '1'. This is just opposite to that of two input AND gate operation. NAND gate operation is same as that of AND gate followed by an inverter. That's why the NAND gate symbol is represented like that. The following image shows the **symbol** of NAND gate, which is having two inputs A, B and one output, Y.



A		
0		
0	1	1
1	0	1
1	1	0

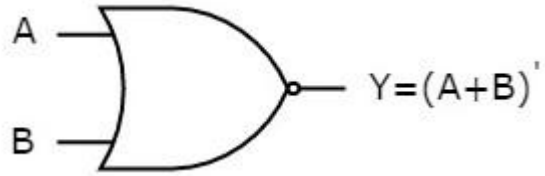
NOR gate

NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.
The following table shows the **truth table** of 2-input NOR gate

A	B	Y = $A + B'$
0	0	1
0	1	0
1	0	0
1	1	0

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

The following figure shows the **symbol** of NOR gate, which is having two inputs A, B and one output, Y.



NOR gate operation is same as that of OR gate followed by an inverter. That's why the NOR gate symbol is represented like that.

'

Special Gates

Ex-OR & Ex-NOR gates are called as special gates. Because, these two gates are special cases of OR & NOR gates.

Ex-OR gate

The full form of Ex-OR gate is **Exclusive-OR** gate. Its function is same as that of OR gate except for some cases, when the inputs having even number of ones.

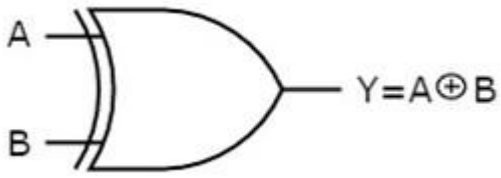
The following table shows the **truth table** of 2-input Ex-OR gate.

A	B	Y = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Here A, B are the inputs and Y is the output of two input Ex-OR gate. The truth table of Ex-OR gate is same as that of OR gate for first three rows. The only modification is in the fourth row. That means, the output Y is zero instead of one, when both the inputs are one, since the inputs having even number of ones.

Therefore, the output of Ex-OR gate is '1', when only one of the two inputs is '1'. And it is zero, when both inputs are same.

Below figure shows the **symbol** of Ex-OR gate, which is having two inputs A, B and one output, Y.



Ex-OR gate operation is similar to that of OR gate, except for few combinations of inputs. That's why the Ex-OR gate symbol is represented like that. The output of Ex-OR gate is '1', when odd number of ones present at the inputs. Hence, the output of Ex-OR gate is also called as an **odd function**.

Ex-NOR gate

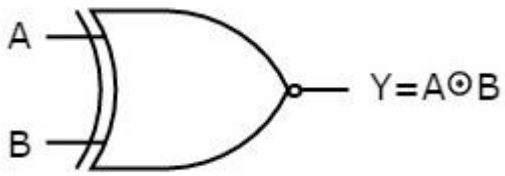
The full form of Ex-NOR gate is **Exclusive-NOR** gate. Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones. The following table shows the **truth table** of 2-input Ex-NOR gate.

Here A, B are the inputs and Y is the output. The truth table of Ex-NOR gate is same as that of NOR gate for first three rows. The only modification is in the fourth row. That means, the output is one instead of zero, when both the inputs are one.

A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Therefore, the output of Ex-NOR gate is '1', when both inputs are same. And it is zero, when both the inputs are different.

The following figure shows the **symbol** of Ex-NOR gate, which is having two inputs A, B and one output, Y.



Ex-NOR gate operation is similar to that of NOR gate, except for few combinations of inputs. That's why the Ex-NOR gate symbol is represented like that. The output of Ex-NOR gate is '1', when even number of ones present at the inputs. Hence, the output of Ex-NOR gate is also called as an **even function**.

From the above truth tables of Ex-OR & Ex-NOR logic gates, we can easily notice that the Ex-NOR operation is just the logical inversion of Ex-OR operation.

Basic Theorem and Properties

Basic Laws of Boolean Algebra: Following are the three basic laws of Boolean Algebra:

- Commutative law
- Associative law
- Distributive law

Commutative Law

If any logical operation of two Boolean variables give the same result irrespective of the order of those two variables, then that logical operation is said to be **Commutative**. The logical OR & logical AND operations of two Boolean variables x & y are shown below:

$x + y = y + x$

$x.y = y.x$

The symbol '+' indicates logical OR operation. Similarly, the symbol '.' indicates logical AND operation and it is optional to represent. Commutative law obeys for logical OR & logical AND operations.

Associative Law

If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be **Associative**. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$(x + y) + z = x + (y + z)$

$(x.y).z = x.(y.z)$

Associative law obeys for logical OR & logical AND operations.

Distributive Law

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be **Distributive**. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$x.(y + z)= (x.y) + (x.z)$

$x + (y.z) = (x + y).(x + z)$

Distributive law obeys for logical OR and logical AND operations.

These are the Basic laws of Boolean algebra. We can verify these laws easily, by substituting the Boolean variables with '0' or '1'.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions

1. Which logic gate outputs true only when all inputs are true?

A. OR
B. AND
C. NOT
D. XOR

Answer: B

2. What is the output of an OR gate when both inputs are 0?

A. 0
B. 1
C. Undefined
D. Same as input

Answer: A

3. Which gate inverts the input signal?

A. AND
B. OR
C. NOT
D. NAND

Answer: C

4. What is the output of a NAND gate when both inputs are 1?

A. 1
B. 0
C. Undefined
D. Same as input

Answer: B

5. Which gate gives a true output only when inputs are different?

A. AND
B. OR
C. XOR
D. NOR

Answer: C

6. What is the symbol used to represent a NOT gate?

A. ∧
B. ∨
C. ¬
D. ⊕

Answer: C

7. Which gate is the inverse of the OR gate?

A. AND
B. NOR
C. XOR
D. NAND

Answer: B

8. What is the Boolean expression for an AND gate?

A. A + B

B. $A \cdot B$
C. $A \oplus B$
D. $\neg A$
Answer: B
9. Which gate is considered a universal gate?
A. XOR
B. NAND
C. OR
D. NOT
Answer: B
10. What is the output of a NOR gate when both inputs are 0?
A. 0
B. Undefined
C. 1
D. Same as input
Answer: C

Fill in the Blank Questions
1. The _____ gate outputs true only when all inputs are true. → *AND*
2. The _____ gate inverts the input signal. → *NOT*
3. A _____ gate gives true output only when inputs differ. → *XOR*
4. The Boolean expression for an AND gate is _____. → $A \cdot B$
5. The _____ gate is known as a universal gate. → *NAND*

Unit 4: Boolean Algebra

A Boolean algebra is a closed algebraic system containing a set K of two or more elements and the two operators · and + which refer to logical AND and logical OR

- x + 0 = x
- x · 0 = 0
- x + 1 = 1
- x · 1 = 1
- x + x = x
- x · x = x
- x + x’ = x
- x · x’ = 0
- x + y = y + x
- xy = yx
- x + (y + z) = (x + y) + z
- x (yz) = (xy) z
- x (y + z) = xy + xz
- x + yz = (x + y)(x + z)
- (x + y)’ = x’ y’
- (xy)’ = x’ + y’
- (x’)’ = x

Theorems of Boolean Algebra

The following two theorems are used in Boolean algebra.

- Duality theorem
- DeMorgan’s theorem

Duality Theorem This theorem states that the **dual** of the Boolean function is obtained by interchanging the logical AND operator with logical OR operator and zeros with ones. For every Boolean function, there will be a corresponding Dual function. Let us make the Boolean equations relations that we discussed in the section of Boolean postulates and basic laws into two groups. The following table shows these two groups

DeMorgan’s Theorem

This theorem is useful in finding the **complement of Boolean function**. It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable.
DeMorgan’s theorem with 2 Boolean variables x and y can be represented as

- (x+y)’ = x’.y’

The dual of the above Boolean function is

- (x.y)’ = x’ + y’

Therefore, the complement of logical AND of two Boolean variables is equal to the logical OR of each complemented variable. Similarly, we can apply DeMorgan's theorem for more than 2 Boolean variables also.

Group1	Group2
x + 0 = x	x.1 = x
x + 1 = 1	x.0 = 0
x + x = x	x.x = x
x + x′ = 1	x.x′ = 0
x + y = y + x	x.y = y.x
x + $y + z$ = $x + y$ + z	x.$y.z$ = $x.y$.z
x.$y + z$ = x.y + x.z	x + $y.z$ = $x + y.x + z$

Example
Let us find the **complement** of the Boolean function, f = p'q + pq'.
The complement of Boolean function is f' = p′q + pq′.
Step 1 − Use DeMorgan's theorem, x+y' = x'.y'.
⇒ f' = p′q'.pq″
Step 2 − Use DeMorgan's theorem, x.y' = x' + y'
⇒ f' = {p'' + q'}.{p' + q''}
Step3 − Use the Boolean postulate, x''=x.
⇒ f' = {p + q'}.{p' + q}
⇒ f' = pp' + pq + p'q' + qq'
Step 4 − Use the Boolean postulate, xx'=0.
⇒ f = 0 + pq + p'q' + 0
⇒ f = pq + p'q'
Therefore, the **complement** of Boolean function, p'q + pq' is **pq + p'q'**.

Unit 5: Canonical & Standard Forms

We will get four Boolean product terms by combining two variables x and y with logical AND operation. These Boolean product terms are called as **min terms** or **standard product terms**. The min terms are x'y', x'y, xy' and xy.
Similarly, we will get four Boolean sum terms by combining two variables x and y with logical OR operation. These Boolean sum terms are called as **Max terms** or **standard sum terms**. The Max terms are x + y, x + y', x' + y and x' + y'. The following table shows the representation of min terms and MAX terms for 2 variables.

x	y	Min terms	Max terms
0	0	$m_0=x'y'$	$M_0=x+y$
0	1	$m_1=x'y$	$M_1=x+y'$
1	0	$m_2=xy'$	$M_2=x'+y$
1	1	$m_3=xy$	$M_3=x'+y'$

If the binary variable is '0', then it is represented as complement of variable in min term and as the variable itself in Max term. Similarly, if the binary variable is '1', then it is represented as complement of variable in Max term and as the variable itself in min term.
From the above table, we can easily notice that min terms and Max terms are complement of each other. If there are 'n' Boolean variables, then there will be 2^n min terms and 2^n Max terms.

Canonical SoP and PoS forms

A truth table consists of a set of inputs and outputs. If there are 'n' input variables, then there will be 2^n possible combinations with zeros and ones. So the value of each output variable depends on the combination of input variables. So, each output variable will have '1' for some combination of input variables and '0' for some other combination of input variables.
Therefore, we can express each output variable in following two ways.

- Canonical SoP form
- Canonical PoS form

Canonical SoP form

Canonical SoP form means Canonical Sum of Products form. In this form, each product term contains all literals. So, these product terms are nothing but the min terms. Hence, canonical SoP form is also called as **sum of min terms** form.

First, identify the min terms for which, the output variable is one and then do the logical OR of those min terms in order to get the Boolean expression function corresponding to that output variable. This Boolean function will be in the form of sum of min terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Example

Consider the following **truth table**.

Inputs			Output
p	q	r	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Here, the output f is '1' for four combinations of inputs. The corresponding min terms are p'qr, pq'r, pqr', pqr. By doing logical OR of these four min terms, we will get the Boolean function of output f.

Therefore, the Boolean function of output is, f = p'qr + pq'r + pqr' + pqr.

This is the **canonical SoP form** of output, f. We can also represent this function in following two notations.

$F = m_3 + m_5 + m_6 + m_7$

$f = \sum m (3, 5, 6, 7)$

In one equation, we represented the function as sum of respective min terms. In other equation, we used the symbol for summation of those min terms.

Canonical PoS form

Canonical PoS form means Canonical Product of Sums form. In this form, each sum term contains all literals. So, these sum terms are nothing but the Max terms. Hence, canonical PoS form is also called as **product of Max terms** form.

First, identify the Max terms for which, the output variable is zero and then do the logical AND of those Max terms in order to get the Boolean expression function corresponding to that output variable. This Boolean function will be in the form of product of Max terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Example

Consider the same truth table of previous example. Here, the output f is '0' for four combinations of inputs. The corresponding Max terms are p + q + r, p + q + r', p + q' + r, p' + q + r. By doing logical AND of these four Max terms, we will get the Boolean function of output f.

Therefore, the Boolean function of output is, f = p + q + r.p + q + r′.p + q′ + r.p′ + q + r. This is the **canonical PoS form** of output, f. We can also represent this function in following two notations.

$F = M_0.M_1.M_2.M_4$

$f = \prod M (0, 1, 2, 4)$

In one equation, we represented the function as product of respective Max terms. In other equation, we used the symbol for multiplication of those Max terms.

The Boolean function, f = p + q +r.p + q + r′.p + q′ + r.p′ + q + r is the dual of the Boolean function, f = p'qr + pq'r + pqr' + pqr.

Therefore, both canonical SoP and canonical PoS forms are **Dual** to each other. Functionally, these two forms are same. Based on the requirement, we can use one of these two forms.

Standard SoP and PoS forms

We discussed two canonical forms of representing the Boolean outputs. Similarly, there are two standard forms of representing the Boolean outputs. These are the simplified version of canonical forms.

- Standard SoP form
- Standard PoS form

We will discuss about Logic gates in later chapters. The main **advantage** of standard forms is that the number of inputs applied to logic gates can be minimized. Sometimes, there will be reduction in the total number of logic gates required.

Standard SoP form

Standard SoP form means **Standard Sum of Products** form. In this form, each product term need not contain all literals. So, the product terms may

or may not be the min terms. Therefore, the Standard SoP form is the simplified form of canonical SoP form.

We will get Standard SoP form of output variable in two steps.

- Get the canonical SoP form of output variable
- Simplify the above Boolean function, which is in canonical SoP form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical SoP form. In that case, both canonical and standard SoP forms are same.

Example

Convert the following Boolean function into Standard SoP form.

Convert the following Boolean function into Standard SoP form.

F = p'qr + pq'r + pqr' + pqr

The given Boolean function is in canonical SoP form. Now, we have to simplify this Boolean function in order to get standard SoP form.

Step 1 – Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

⇒ f = p'qr + pq'r + pqr' + pqr + pqr + pqr

Step 2 – Use **Distributive law** for 1^{st} and 4^{th} terms, 2^{nd} and 5^{th} terms, 3^{rd} and 6^{th} terms.

⇒ f = qrp′ + p + prq′ + q + pqr′ + r

Step 3 – Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in each parenthesis.

⇒ f = qr1 + pr1 + pq1

Step 4 – Use **Boolean postulate**, x.1 = x for simplifying above three terms.

⇒ f = qr + pr + pq

⇒ f = pq + qr + pr

This is the simplified Boolean function. Therefore, the **standard SoP form** corresponding to given canonical SoP form is **f = pq + qr + pr**

Standard PoS form

Standard PoS form means **Standard Product of Sums** form. In this form, each sum term need not contain all literals. So, the sum terms may or may not be the Max terms. Therefore, the Standard PoS form is the simplified form of canonical PoS form.

We will get Standard PoS form of output variable in two steps.

- Get the canonical PoS form of output variable
- Simplify the above Boolean function, which is in canonical PoS form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical PoS form. In that case, both canonical and standard PoS forms are same.

Tutor Marked Assignment

1. What is the logical expression for Y = A + A´B?
2. What is the octal equivalent of $(F3B1)_{16}$?
3. Minimum number of 2 input NOR Gates required to realize f = C + AB is?
4. Realize W = AB + CD + EF + GH using 2 input NAND gates.
5. Convert (312)8 into decimal

SELF-ASSESMENT EXERCISES

Multiple Choice Questions

1. What is Boolean algebra primarily used for?

A. Designing mechanical systems
B. Simplifying logical expressions
C. Calculating interest rates
D. Programming in Python

Answer: B

2. Which symbol represents the AND operation in Boolean algebra?

A. ·
B. ⊕
C. ¬
D. #

Answer: A

3. What is the identity element for the OR operation?

A. 0
B. 1
C. A
D. ¬A

Answer: B

4. What is the result of A + 0 in Boolean algebra?

A. 0
B. A
C. 1
D. ¬A

Answer: B

5. Which law states that A + A = A?

A. Identity Law
B. Idempotent Law
C. Complement Law
D. Distributive Law

Answer: B

6. What is the complement of 1 in Boolean algebra?

A. 1
B. 0
C. A
D. ¬A

Answer: B

7. Which law is represented by $A \cdot (B + C) = A \cdot B + A \cdot C$?
A. Associative Law
B. De Morgan's Law
C. Distributive Law
D. Absorption Law
Answer: C
8. What does De Morgan's first law state?
A. $\neg(A + B) = \neg A \cdot \neg B$
B. $\neg(A \cdot B) = \neg A \cdot \neg B$
C. $A + A = A$
D. $A \cdot 1 = A$
Answer: A
9. Which of the following is a valid simplification using Boolean laws?
A. $A + A = 1$
B. $A \cdot 0 = 0$
C. $A \cdot A = 0$
D. $A + 1 = 0$
Answer: B
10. What is the result of $A \cdot 1$ in Boolean algebra?
A. 0
B. A
C. 1
D. $\neg A$
Answer: B

Fill in the Blank Questions
1. Boolean algebra is used to _____ logical expressions. → *simplify*
2. The symbol for the AND operation is _____. → ·
3. The law that states $A + A = A$ is called the _____ law. → *idempotent*
4. De Morgan's first law states that $\neg(A + B) =$ _____. → $\neg A \cdot \neg B$
5. The result of $A \cdot 0$ is _____. → *0*

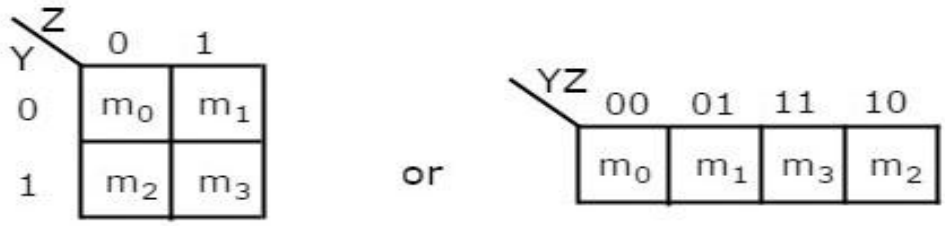
Module 2 Minimization Techniques

Unit 1 Karnaugh Map Method
Unit 2 Manipulation and Minimisation
Unit 3 Physical Properties of Gates

Unit 1 The Karnaugh Map Method

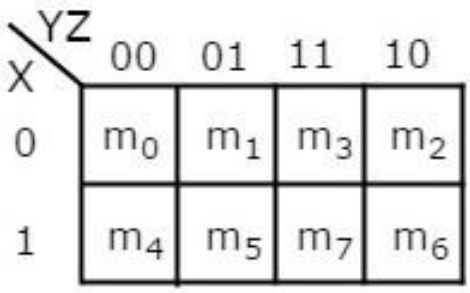
In previous lecture, we have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.
To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 5 Variables
K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.
2 Variable K-Map
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map



The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.

- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.

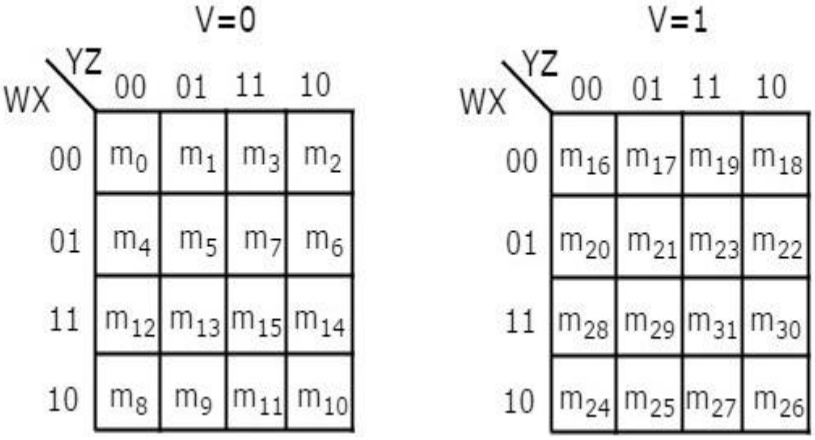
WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are {(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)}.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variables is 5. The following figure shows **5 variable K-Map**.

- There is only one possibility of grouping 32 adjacent min terms.



- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m_0 to m_{15} and m_{16} to m_{31}.
- If v=0, then 5 variable K-map becomes 4 variable K-map.

In the above all K-maps, we used exclusively the min terms notation. Similarly, you can use exclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is in **standard sum of products** form after simplifying the K-map.

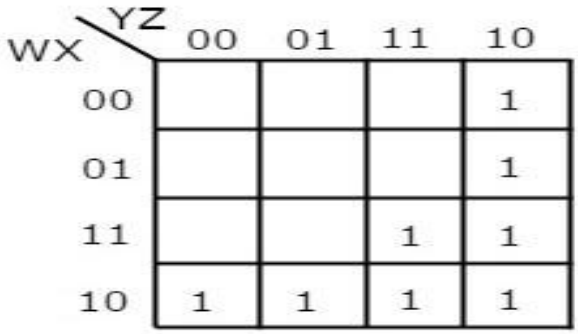
Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

Follow these **rules for simplifying K-maps** in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.
- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single '1' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note 1 − If outputs are not defined for some combination of inputs, then those output values will be represented with **don't care symbol 'x'**. That means, we can consider them as either '0' or '1'.
Note 2 − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1'.

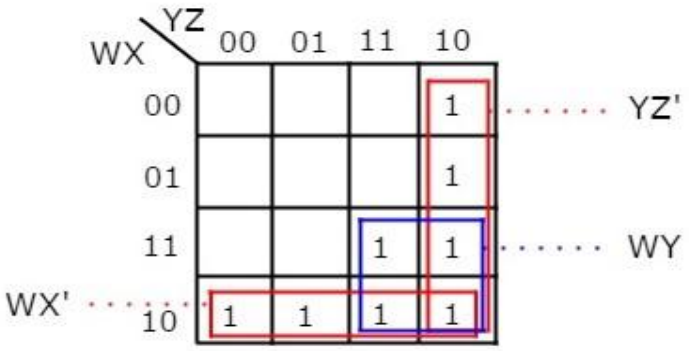
Here, 1s are placed in the following cells of K-map.



- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, **WX'Y'**.
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, **WY**.
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, **W'YZ'**.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The **4 variable K-map** with these three **groupings** is shown in the following figure.

Here, we got three prime implicants WX', WY & YZ'. All these prime



implicants are **essential** because of following reasons.

- Two ones **(m_8 & m_9)** of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.

- Single one **(m_{15})** of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.
- Two ones **(m_2 & m_6)** of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the **simplified Boolean function** is

f = WX' + WY + YZ'

Follow these **rules for simplifying K-maps** in order to get standard product of sums form.

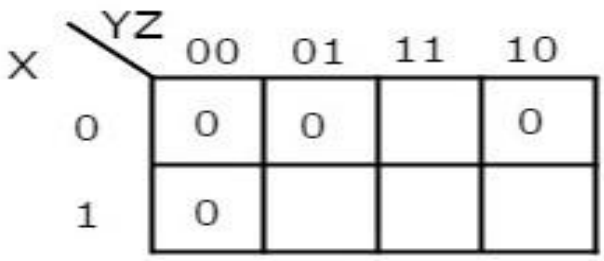
- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one sum term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single '0' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

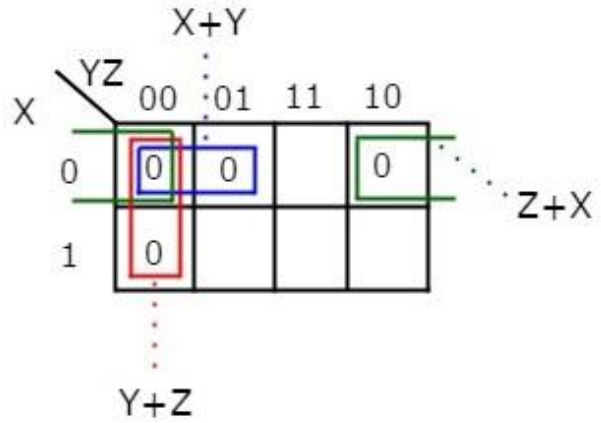
Example

Let us **simplify** the following Boolean function, f (X,Y,Z) = ∏ M (0, 1, 2, 4) using K-map.

The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max



terms are M_0, M_1, M_2 & M_4. The 3 **variable K-map** with zeroes corresponding to the given Max terms is shown in the following figure.
There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes. After these three groupings, there is no single zero left as ungrouped. The **3 variable K-map** with these three **groupings** is shown in the following figure.



Here, we got three prime implicants X + Y, Y + Z & Z + X. All these prime implicants are **essential** because one zero in each grouping is not covered by any other groupings except with their individual groupings.
Therefore, the **simplified Boolean function** is
f = X + Y.Y + Z.Z + X
In this way, we can easily simplify the Boolean functions up to 5 variables using K-map method. For more than 5 variables, it is difficult to simplify the functions using K-Maps. Because, the number of **cells** in K-map gets **doubled** by including a new variable.
Due to this checking and grouping of adjacent ones *Minterms* or adjacent zeros *Maxterms* will be complicated.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the primary purpose of the Karnaugh Map (K-map) method?**

A. To design logic gates
B. To simplify Boolean functions
C. To convert decimal to binary
D. To build truth tables

Answer: B

Explanation: K-map is a graphical method used to simplify Boolean expressions efficiently.

2. **How many cells are in a 3-variable K-map?**

A. 4

B. 8
C. 16
D. 32
Answer: B
Explanation: A 3-variable K-map contains $2^3 = 8$ cells.

3. **Which grouping size is NOT valid in a K-map simplification?**
A. 1
B. 2
C. 3
D. 4
Answer: C
Explanation: Groupings must be in powers of two: 1, 2, 4, 8, etc.

4. **What is a prime implicant in K-map terminology?**
A. A variable in the function
B. A group of adjacent zeros
C. A product term from a grouping
D. A minimized truth table
Answer: C
Explanation: Each valid grouping in a K-map yields a product term called a prime implicant.

5. **What makes a prime implicant essential?**
A. It contains only one variable
B. It covers at least one '1' not covered by any other group
C. It is the largest group possible
D. It appears in every row
Answer: B
Explanation: Essential prime implicants uniquely cover at least one '1' in the K-map.

6. **What does a don't care condition ('x') represent in a K-map?**
A. A required '0'
B. A required '1'
C. An undefined output that can be treated as either '0' or '1'
D. A redundant variable
Answer: C
Explanation: Don't care conditions can be used to optimize groupings by treating them as '0' or '1'.

7. **What is the result of simplifying a K-map using min terms?**
A. Product of sums
B. Sum of products
C. Truth table
D. Binary code
Answer: B
Explanation: Grouping '1's in a K-map leads to a simplified sum of products expression.

8. **How many cells are in a 5-variable K-map?**
A. 16

B. 32
C. 64
D. 8
Answer: B
Explanation: A 5-variable K-map contains $2^5 = 32$ cells.

9. **Which of the following is a valid grouping in a 4-variable K-map?**
A. (m0, m1, m2)
B. (R1, R2)
C. (C1, C2, C3)
D. (m0, m4, m8, m12, m1)
Answer: B
Explanation: Rows R1 and R2 represent valid groupings of 8 adjacent min terms.

10. **What is the simplified Boolean function from the example given in the unit?**
A. WX + WY + YZ
B. WX' + WY + YZ'
C. WX'Y' + WY + W'YZ'
D. WX + WY + YZ'
Answer: B
Explanation: The final simplified function from the example is f = WX' + WY + YZ'.

Fill in the Blank Questions

1. A K-map with 4 variables contains _____ cells.
Answer: 16

2. Groupings in a K-map must be in powers of _____.
Answer: two

3. A _____ prime implicant covers at least one '1' not covered by any other grouping.
Answer: essential

4. Don't care conditions are represented by the symbol _____ in a K-map.
Answer: x

5. The simplified Boolean function from the product of Max terms example is _____.
Answer: X + Y · Y + Z · Z + X

Unit 2 Manipulation and Minimization

What is Minimization?

- In mathematics, expressions are simplified for a number of reasons, for instance simpler expression are easier to understand and easier to write down, they are also less prone to error in interpretation but, most importantly, simplified expressions are usually more efficient and effective when implemented in practice.
- A Boolean expression is composed of variables and terms. The simplification of Boolean expressions can lead to more effective computer programs, algorithms and circuits.

Algebraic Manipulation of Boolean Expressions

- We can now start doing some simplifications

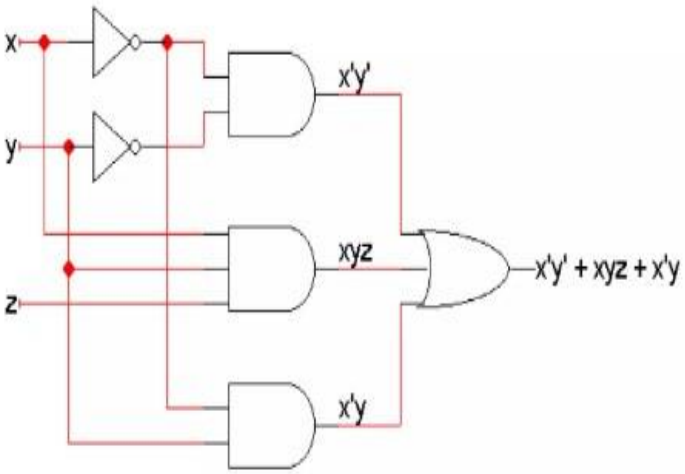
Tabular Method of Minimization

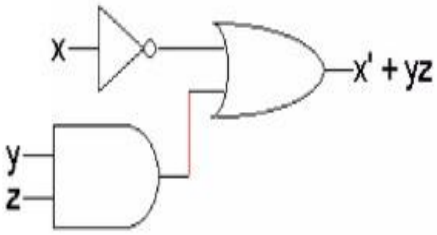
x'

= x'(y' + y) + xyz [Distributive: x'y' + x'y = x'(y' + y)]
= x'•1 + xyz [complement: x' + x = 1]
= x' + xyz [identity: x'•1 = x']
= (
=
=

- The tabular method which is also known as the Quine-McCluskey method is particularly useful when minimising functions having a large number of variables, e.g. The six-variable functions. Computer programs have been developed employing this algorithm.
- Minimisation can be achieved by a number of methods, three well known methods are:
  1. Algebraic Manipulation of Boolean Expressions
  2. Tabular Method of Minimization
  3. Karnaugh Maps

- Here are two different but *equivalent* circuits.
- In general the one with fewer gates is "better":
  - It costs less to build
  - It requires less power
  - But we had to do some work to find the second form





- The method reduces a function in standard sum of products form to a set of ***prime implicants*** from which as many variables are eliminated as possible.
- These prime implicants are then examined to see if some are redundant.

- We will show how the Quine-McCluskey method can be used to find a minimal expansion equivalent to: $xyz + x\bar{y}z + \bar{x}yz + \bar{x}\,\bar{y}z + \bar{x}\,\bar{y}\,\bar{z}$.

- We will represent the minterms in this expansion by bit strings. The first bit will be 1 if x occurs and 0 if $\bar{x}$ occurs. The second bit will be 1 if y occurs and 0 if $\bar{y}$ occurs. The third bit will be 1 if z occurs and 0 if z occurs.
- We then group these terms according to the number of 1s in the corresponding bit strings. This information is shown in Table 1.

TABLE 1		
Minterm	*Bit String*	*Number of 1s*
xyz	111	3
$x\bar{y}z$	101	2
$\bar{x}yz$	011	2
$\bar{x}\,\bar{y}z$	001	1
$\bar{x}\,\bar{y}\,\bar{z}$	000	0

- Step 1:
  - Minterms that can be combined are those that differ in exactly one literal. Hence, two terms that can be combined differ by exactly one in the number of 1s in the bit strings that represent them.
  - When two minterms are combined into a product, this product contains two literals. A product in two literals is represented using a dash to denote the variable that does not occur.
  - For instance, the minterms $x\bar{y}z$ and $\bar{x}\,\bar{y}z$, represented by bit strings 101 and 001, can be combined into yz, represented by the string _01.

  - All pairs of minterms that can be combined and the product formed from these combinations are shown in Table 2.

TABLE 2					
			Step 1		
	Term	*Bit String*		*Term*	*String*
1	xyz	111	(1,2)	xz	1–1
2	$x\bar{y}z$	101	(1,3)	yz	–11
3	$\bar{x}yz$	011	(2,4)	$\bar{y}z$	–01
4	$\bar{x}\,\bar{y}z$	001	(3,4)	$\bar{x}z$	0–1
5	$\bar{x}\,\bar{y}\,\bar{z}$	000	(4,5)	$\bar{x}\,\bar{y}$	00–

- Step 2:
  - Next, all pairs of products of two literals that can be combined are combined into one literal. Two such products can be combined if they contain literals for the same two variables, and literals for only one of the two variables differ.
  - In terms of the strings representing the products, these strings must have a dash in the same position and must differ in exactly one of the other two slots.
  - We can combine the products yz and $\bar{y}z$, represented by the strings _11 and _01, into z, represented by the string __1.
    - We show all the combinations of terms that can be formed in this way in Table 3.

TABLE 3

Step 1			Step 2		
	Term	String		Term	String
(1,2)	xz	1–1	(1,2,3,4)	z	– –1
(1,3)	yz	–11			
(2,4)	$\bar{y}z$	–01			
(3,4)	$\bar{x}z$	0–1			
(4,5)	$\bar{x}\,\bar{y}$	00–			

- Step 3:
  - In Table 3 we also indicate which terms have been used to form products with fewer literals; these terms will not be needed in a minimal expansion.
  - The next step is to identify a minimal set of products needed to represent the Boolean function.
  - We begin with all those products that were not used to construct products with fewer literals.

- Next, we form Table 4, which has a row for each candidate product formed by combining original terms, and a column for each original term; and we put an X in a position if the original term in the sum-of-products expansion was used to form this candidate product.

TABLE 3
(Step 3)

	xyz	$x\bar{y}z$	$\bar{x}yz$	$\bar{x}\,\bar{y}z$	$\bar{x}\,\bar{y}\,\bar{z}$
z	X	X	X	X	
$\bar{x}\,\bar{y}$				X	X

- In this case, we say that the candidate product **covers** the original minterm. We need to include at least one product that covers each of the original minterms.
- Consequently, whenever there is only one X in a column in the table, the product corresponding to the row this X is in must be used.
- From Table 4 we see that both z and $\bar{x}\,\bar{y}$ are needed.
- Hence, the final answer is $z + \bar{x}\,\bar{y}$.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **Why is it important to simplify Boolean expressions?**

A. To increase the number of variables
B. To make expressions harder to interpret
C. To improve efficiency and reduce errors
D. To convert them into decimal format

Answer: C

Explanation: Simplified expressions are easier to understand, less error-prone, and more efficient in practical applications.

2. **Which of the following is NOT a method for minimizing Boolean expressions?**

A. Karnaugh Maps
B. Tabular Method
C. Algebraic Manipulation

D. Binary Expansion
Answer: D
Explanation: Binary expansion is not a recognized method for Boolean minimization.

3. **What does the Tabular Method primarily use to simplify Boolean expressions?**
A. Truth tables
B. Karnaugh maps
C. Prime implicant tables
D. ASCII codes
Answer: C
Explanation: The Tabular Method uses prime implicant tables to identify minimal expressions.

4. **In the Tabular Method, what does an 'X' in a table cell typically indicate?**
A. A variable is missing
B. A product term covers a minterm
C. A logic gate is required
D. A contradiction in logic
Answer: B
Explanation: An 'X' marks that a candidate product covers a specific minterm.

5. **What is the goal of identifying essential prime implicants in the Tabular Method?**
A. To increase the number of terms
B. To ensure all minterms are covered
C. To eliminate all variables
D. To convert expressions to hexadecimal
Answer: B
Explanation: Essential prime implicants are necessary to cover minterms not covered by other groupings.

6. **Which method is most suitable for simplifying Boolean expressions with more than 5 variables?**
A. Karnaugh Map
B. Tabular Method
C. Algebraic Manipulation
D. ASCII Encoding
Answer: B
Explanation: The Tabular Method is preferred for expressions with more than 5 variables due to complexity.

7. **What is the final simplified Boolean expression from the example in the tabular method?**
A. x + y
B. x * y
C. x - y

D. x / y
Answer: B
Explanation: The final answer derived from the table is x * y.

8. **What does a candidate product represent in the Tabular Method?**
A. A truth table row
B. A minimized literal
C. A combination of original terms
D. A binary digit
Answer: C
Explanation: Candidate products are formed by combining original terms to simplify expressions.

9. **What is the main challenge in parallelizing Boolean expressions for minimization?**
A. Lack of variables
B. Race conditions and deadlocks
C. Excessive memory usage
D. Slow input devices
Answer: B
Explanation: Concurrency introduces coding errors like race conditions and deadlocks.

10. **Which of the following is a benefit of using the Tabular Method over Karnaugh Maps?**
A. Easier for small variable sets
B. Better for visual grouping
C. Suitable for more than 5 variables
D. Requires no computation
Answer: C
Explanation: The Tabular Method handles large variable sets more effectively than Karnaugh Maps.

Fill in the Blank Questions

1. Boolean expressions are simplified to improve _____ and reduce errors.
Answer: efficiency

2. The Tabular Method uses _____ implicants to cover all minterms.
Answer: prime

3. An 'X' in the table indicates that a _____ product covers a minterm.
Answer: candidate

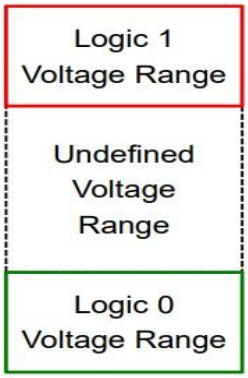
4. The final simplified expression from the example is _____.
Answer: x * y

5. The Tabular Method is preferred over Karnaugh Maps when there are more than _____ variables.
Answer: five

Unit 3 Physical properties of gates

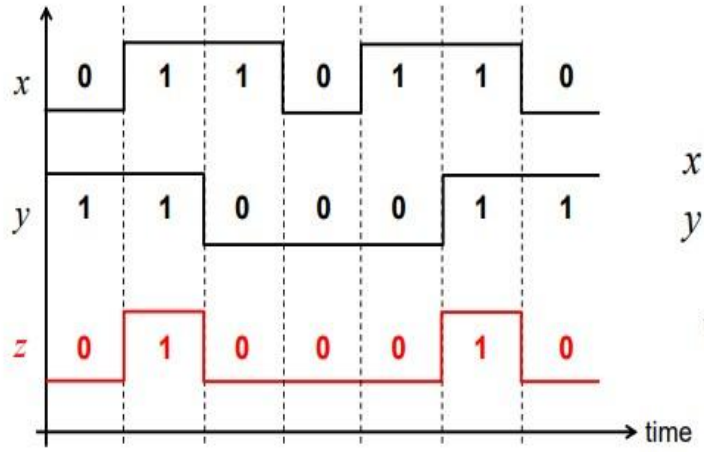
Voltage Levels

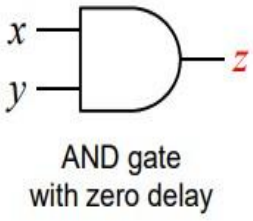
- Logic 1 is a range of voltage values
  - NOT just a single voltage value
- Logic 0 is also a range of voltages
  - Not just zero volt
- The voltage range between logic 0 and 1 is undefined
- Digital signals are not allowed to use voltage values in the undefined range



Timing Diagram

- Shows the logic values of signals in a circuit versus time
- **Waveform:** the shape of a signal over a period of time
- Example: timing diagram of an AND gate (with zero delay)

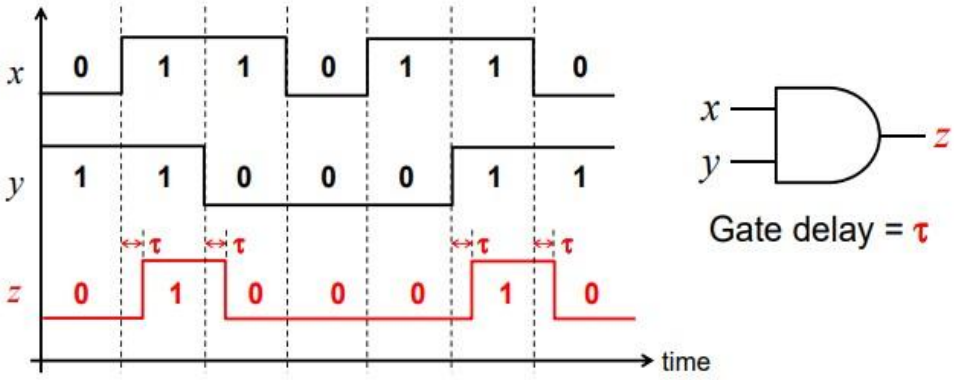




AND gate
with zero delay

Gate Delay

- A change in the inputs of a gate causes a change in its outputs
- However, the change in the output signal is **not instantaneous**
- There is a small delay between an input signal change and an output signal change, called **gate delay**



Propagation Delay in a Circuit

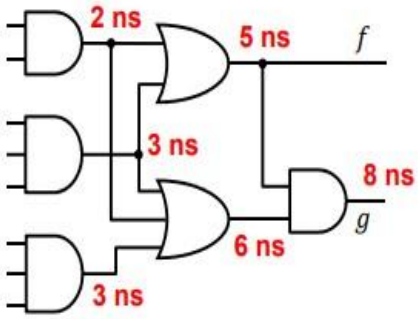
- In a given circuit, each gate has a delay
- The circuit has a **propagation delay** between inputs and outputs
- The propagation delay is computed along the **critical path**
- To compute the propagation delay, start at the inputs:

1. Delay at each gate output = Maximum input delay + Gate delay
2. Propagation delay of a circuit = maximum delay at any output

Computing the Maximum Circuit Delay

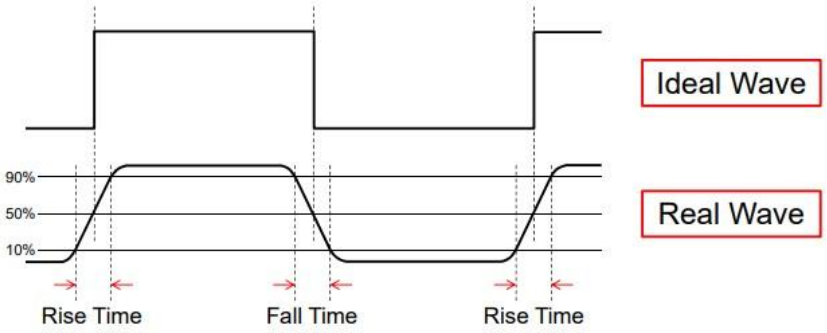
- Consider the following circuit with 8 inputs and 2 outputs
- Delay of a 2-input AND gate = 2 ns
- Delay of a 3-input AND gate = 3 ns
- Delay of a 2-input OR gate = 2 ns
- Delay of a 3-input OR gate = 3 ns

Compute the Maximum Circuit Delay



Rise-Time and Fall-Time

- In logic simulators, a waveform is drawn as an **ideal wave**
- The change from 0 to 1 (or from 1 to 0) is instantaneous
- In reality, a signal has a non-zero **rise-time** and **fall-time**
  - Time taken to change from 10% to 90% of High voltage (and vice versa)



Fan-In

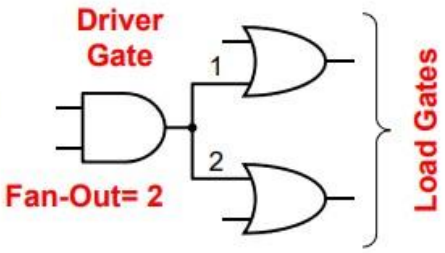
Fan-Out

- In digital circuits, it is common for the output of one gate (called **driver gate**) to be connect to the inputs of several **load gates**
- The **fan-out** of a gate is the number of gate inputs it can feed
- There is a limit on the maximum fan-out of a gate

The output of a driver gate can supply a limited amount of current.

Each input of a load gate consumes a certain amount of current.

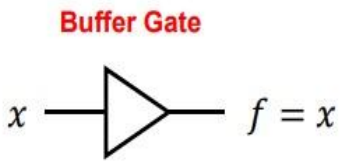
Therefore, the driver gate can only feed a limited number of load gates.

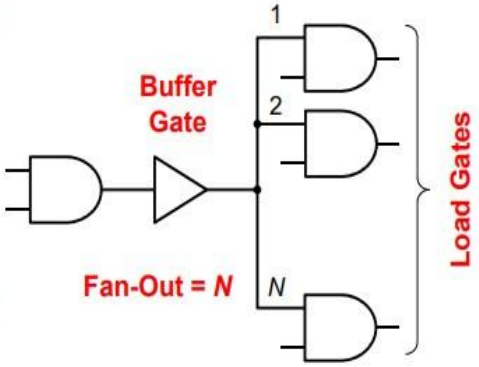


- The **fan-in** is the number of inputs to a gate
- Example: a 3-input AND gate has a Fan-in of 3
- Logic gates with a large fan-in tend to be slow
- Increasing the Fan-in of a gate increases the gate delay
- For example, a 3-input AND gate has a higher delay than a 2-input AND gate made with the same technology
- Using logic gates with higher fan-in is useful when reducing the depth (number of levels) of a logic circuit

Increasing the Fan-Out with a Buffer Gate

- Buffer Gate
  - Output f = Input x
- Buffer provides drive capability
  - Used to amplify an input signal
  - High current output
  - Increases the Fan-Out
- Buffer gate increases the propagation delay of a circuit





Tutor Marked Assignment

1. Find Product of Sum (POS) form of f (a, b, c, d) = $_{TT}$M (1, 4, 6, 9)
2. Simplify f (a , b , c) = $\sum$ m (0, 2, 5, 7)
3. Minimize the Boolean function F(A, B, C, D) = Σ m (0, 1, 2, 5, 7, 8, 9, 10, 13, 15)
4. Minimize the Boolean function F(A, B, C, D) = Σ m (1, 3, 4, 6, 8, 9, 11, 13, 15) + Σ d (0, 2, 14)
5. Minimize the Boolean function F(A, B, C, D) = Σ m (0, 2, 8, 10, 14) + Σ d (5, 15)

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What does gate delay refer to in digital circuits?**

A. The time a signal takes to travel through a wire
B. The time between input change and output response in a gate
C. The time required to power on a circuit
D. The time to store data in memory

Answer: B

Explanation: Gate delay is the time it takes for a gate's output to respond after its input changes.

2. **What is propagation delay in a digital circuit?**

A. The delay caused by the power supply
B. The delay between input and output across the entire circuit
C. The delay in signal transmission through a cable
D. The delay in memory access

Answer: B

Explanation: Propagation delay is the total delay from input to output across the critical path of a circuit.

3. **How is propagation delay calculated?**

A. By summing all gate delays

B. By measuring the longest delay at any output
C. By counting the number of gates
D. By checking the voltage level
Answer: B
Explanation: Propagation delay is the maximum delay at any output, typically along the critical path.

4. **What is the delay of a 2-input AND gate in the given example?**
A. 2 ns
B. 3 ns
C. 4 ns
D. 5 ns
Answer: B
Explanation: The delay of a 2-input AND gate is specified as 3 ns.

5. **Which gate has a higher delay due to increased fan-in?**
A. 2-input AND gate
B. 3-input AND gate
C. NOT gate
D. XOR gate
Answer: B
Explanation: Gates with higher fan-in, like a 3-input AND gate, have more delay than those with fewer inputs.

6. **What does fan-in refer to in digital logic design?**
A. Number of outputs a gate can drive
B. Number of inputs to a gate
C. Number of gates in a circuit
D. Number of bits in a signal
Answer: B
Explanation: Fan-in is the number of inputs connected to a logic gate.

7. **What is fan-out in digital circuits?**
A. Number of gates in a circuit
B. Number of outputs a gate can produce
C. Number of gate inputs a gate's output can feed
D. Number of bits in a signal
Answer: C
Explanation: Fan-out is the number of gate inputs that a single gate output can drive.

8. **Why is there a limit to fan-out in digital circuits?**
A. Due to voltage drop
B. Due to current supply limitations
C. Due to signal frequency
D. Due to gate size
Answer: B
Explanation: A gate can only supply a limited amount of current, limiting how many gates it can drive.

9. **What is the purpose of a buffer gate?**
A. To reduce voltage

B. To increase fan-out and drive capability
C. To store data
D. To convert analog to digital
Answer: B
Explanation: Buffer gates amplify signals and increase the number of gates that can be driven.

10. **What effect does a buffer gate have on propagation delay?**
A. It eliminates delay
B. It reduces delay
C. It increases delay
D. It has no effect
Answer: C
Explanation: Buffer gates increase propagation delay due to added circuitry.

Fill in the Blank Questions

1. Gate delay is the time between a change in _____ and the corresponding change in output.
Answer: input

2. Propagation delay is calculated along the _____ path of a circuit.
Answer: critical

3. Fan-in refers to the number of _____ connected to a logic gate.
Answer: inputs

4. Fan-out refers to the number of gate _____ a gate's output can feed.
Answer: inputs

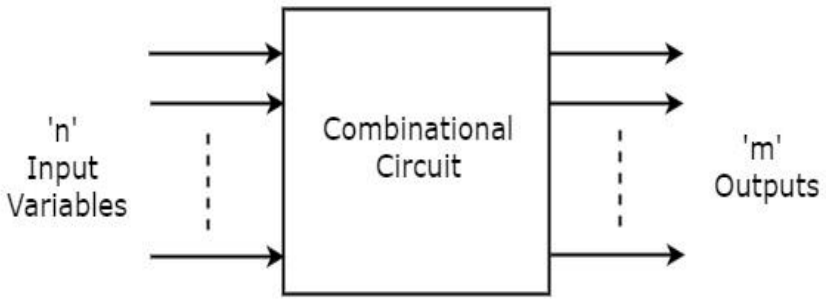
5. A _____ gate is used to amplify signals and increase drive capability.
Answer: buffer

Module 3 Combinational and Sequential Circuits

Unit 1 Combinational Circuits and Design Procedure
Unit 2 Binary Subtractor
Unit 3 Multiplexers
Unit 4 De-multiplexers
Unit 5 Decoders
Unit 6 Encoders
Unit 7 Latches
Unit 8 Flip-Flops

Unit 1 Combinational Circuits and Design Procedure

Combinational circuits consist of Logic gates. These circuits operate with binary values. The outputs of combinational circuit depends on the combination of present inputs. The following figure shows the **block diagram** of combinational circuit.



This combinational circuit has ‘n’ input variables and ‘m’ outputs. Each combination of input variables will affect the outputs.
Design procedure of Combinational circuits

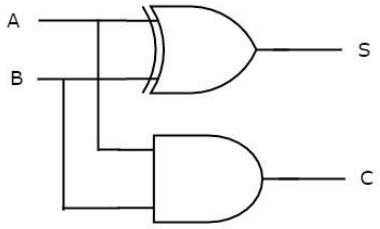
- Find the required number of input variables and outputs from given specifications.
- Formulate the **Truth table**. If there are ‘n’ input variables, then there will be 2n possible combinations. For each combination of input, find the output values.
- Find the **Boolean expressions** for each output. If necessary, simplify those expressions.
- Implement the above Boolean expressions corresponding to each output by using **Logic gates**.

Binary Adder
The most basic arithmetic operation is addition. The circuit, which performs the addition of two binary numbers is known as Binary adder. First, let us implement an adder, which performs the addition of two bits.
Half Adder

Half adder is a combinational circuit, which performs the addition of two binary numbers A and B are of single bit. It produces two outputs sum, S & carry, C.
The Truth table of Half adder is shown below.

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

When we do the addition of two bits, the resultant sum can have the values ranging from 0 to 2 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent decimal digit 2 with single bit in binary. So, we require two bits for representing it in binary.
When we do the addition of two bits, the resultant sum can have the values ranging from 0 to 2 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent decimal digit 2 with single bit in binary. So, we require two bits for representing it in binary.
Let, sum, S is the Least significant bit and carry, C is the Most significant bit of the resultant sum. For first three combinations of inputs, carry, C is zero and the value of S will be either zero or one based on the **number of ones** present at the inputs. But, for last combination of inputs, carry, C is one and sum, S is zero, since the resultant sum is two.
From Truth table, we can directly write the **Boolean functions** for each output as
$S = A \oplus B$
$C = AB$
We can implement the above functions with 2-input Ex-OR gate & 2-input AND gate. The **circuit diagram** of Half adder is shown in the following figure.



In the above circuit, a two input Ex-OR gate & two input AND gate produces sum, S & carry, C respectively. Therefore, Half-adder performs the addition of two bits.

Full Adder
Full adder is a combinational circuit, which performs the **addition of three bits** A, B and C_{in}. Where, A & B are the two parallel significant bits and C_{in} is the carry bit, which is generated from previous stage. This Full

adder also produces two outputs sum, S & carry, C_{out}, which are similar to Half adder.

The **Truth table** of Full adder is shown below.

Inputs			**Outputs**	
A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0	0	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	1	1

When we do the addition of three bits, the resultant sum can have the values ranging from 0 to 3 in decimal. We can represent the decimal digits 0 and 1 with single bit in binary. But, we can't represent the decimal digits 2 and 3 with single bit in binary. So, we require two bits for representing those two decimal digits in binary.

Let, sum, S is the Least significant bit and carry, C_{out} is the Most significant bit of resultant sum. It is easy to fill the values of outputs for all combinations of inputs in the truth table. Just count the **number of ones** present at the inputs and write the equivalent binary number at outputs. If C_{in} is equal to zero, then Full adder truth table is same as that of Half adder truth table.

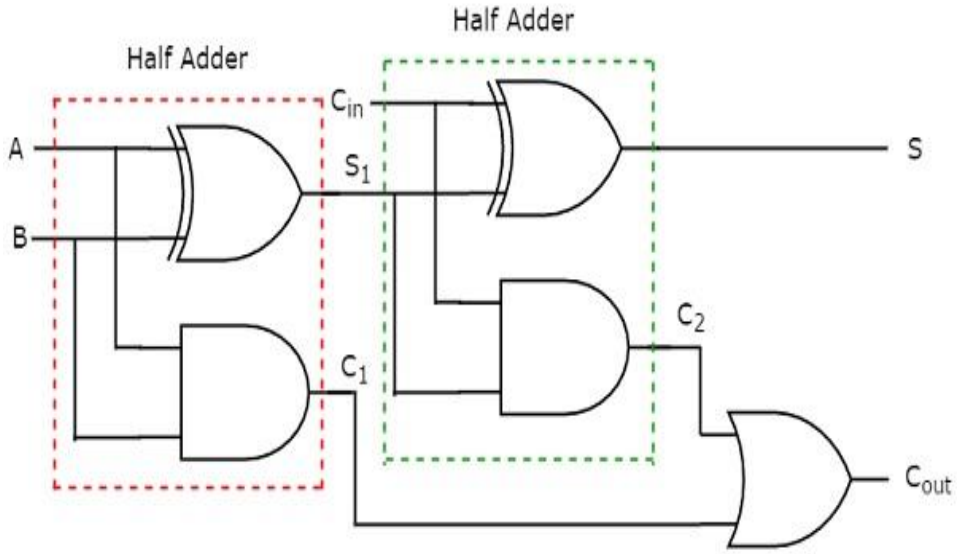
We will get the following **Boolean functions** for each output after simplification.

$S = A \oplus B \oplus C_{in}$

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

The sum, S is equal to one, when odd number of ones present at the inputs. We know that Ex-OR gate produces an output, which is an odd function. So, we can use either two 2input Ex-OR gates or one 3-input Ex-OR gate in order to produce sum, S. We can implement carry, C_{out} using two 2-input AND gates & one OR gate. The **circuit diagram** of Full adder is shown in the following figure.

This adder is called as **Full adder** because for implementing one Full



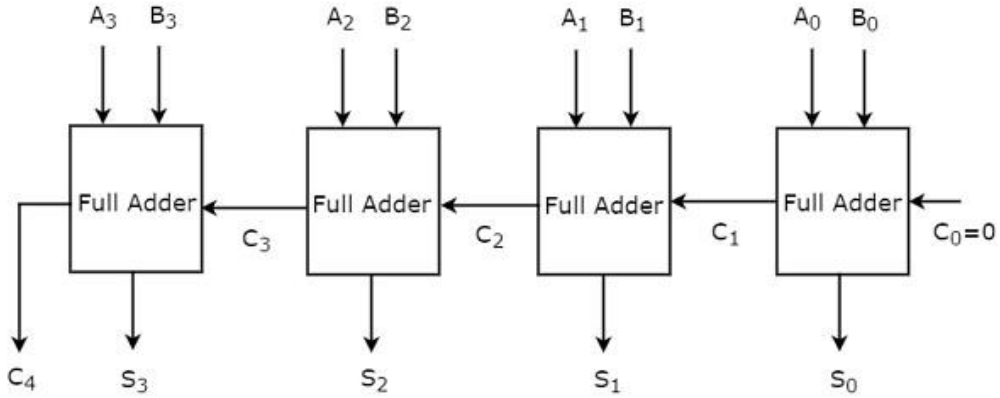
adder, we require two Half adders and one OR gate. If C_{in} is zero, then Full adder becomes Half adder. We can verify it easily from the above circuit diagram or from the Boolean functions of outputs of Full adder.

4-bit Binary Adder

The 4-bit binary adder performs the **addition of two 4-bit numbers**. Let the 4-bit binary numbers, A = $A_3A_2A_1A_0$ and B = $B_3B_2B_1B_0$. We can implement 4-bit binary adder in one of the two following ways.

- Use one Half adder for doing the addition of two Least significant bits and three Full adders for doing the addition of three higher significant bits.
- Use four Full adders for uniformity. Since, initial carry C_{in} is zero, the Full adder which is used for adding the least significant bits becomes Half adder.

For the time being, we considered second approach. The **block diagram** of 4-bit binary adder is shown in the following figure.



Here, the 4 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs A & B. The carry output of one Full adder will be the carry input of subsequent higher order Full adder. This

4-bit binary adder produces the resultant sum having at most 5 bits. So, carry out of last stage Full adder will be the MSB.
In this way, we can implement any higher order binary adder just by cascading the required number of Full adders. This binary adder is also called as **ripple carry** binary **adder** because the carry propagates ripples from one stage to the next stage.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions

1. What defines a combinational logic circuit?
A. It stores data
B. Its output depends only on current inputs
C. It has memory elements
D. It operates sequentially

Answer: B

2. Which of the following is a basic combinational circuit?
A. Flip-flop
B. Counter
C. Multiplexer
D. Register

Answer: C

3. What is the function of a decoder?
A. To store binary data
B. To convert binary input into a unique output line
C. To perform arithmetic operations
D. To generate clock signals

Answer: B

4. Which combinational circuit selects one input from many?
A. Decoder
B. Demultiplexer
C. Comparator
D. Multiplexer

Answer: E

5. What does a half adder do?
A. Multiplies two bits
B. Adds two bits and gives sum and carry
C. Subtracts two bits
D. Stores binary numbers

Answer: B

6. Which logic gates are used in a half adder?
A. AND and OR
B. XOR and AND
C. NAND and NOR
D. NOT and XOR

Answer: B

7. What is the difference between a half adder and a full adder?

A. Full adder adds three bits including carry-in
B. Half adder is faster
C. Full adder uses fewer gates
D. Half adder stores data
Answer: A
8. What is the output of a 2-to-4 decoder?
A. 2 lines
B. 4 lines
C. 8 lines
D. 1 line
Answer: B
9. Which circuit distributes one input to multiple outputs?
A. Multiplexer
B. Demultiplexer
C. Decoder
D. Adder
Answer: B
10. What is the purpose of a comparator?
A. To compare two binary numbers
B. To store data
C. To decode signals
D. To generate clock pulses
Answer: A

Fill in the Blank Questions

1. A _____ logic circuit's output depends only on current inputs. → *combinational*
2. A _____ adds two bits and produces sum and carry. → *half adder*
3. A _____ selects one input from many inputs. → *multiplexer*
4. A _____ converts binary input into a unique output line. → *decoder*

Unit 2 Binary Subtractor

The circuit, which performs the subtraction of two binary numbers is known as **Binary subtractor**. We can implement Binary subtractor in following two methods.

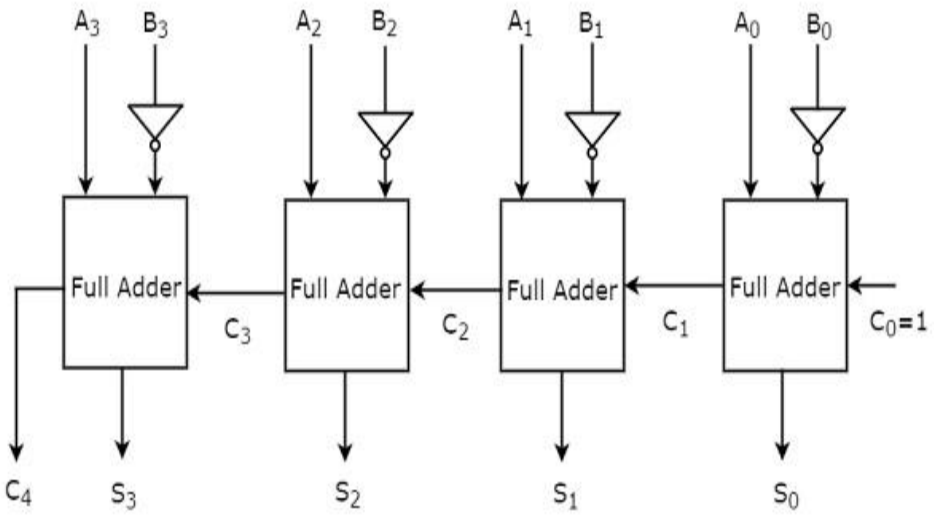
- Cascade Full subtractors
- 2's complement method

In first method, we will get an n-bit binary subtractor by cascading 'n' Full subtractors. So, first you can implement Half subtractor and Full subtractor, similar to Half adder & Full adder. Then, you can implement an n-bit binary subtractor, by cascading 'n' Full subtractors. So, we will be having two separate circuits for binary addition and subtraction of two binary numbers.

In second method, we can use same binary adder for subtracting two binary numbers just by doing some modifications in the second input. So, internally binary addition operation takes place but, the output is resultant subtraction.
We know that the subtraction of two binary numbers A & B can be written as,
$A - B = A + (2'$s compliment of B)
$\Rightarrow A - B = A + (1'$s compliment of B) + 1

4-bit Binary Subtractor

The 4-bit binary subtractor produces the **subtraction of two 4-bit numbers**. Let the 4bit binary numbers, $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$. Internally, the operation of 4-bit Binary subtractor is similar to that of 4-bit Binary adder. If the normal bits of binary number A, complemented bits of binary number B and initial carry borrow, C_{in} as one are applied to 4-bit Binary adder, then it becomes 4-bit Binary subtractor. The **block diagram** of 4-bit binary subtractor is shown in the following figure.



This 4-bit binary subtractor produces an output, which is having at most 5 bits. If Binary number A is greater than Binary number B, then MSB of the output is zero and the remaining bits hold the magnitude of A-B. If Binary number A is less than Binary number B, then MSB of the output is one. So, take the 2's complement of output in order to get the magnitude of A-B.

In this way, we can implement any higher order binary subtractor just by cascading the required number of Full adders with necessary modifications.

Binary Adder / Subtractor

The circuit, which can be used to perform either addition or subtraction of two binary numbers at any time is known as **Binary Adder / subtractor**. Both, Binary adder and Binary subtractor contain a set of Full adders, which are cascaded. The input bits of binary number A are directly applied in both Binary adder and Binary subtractor.

There are two differences in the inputs of Full adders that are present in Binary adder and Binary subtractor.

- The input bits of binary number B are directly applied to Full adders in Binary adder, whereas the complemented bits of binary number B are applied to Full adders in Binary subtractor.
- The initial carry, $C_0 = 0$ is applied in 4-bit Binary adder, whereas the initial carry borrow, $C_0 = 1$ is applied in 4-bit Binary subtractor.

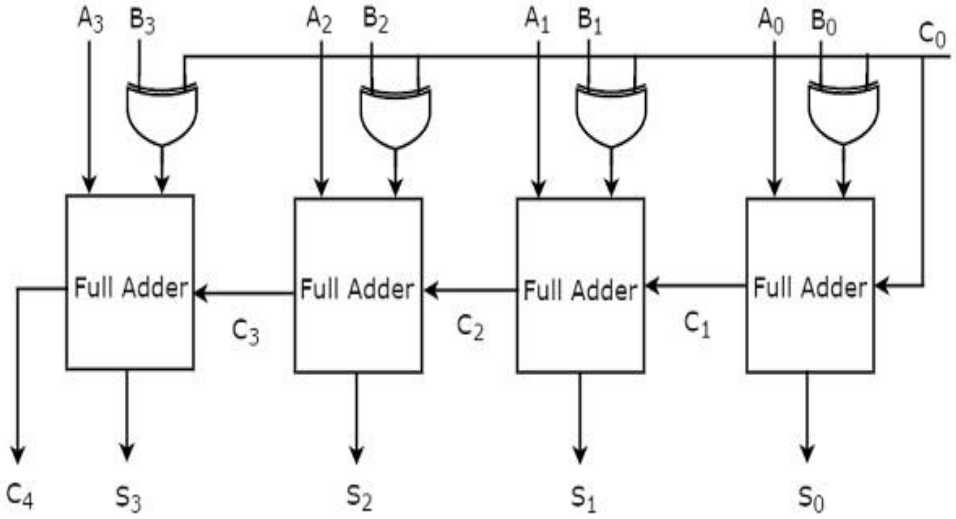
We know that a **2-input Ex-OR gate** produces an output, which is same as that of first input when other input is zero. Similarly, it produces an output, which is complement of first input when other input is one.

Therefore, we can apply the input bits of binary number B, to 2-input Ex-OR gates. The other input to all these Ex-OR gates is C_0. So, based on the value of C_0, the Ex-OR gates produce either the normal or complemented bits of binary number B.

4-bit Binary Adder / Subtractor

The 4-bit binary adder / subtractor produces either the addition or the subtraction of two 4-bit numbers based on the value of initial carry or borrow, C_0. Let the 4-bit binary numbers, A = $A_3A_2A_1A_0$ and B = $B_3B_2B_1B_0$. The operation of 4-bit Binary adder / subtractor is similar to that of 4-bit Binary adder and 4-bit Binary subtractor.

Apply the normal bits of binary numbers A and B & initial carry or borrow, C_0 from externally to a 4-bit binary adder. The **block diagram** of 4-bit binary adder / subtractor is shown in the following figure.



If initial carry, C_0 is zero, then each full adder gets the normal bits of binary numbers A & B. So, the 4-bit binary adder / subtractor produces an output, which is the **addition of two binary numbers** A & B.

If initial borrow, C_0 is one, then each full adder gets the normal bits of binary number A & complemented bits of binary number B. So, the 4-bit binary adder / subtractor produces an output, which is the **subtraction of two binary numbers** A & B.

Therefore, with the help of additional Ex-OR gates, the same circuit can be used for both addition and subtraction of two binary numbers.

SELF –ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the main function of a binary subtractor?**

A. To multiply binary numbers
B. To divide binary numbers
C. To subtract binary numbers
D. To convert binary to decimal

Answer: C

Explanation: A binary subtractor performs subtraction between two binary numbers.

2. **Which method uses a binary adder to perform subtraction?**

A. Cascade full subtractors

B. 2’s complement method
C. Decimal subtraction
D. Half subtractor method
Answer: B
Explanation: The 2’s complement method modifies the second input and uses a binary adder to perform subtraction.

3. **What is the result of subtracting B from A using 2’s complement?**
A. A – B = A + B
B. A – B = A + (1’s complement of B)
C. A – B = A + (2’s complement of B)
D. A – B = A – (2’s complement of B)
Answer: C
Explanation: Subtraction is performed by adding the 2’s complement of B to A.

4. **How many bits can the output of a 4-bit binary subtractor have at most?**
A. 4
B. 5
C. 6
D. 8
Answer: B
Explanation: The output may include an extra bit for the sign, making it up to 5 bits.

5. **What does the MSB of the output indicate in a binary subtractor?**
A. The parity of the result
B. The overflow condition
C. Whether A is greater than B
D. The carry bit
Answer: C
Explanation: MSB indicates whether A is greater than B (0) or less than B (1).

6. **What is the role of Ex-OR gates in a binary adder/subtractor circuit?**
A. To perform multiplication
B. To generate carry bits
C. To complement bits based on control input
D. To store intermediate results
Answer: C
Explanation: Ex-OR gates produce either normal or complemented bits of B based on the control input C_0.

7. **What value of C_0 is used for subtraction in a binary adder/subtractor?**
A. 0
B. 1

C. Depends on A
D. Depends on B
Answer: B
Explanation: $C_0 = 1$ is used to initiate subtraction using the 2's complement method.

8. **Which component is used in both binary adder and subtractor circuits?**
A. Decoder
B. Full adder
C. Multiplexer
D. Comparator
Answer: B
Explanation: Full adders are used in both addition and subtraction operations.

9. **What happens when A < B in a binary subtractor?**
A. The result is negative
B. The result is zero
C. The result is undefined
D. The result is positive
Answer: A
Explanation: When A is less than B, the MSB is 1, indicating a negative result.

10. **What is the advantage of using a binary adder/subtractor circuit?**
A. It reduces power consumption
B. It simplifies hardware design
C. It performs multiplication
D. It increases memory size
Answer: B
Explanation: A single circuit can perform both addition and subtraction, simplifying design.

Fill in the Blank Questions

1. A binary subtractor can be implemented using either cascade full subtractors or the _____ method.
Answer: 2's complement

2. The 2's complement of a binary number is obtained by taking the 1's complement and adding _____.
Answer: 1

3. In a 4-bit binary subtractor, the MSB of the output indicates whether A is _____ than B.
Answer: greater

4. The control input C_0 is set to _____ to perform subtraction in a binary adder/subtractor.
Answer: 1

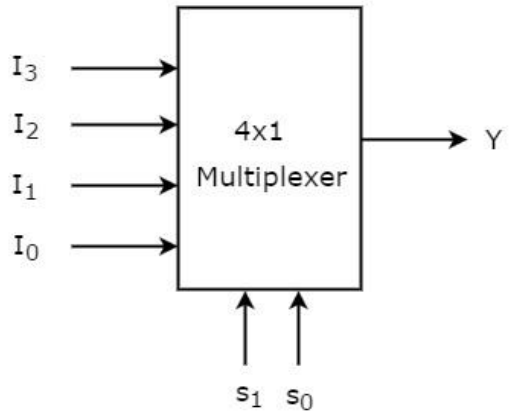
5. Ex-OR gates are used to produce either normal or _____ bits of B based on the value of C_0.
Answer: complemented

Unit 3: Multiplexers

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.
Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure. One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

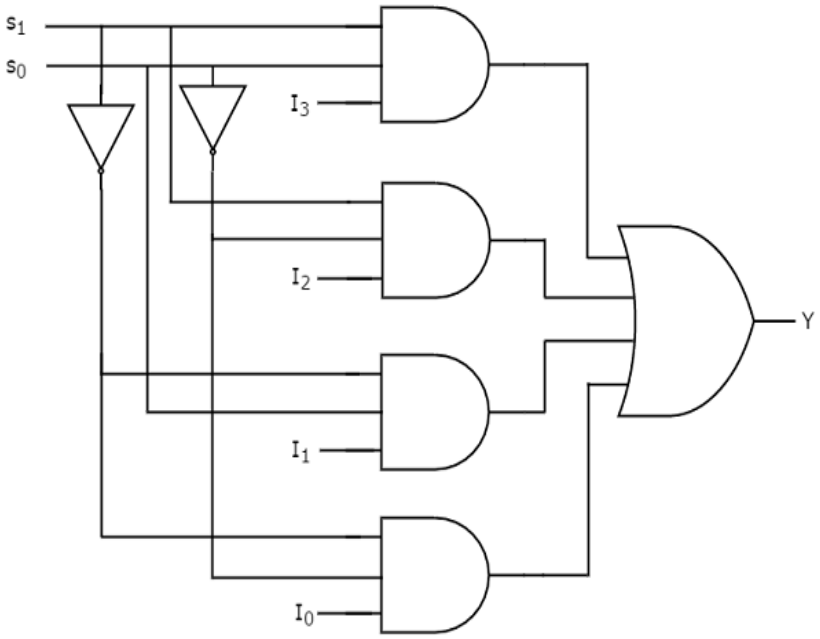


Selection Lines		**Output**
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$Y = S_1' S_0' I_0 + S_1' S_0 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the



following figure.

We can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

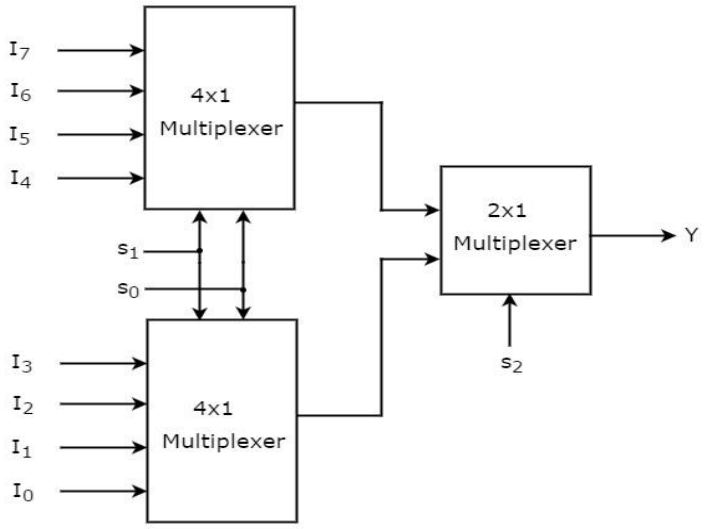
Implementation of Higher-order Multiplexers.
Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer
In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.
So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.
Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.
The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers.



The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.
The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

16x1 Multiplexer

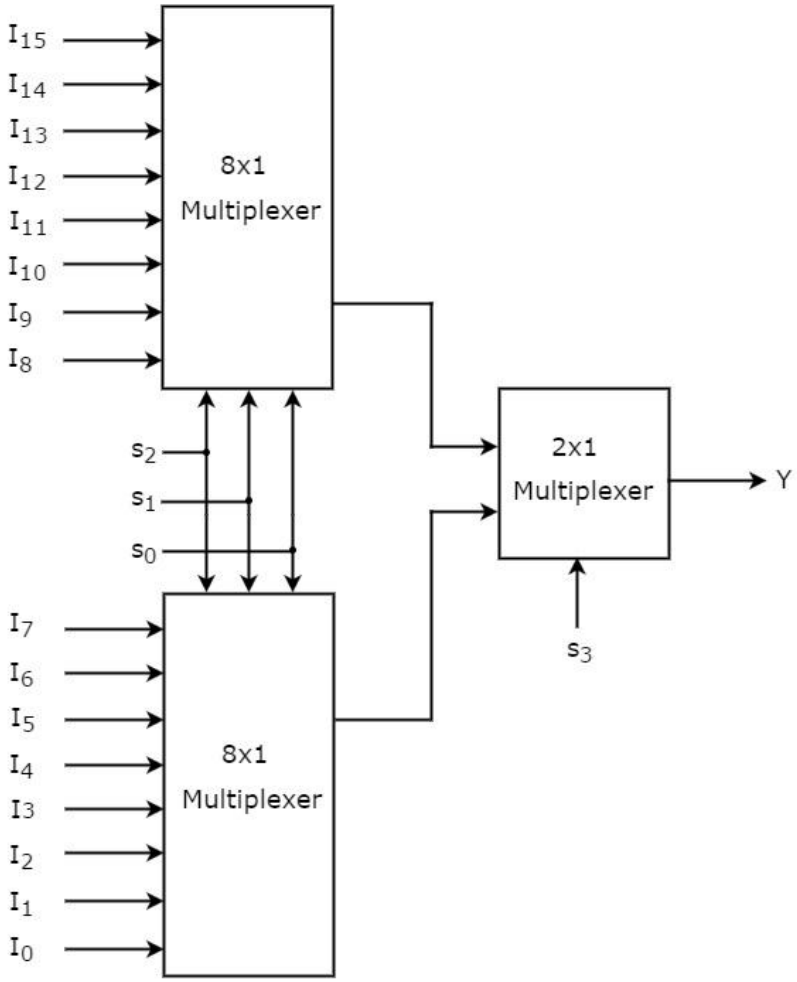
In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.

So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs				**Output**
S_3	**S_2**	**S_1**	**S_0**	**Y**
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table.
The **block diagram** of 16x1 Multiplexer is shown in the following figure.



The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.
The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is$_7$ to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the primary function of a multiplexer?**

A. To perform arithmetic operations
B. To select one of many inputs and forward it to the output
C. To store binary data
D. To convert analog signals to digital

Answer: B

Explanation: A multiplexer selects one of several input signals and forwards it to a single output line based on selection inputs.

2. **How many data inputs does a 4x1 multiplexer have?**

A. 2
B. 4
C. 8
D. 1

Answer: B

Explanation: A 4x1 multiplexer has 4 data inputs and 2 selection lines.

3. **What is the Boolean expression for the output of a 4x1 multiplexer?**

A. Y = S1S0I0 + S1S0I1 + S1S0I2 + S1S0I3
B. Y = S1′S0′I0 + S1′S0I1 + S1S0′I2 + S1S0I3
C. Y = I0 + I1 + I2 + I3
D. Y = S1 + S0 + I0

Answer: B

Explanation: The output is determined by the combination of selection lines and corresponding input.

4. **How many selection lines are needed for an 8x1 multiplexer?**

A. 2
B. 3
C. 4
D. 1

Answer: B

Explanation: An 8x1 multiplexer requires 3 selection lines to choose among 8 inputs.

5. **What is the role of the 2x1 multiplexer in an 8x1 multiplexer implementation using 4x1 multiplexers?**

A. To generate selection lines
B. To combine outputs from the 4x1 multiplexers
C. To store data
D. To perform logical operations

Answer: B

Explanation: The 2x1 multiplexer selects between the outputs of the two 4x1 multiplexers.

6. **How many data inputs does a 16x1 multiplexer have?**

A. 8

B. 4
C. 16
D. 32
Answer: C
Explanation: A 16x1 multiplexer has 16 data inputs and 4 selection lines.
7. **Which selection line determines whether the upper or lower 8x1 multiplexer is selected in a 16x1 multiplexer?**
A. S0
B. S1
C. S2
D. S3
Answer: D
Explanation: S3 is used by the 2x1 multiplexer to select between the outputs of the two 8x1 multiplexers.
8. **What is the total number of possible input combinations for a 4x1 multiplexer?**
A. 2
B. 4
C. 8
D. 16
Answer: B
Explanation: With 2 selection lines, there are $2^2 = 4$ possible combinations.
9. **What is the advantage of using lower-order multiplexers to build higher-order ones?**
A. Reduces power consumption
B. Simplifies circuit design
C. Increases memory
D. Improves signal strength
Answer: B
Explanation: Using lower-order multiplexers allows modular and scalable design of complex circuits.
10. **What does the output of a multiplexer depend on?**
A. The number of gates
B. The selection line values
C. The clock signal
D. The power supply
Answer: B
Explanation: The output is determined by the values of the selection lines.
Fill in the Blank Questions
1. A multiplexer with n selection lines can handle up to _____ data inputs.
Answer: 2^n
2. A 4x1 multiplexer uses _____ selection lines.
Answer: 2

3. The Boolean expression for a 4x1 multiplexer output is derived from its _____ table.
Answer: truth
4. In a 16x1 multiplexer, the selection line _____ chooses between the two 8x1 multiplexers.
Answer: S3
5. The output of a multiplexer is determined by the values of the _____ lines.
Answer: selection

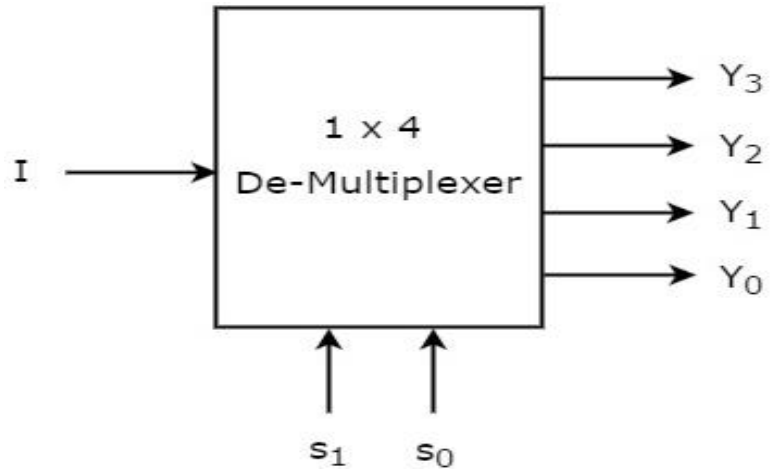
Unit 4 De-Multiplexers

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

1x4 De-Multiplexer

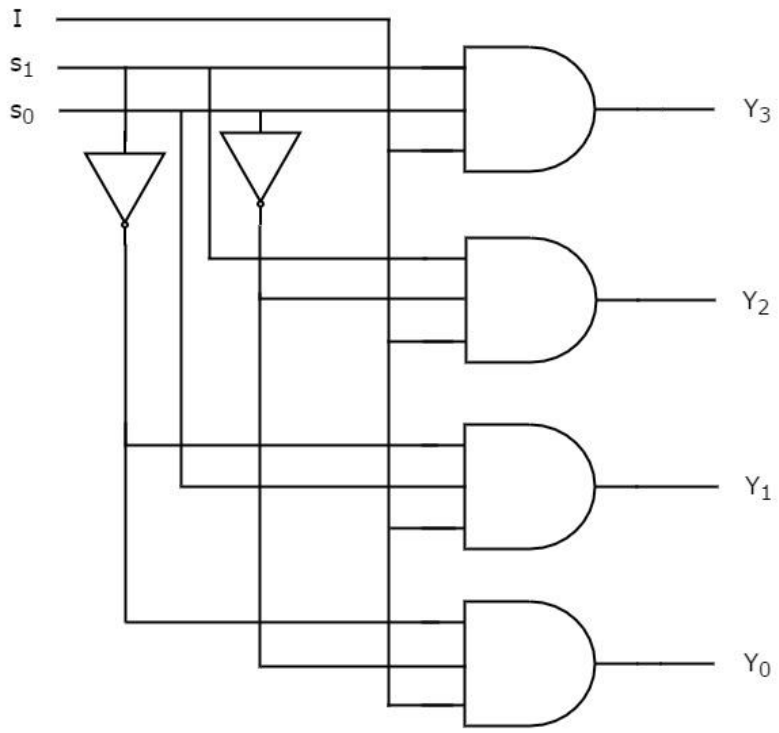
1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.
The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.



Selection Inputs		**Outputs**			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output as
$Y_3 = s_1 s_0 I$
$Y_2 = s_1 s_0' I$
$Y_1 = s_1' s_0 I$
$Y_0 = s_1' s_0' I$

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order De-Multiplexers

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

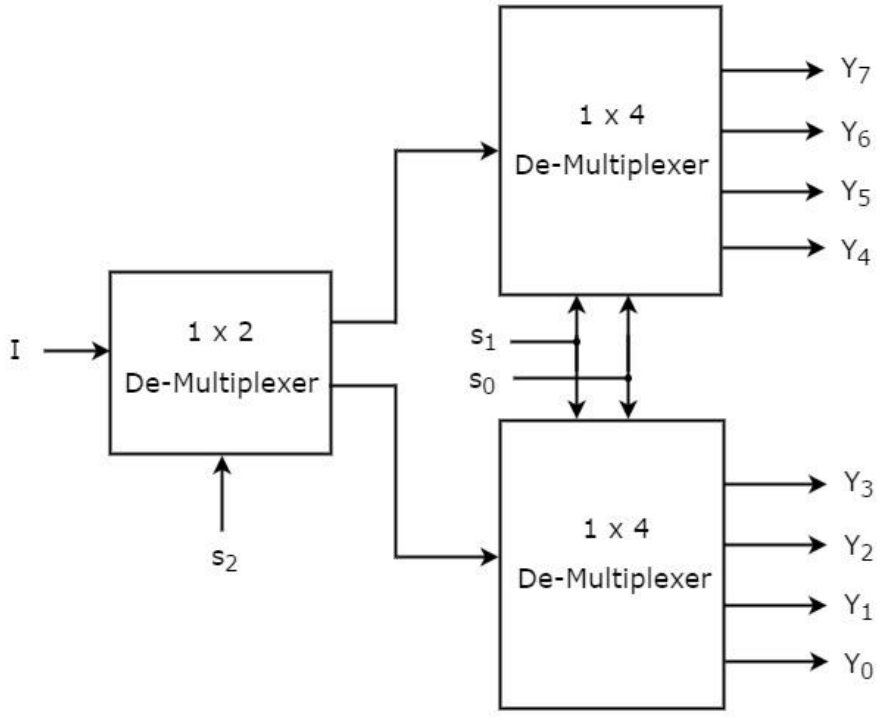
1x8 De-Multiplexer

In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

Selection Inputs			Outputs							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	**1**
0	0	1	0	0	0	0	0	0	**1**	0
0	1	0	0	0	0	0	0	**1**	0	0
0	1	1	0	0	0	0	**1**	0	0	0
1	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	0	**1**	0	0	0	0	0
1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	**1**	0	0	0	0	0	0	0

So, we require two **1x4 De-Multiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexe.
Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0. The **Truth table** of 1x8 De-Multiplexer is shown below.

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 De-Multiplexer is shown in the following figure.



The common **selection lines, s_1 & s_0** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.

The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

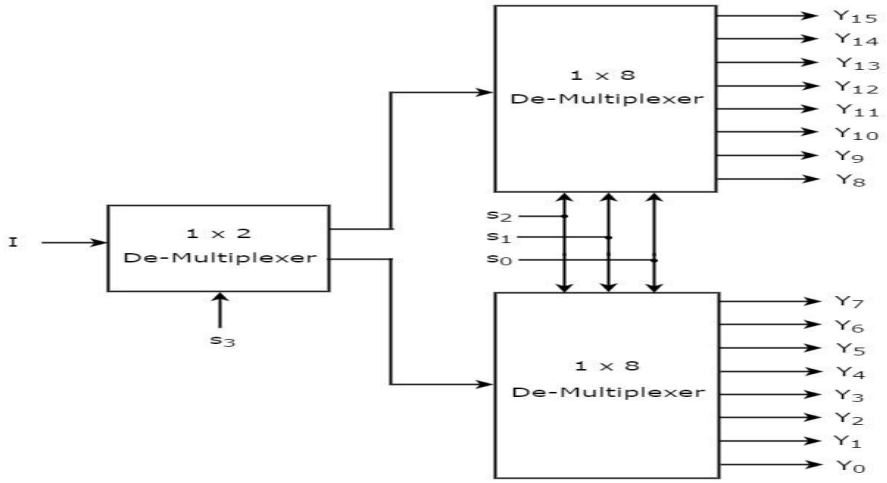
1x16 De-Multiplexer

In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs. So, we require two **1x8 De-Multiplexers** in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & s_0 and outputs Y_{15} to Y_0. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.

The common **selection lines s_2, s_1 & s_0** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 DeMultiplexer are Y_7 to Y_0.

The other **selection line, s_3** is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0.



SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the primary function of a De-Multiplexer?**

A. To combine multiple inputs into one output
B. To convert analog signals to digital
C. To distribute a single input to one of many outputs
D. To store binary data

Answer: C
E.
Explanation: A De-Multiplexer routes a single input to one of several outputs based on selection lines.

2. **How many outputs does a 1x4 De-Multiplexer have?**

A. 2
B. 4
C. 8
D. 1

Answer: B
Explanation: A 1x4 De-Multiplexer has 4 outputs and 2 selection lines.

3. **What is the Boolean expression for output Y0 in a 1x4 De-Multiplexer?**

A. s1 \cdot s0 \cdot I
B. s1' \cdot s0' \cdot I
C. s1 \cdot s0' \cdot I
D. s1' \cdot s0 \cdot I

Answer: B
Explanation: Y0 is active when both selection lines are 0, so the expression is s1' \cdot s0' \cdot I.

4. **How many selection lines are needed for a 1x8 De-Multiplexer?**

A. 2
B. 3
C. 4
D. 1

Answer: B
Explanation: A 1x8 De-Multiplexer requires 3 selection lines to choose among 8 outputs.

5. **What is the role of the 1x2 De-Multiplexer in the 1x8 De-Multiplexer implementation?**

A. To generate selection lines
B. To split the input into two paths for the 1x4 De-Multiplexers
C. To store data
D. To perform logical operations
E.
Answer: B

Explanation: The 1x2 De-Multiplexer directs the input to one of the two 1x4 De-Multiplexers.

6. **How many outputs does a 1x16 De-Multiplexer have?**

A. 8
B. 4
C. 16
D. 32

Answer: C

Explanation: A 1x16 De-Multiplexer has 16 outputs and 4 selection lines.

7. **Which selection line determines whether the upper or lower 1x8 De-Multiplexer is selected in a 1x16 De-Multiplexer?**

A. S0
B. S1
C. S2
D. S3

Answer: D

Explanation: S3 is used by the 1x2 De-Multiplexer to select between the two 1x8 De-Multiplexers.

8. **What is the total number of possible output combinations for a 1x4 De-Multiplexer?**

A. 2
B. 4
C. 8
D. 16

Answer: B

Explanation: With 2 selection lines, there are $2^2 = 4$ possible output paths.

9. **What is the advantage of using lower-order De-Multiplexers to build higher-order ones?**

A. Reduces power consumption
B. Simplifies circuit design
C. Increases memory
D. Improves signal strength

Answer: B

Explanation: Using lower-order De-Multiplexers allows modular and scalable design of complex circuits.

10. **What determines which output line is activated in a De-Multiplexer?**

A. The number of gates
B. The selection line values
C. The clock signal
D. The power supply
E.

Answer: B

Explanation: The output is determined by the values of the selection lines.

Fill in the Blank Questions

1. A De-Multiplexer with n selection lines can have up to _____ outputs.

Answer: 2^n

2. A 1x4 De-Multiplexer uses _____ selection lines.

Answer: 2

3. The Boolean expression for output Y3 in a 1x4 De-Multiplexer is _____.

Answer: s1 \cdot s0 \cdot I

4. In a 1x16 De-Multiplexer, the selection line _____ chooses between the two 1x8 De-Multiplexers.

Answer: S3

5. The output of a De-Multiplexer is determined by the values of the _____ lines.

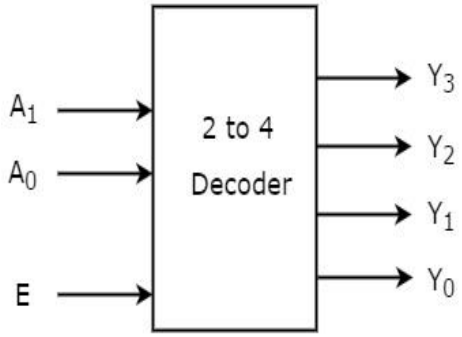
Answer: selection

Unit 5 Decoder

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables *lines*, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

Enable	**Inputs**		**Outputs**			
E	**A_1**	**A_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

$Y_3 = E.A_1.A_0$

$Y_2 = E.A_1.A_0'$

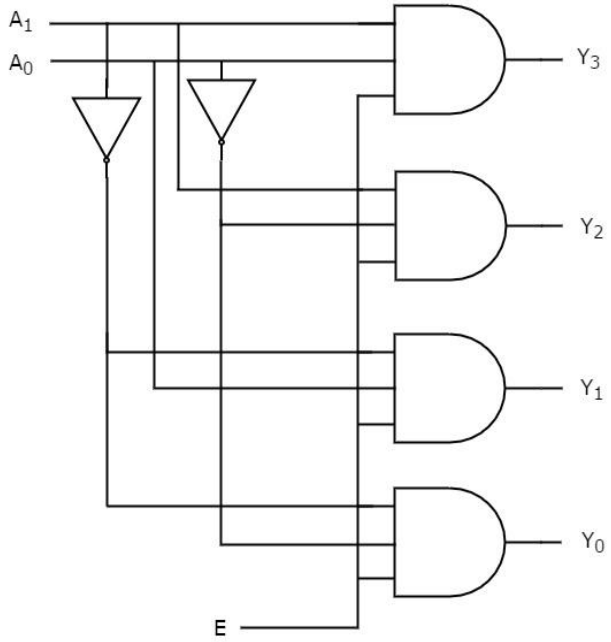
$Y_1 = E.A_1'.A_0$

$Y_0 = E.A_1'.A_0'$

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.

Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E



is zero, then all the outputs of decoder will be equal to zero.
Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.
Implementation of Higher-order Decoders
Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

3 to 8 Decoder

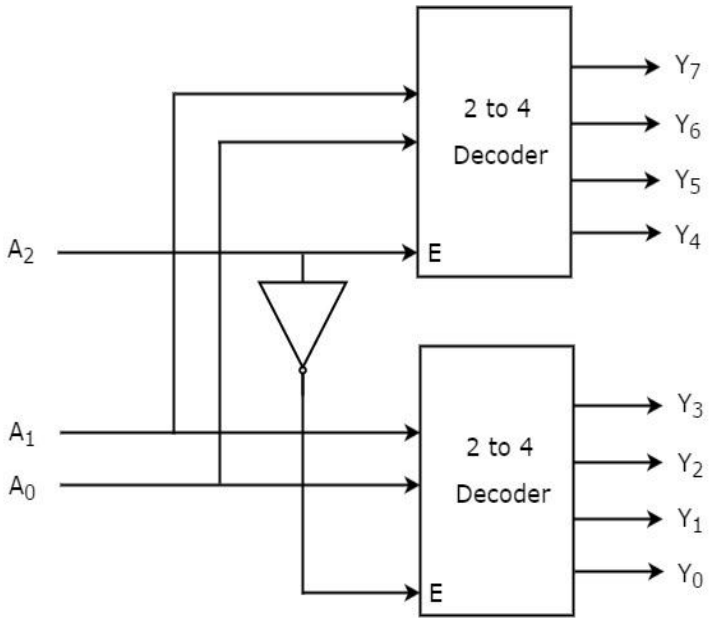
In this section, let us implement **3 to 8 decoder using 2 to 4 decoders**. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.
We can find the number of lower order decoders required for implementing higher order decoder using the following formula.
Required number of lower order decoders = m_2 / m_1
Where,
m_1 is the number of outputs of lower order decoder.
m_2 is the number of outputs of higher order decoder.

Here, $m_1 = 4$ and $m_2 = 8$. Substitute, these two values in the above formula.

Required number of 2 *to* 4 *decoders* = 8 / 4 = 2

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.

The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**.



The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

4 to 16 Decoder

In this section, let us implement **4 to 16 decoder using 3 to 8 decoders**. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0

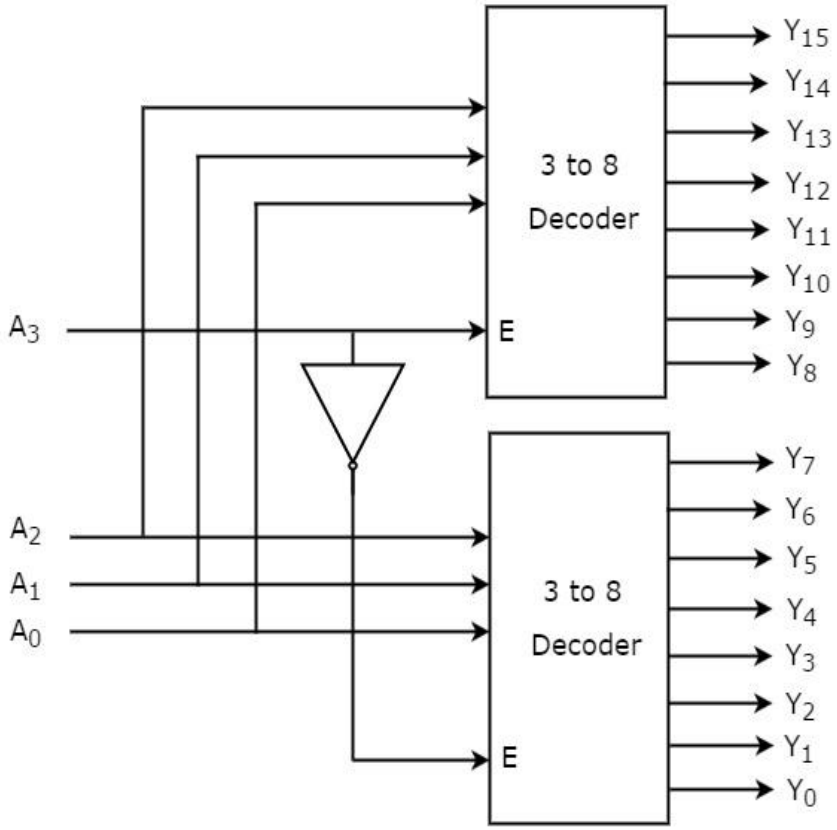
We know the following formula for finding the number of lower order decoders required.

Required number of lower order decoders = m_2 / m_1

Substitute, $m_1 = 8$ and $m_2 = 16$ in the above formula.

Required number of 3 *to* 8 *decoders* = 16 / 8 = 2

Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder. The **block diagram** of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure



The parallel inputs A_2, A_1 & A_0 are applied to each 3 to 8 decoder. The complement of input, A3 is connected to Enable, E of lower 3 to 8 decoder in order to get the outputs, Y_7 to Y_0. These are the **lower eight min terms**. The input, A_3 is directly connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, Y_{15} to Y_8. These are the **higher eight min terms**.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the primary function of a decoder in digital circuits?**

A. To store binary data
B. To convert analog signals to digital
C. To detect a specific input combination and activate one output
D. To perform arithmetic operations

Answer: C

Explanation: A decoder activates one output based on a specific combination of input signals.

2. **How many outputs does a 2-to-4 decoder have?**

A. 2
B. 4
C. 8

D. 16
Answer: B
Explanation: A 2-to-4 decoder has 4 outputs and 2 input lines.
3. **What condition must be true for a decoder to activate any output?**
A. All inputs must be zero
B. Enable signal must be high
C. All outputs must be high
D. Inputs must be complemented
Answer: B
Explanation: The decoder only activates outputs when the enable signal is high.
4. **What is the Boolean expression for output Y0 in a 2-to-4 decoder?**
A. E \cdot A1 \cdot A0
B. E \cdot A1' \cdot A0'
C. A1 \cdot A0
D. A1' \cdot A0'
Answer: B
Explanation: Y0 is active when both inputs are 0 and enable is 1.
5. **How many 2-to-4 decoders are needed to implement a 3-to-8 decoder?**
A. 1
B. 2
C. 3
D. 4
Answer: B
Explanation: Two 2-to-4 decoders are required to implement a 3-to-8 decoder.
6. **Which input is used to control the enable signal in the 3-to-8 decoder implementation?**
A. A0
B. A1
C. A2
D. E
Answer: C
Explanation: A2 is used to enable either the upper or lower 2-to-4 decoder.
7. **How many outputs does a 4-to-16 decoder have?**
A. 8
B. 12
C. 16
D. 32
Answer: C
Explanation: A 4-to-16 decoder has 16 outputs and 4 input lines.

8. **How many 3-to-8 decoders are needed to implement a 4-to-16 decoder?**
A. 1
B. 2
C. 4
D. 8
Answer: B
Explanation: Two 3-to-8 decoders are required to implement a 4-to-16 decoder.
9. **What do the outputs of a decoder represent when enabled?**
A. Max terms
B. Sum terms
C. Min terms
D. Logic gates
Answer: C
Explanation: Decoder outputs represent the min terms of the input variables.
10. **What happens to the outputs of a decoder when the enable signal is 0?**
A. All outputs are high
B. All outputs are undefined
C. All outputs are low
D. Outputs depend on inputs
Answer: C
Explanation: When enable is 0, all outputs of the decoder are 0.
Fill in the Blank Questions
1. A decoder with n input lines can have up to _____ output lines.
Answer: 2^n
2. The output of a decoder is active only when the _____ signal is high.
Answer: enable
3. The Boolean expression for output Y3 in a 2-to-4 decoder is _____.
Answer: E \cdot A1 \cdot A0
4. To implement a 3-to-8 decoder using 2-to-4 decoders, the input _____ is used to control the enable lines.
Answer: A2
5. The outputs of a decoder represent the _____ of the input variables.
Answer: min terms

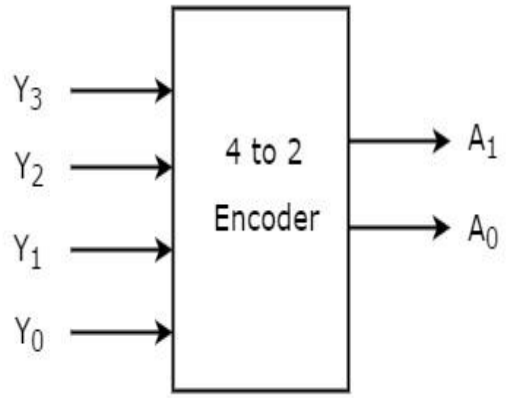
Unit 6: Encoders

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output

lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

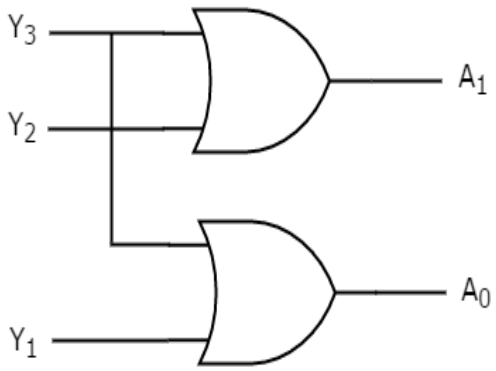
Inputs				**Outputs**	
Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$A_1 = Y_3 + Y_2$

$A_0 = Y_3 + Y_1$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.

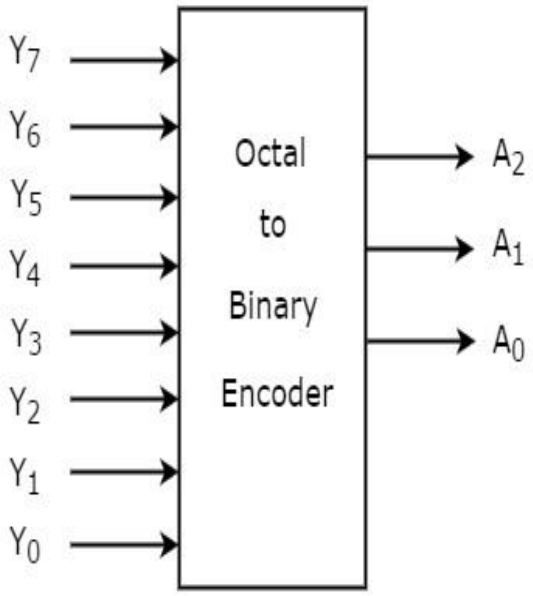


The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.

At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below



From Truth table, we can write the **Boolean functions** for each output as

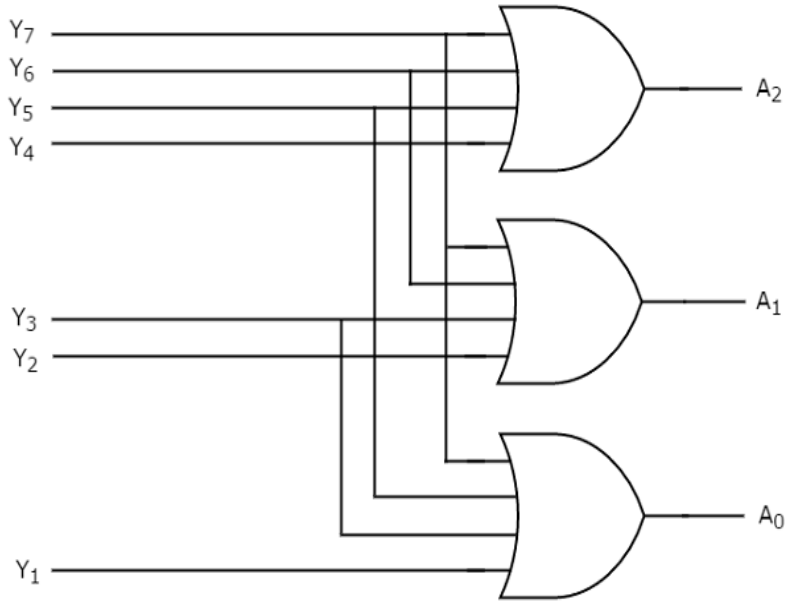
Inputs								**Outputs**		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$A_2 = Y_7 + Y_6 + Y_5 + Y_4$

$A_1 = Y_7 + Y_6 + Y_3 + Y_2$

$A_0 = Y_7 + Y_5 + Y_3 + Y_1$

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure



The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Drawbacks of Encoder

Following are the drawbacks of normal encoder.

- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For **example**, if both Y_3 and Y_6 are '1', then the encoder produces 111 at the output. This is neither

equivalent code corresponding to Y_3, when it is '1' nor the equivalent code corresponding to Y_6, when it is '1'.
So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the *binary* code corresponding to the active High inputs, which has higher priority. This encoder is called as **priority encoder**.

Priority Encoder
A 4 to 2 priority encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. Here, the input, Y_3 has the highest priority, whereas the input, Y_0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the *binary* code corresponding to the input, which is having **higher priority**.

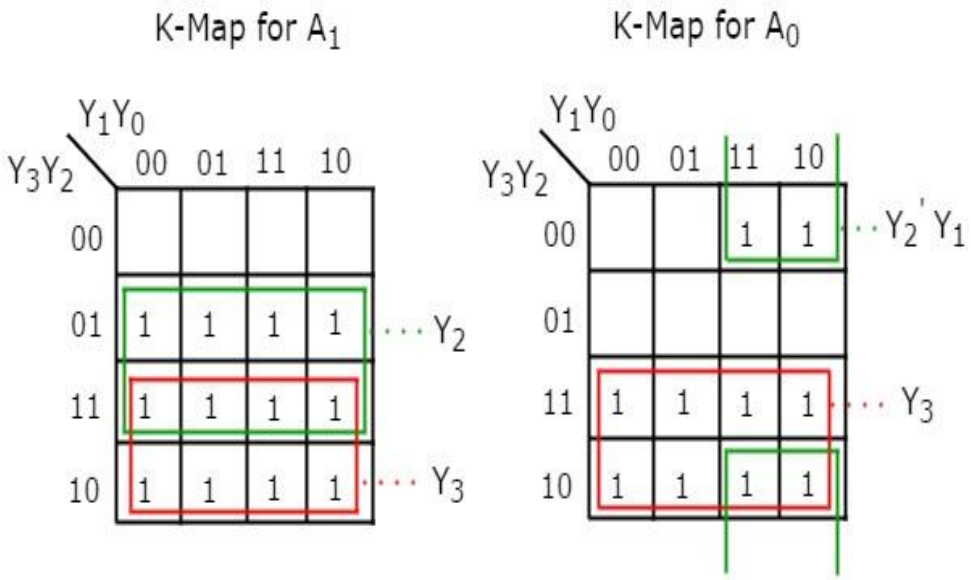
We considered one more **output, V** in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.
-

The **Truth table** of 4 to 2 priority encoder is shown below.

Inputs				**Outputs**		
Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**	**V**
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Use **4 variable K-maps** for getting simplified expressions for each output.



The simplified **Boolean functions** are

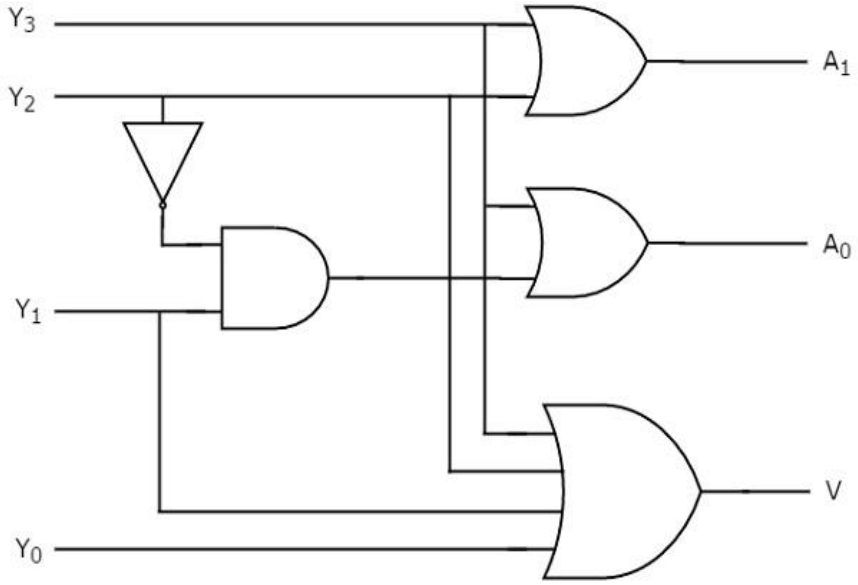
$A_1 = Y_3 + Y_2$

$A_0 = Y_3 + Y_2'Y_1$

Similarly, we will get the Boolean function of output, V as

$V = Y_3 + Y_2 + Y_1 + Y_0$

We can implement the above Boolean functions using logic gates. The **circuit diagram** of 4 to 2 priority encoder is shown in the following figure.



The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter. Here AND gate & inverter combination are used for producing a valid code at the outputs, even when

multiple inputs are equal to '1' at the same time. Hence, this circuit encodes the four inputs with two bits based on the **priority** assigned to each input.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the primary function of an encoder in digital circuits?**

A. To decode binary inputs
B. To convert binary to decimal
C. To generate a binary code from active input lines
D. To store binary data

Answer: C

Explanation: An encoder converts active High input signals into a binary code.

2. **How many outputs does a 4-to-2 encoder produce?**

A. 2
B. 4
C. 8
D. 1

Answer: A

Explanation: A 4-to-2 encoder has 4 inputs and produces 2 output bits.

3. **What is the Boolean expression for output A1 in a 4-to-2 encoder?**

A. Y3 + Y1
B. Y3 + Y2
C. Y2 + Y1
D. Y3 + Y0

Answer: B

Explanation: A1 = Y3 + Y2

4. **How many inputs does an octal-to-binary encoder have?**

A. 4
B. 8
C. 2
D. 16

Answer: B

Explanation: An octal-to-binary encoder has 8 inputs and 3 outputs.

5. **What is the Boolean expression for output A0 in an octal-to-binary encoder?**

A. Y7 + Y5 + Y3 + Y1
B. Y7 + Y6 + Y3 + Y2
C. Y7 + Y6 + Y5 + Y4
D. Y7 + Y5 + Y2 + Y0

Answer: A

Explanation: A0 = Y7 + Y5 + Y3 + Y1

6. **What is a major drawback of a normal encoder?**

A. It requires too many gates
B. It cannot handle decimal inputs
C. It produces ambiguous output when multiple inputs are active
D. It consumes high power
Answer: C
Explanation: Normal encoders can produce incorrect outputs if more than one input is active.

7. **What is the solution to the ambiguity in normal encoders?**
A. Use more gates
B. Use a decoder instead
C. Assign priorities to inputs
D. Increase the number of outputs
Answer: C
Explanation: Priority encoders resolve ambiguity by assigning priority to inputs.

8. **In a 4-to-2 priority encoder, which input has the highest priority?**
A. Y0
B. Y1
C. Y2
D. Y3
Answer: D
Explanation: Y3 has the highest priority.

9. **What does the output V represent in a priority encoder?**
A. Voltage level
B. Validity of the output code
C. Number of active inputs
D. Enable signal
Answer: B
Explanation: V indicates whether the output code is valid.

10. **What is the Boolean expression for output A0 in a priority encoder?**
A. Y3 + Y2
B. Y3 + Y2′Y1
C. Y2 + Y1
D. Y3 + Y1
Answer: B
Explanation: A0 = Y3 + Y2′Y1

Fill in the Blank Questions

1. An encoder converts active High inputs into a _____ code.
Answer: binary

2. A 4-to-2 encoder has _____ inputs and 2 outputs.
Answer: 4

3. The Boolean expression for output A2 in an octal-to-binary encoder is _____.
Answer: Y7 + Y6 + Y5 + Y4
4. A priority encoder resolves ambiguity by assigning _____ to inputs.
Answer: priority
5. The output V in a priority encoder indicates whether the output is _____.
Answer: valid

Unit 7: Latches

There are two types of memory elements based on the type of triggering that is suitable to operate it.

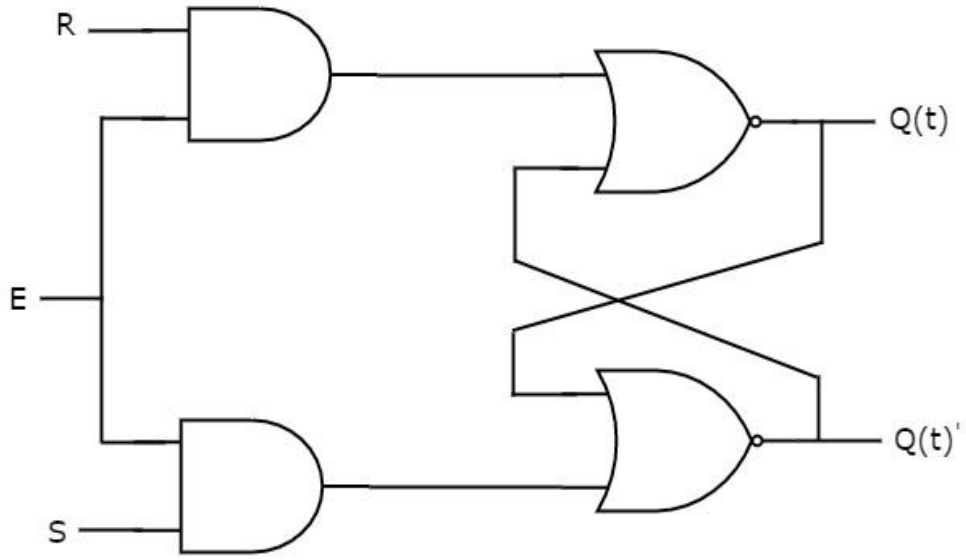
- Latches
- Flip-flops

Latches operate with enable signal, which is **level sensitive**. Whereas, flip-flops are edge sensitive. We will discuss about flip-flops in next lecture. Now, let us discuss about SR Latch & D Latch one by one.

SR Latch

SR Latch is also called as **Set Reset Latch**. This latch affects the outputs as long as the enable, E is maintained at '1'. The **circuit diagram** of SR Latch is shown

in the following figure.



This circuit has two inputs S & R and two outputs Qt & Qt'. The **upper NOR gate** has two inputs R & complement of present state, Qt' and produces next state, Qt + 1 when enable, E is '1'.

Similarly, the **lower NOR gate** has two inputs S & present state, Qt and produces complement of next state, Qt + 1 when enable, E is '1'.

We know that a **2-input NOR gate** produces an output, which is the complement of another input when one of the input is '0'. Similarly, it produces '0' output, when one of the input is '1'.

- If S = 1, then next state Qt + 1 will be equal to '1' irrespective of present state, Qt values.
- If R = 1, then next state Qt + 1 will be equal to '0' irrespective of present state, Qt values.

At any time, only of those two inputs should be '1'. If both inputs are '1', then the next state Qt + 1 value is undefined.

The following table shows the **state table** of SR latch.

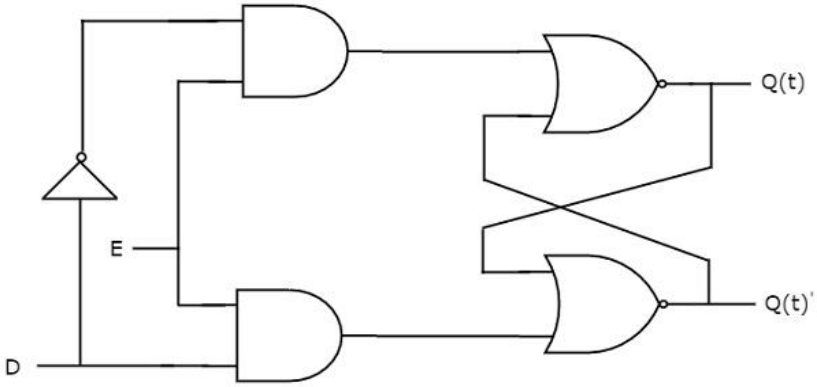
S	**R**	**Q**t + 1
0	0	Qt
0	1	0
1	0	1
1	1	-

Therefore, SR Latch performs three types of functions such as Hold, Set & Reset based on the input conditions.

D Latch

There is one drawback of SR Latch. That is the next state value can't be predicted when both the inputs S & R are one. So, we can overcome this difficulty by D Latch. It is also called as Data Latch. The **circuit diagram** of D Latch is shown in the following figure.

This circuit has single input D and two outputs Qt & Qt'. D Latch is obtained from SR Latch by placing an inverter between S amp;& R inputs and connect D input to S. That means we eliminated the combinations of



S & R are of same value.

- If D = 0 → S = 0 & R = 1, then next state Qt + 1 will be equal to '0' irrespective of present state, Qt values. This is corresponding to the second row of SR Latch state table.
- If D = 1 → S = 1 & R = 0, then next state Qt + 1 will be equal to '1' irrespective of present state, Qt values. This is corresponding to the third row of SR Latch state table.

The following table shows the **state table** of D latch.

D	**Q**t + 1
0	0
1	1

Therefore, D Latch Hold the information that is available on data input, D. That means the output of D Latch is sensitive to the changes in the input, D as long as the enable is High.

In this chapter, we implemented various Latches by providing the cross coupling between NOR gates. Similarly, you can implement these Latches using NAND gates.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What distinguishes latches from flip-flops in terms of triggering?**

A. Latches are edge-triggered
B. Flip-flops are level-sensitive
C. Latches are level-sensitive
D. Flip-flops use enable signals

Answer: C

Explanation: Latches respond to the level of the enable signal, while flip-flops respond to clock edges.

2. **What does SR in SR Latch stand for?**

A. Set and Reset
B. Store and Retrieve
C. Signal and Response
D. Start and Run

Answer: A

Explanation: SR stands for Set and Reset, the two control inputs of the latch.

3. **What happens when both S and R inputs are 1 in an SR Latch?**

A. Output is 1
B. Output is 0
C. Output holds previous state
D. Output is undefined

Answer: D

Explanation: When both inputs are 1, the output becomes unpredictable or undefined.

4. **What is the function of the enable signal in a latch?**

A. It resets the latch
B. It stores the output
C. It allows the latch to respond to inputs
D. It disables the circuit

Answer: C

Explanation: The latch responds to input changes only when the enable signal is high.

5. **Which logic gate is used in the basic SR Latch implementation?**

A. AND
B. OR
C. NOR
D. XOR

Answer: C

Explanation: SR Latches are commonly implemented using cross-coupled NOR gates.

6. **What is the main drawback of the SR Latch?**

A. It consumes too much power
B. It cannot store data
C. It has an undefined state when both inputs are 1
D. It requires a clock signal

Answer: C

Explanation: The SR Latch becomes unstable when both S and R are 1.

7. **How does the D Latch resolve the SR Latch's drawback?**

A. By using flip-flops
B. By removing the enable signal
C. By ensuring S and R are never both 1
D. By adding a clock input

Answer: C

Explanation: The D Latch uses an inverter to prevent S and R from being 1 simultaneously.

8. **What does the D input in a D Latch represent?**

A. Delay
B. Data
C. Drive
D. Direction

Answer: B

Explanation: D stands for Data, which is stored when the latch is enabled.

9. **What is the output of a D Latch when D = 1 and enable is high?**

A. 0
B. 1
C. Undefined
D. Previous state

Answer: B

Explanation: The output follows the input D when enabled.

10. **Which gates can also be used to implement latches besides NOR gates?**

A. XOR gates
B. NAND gates
C. AND gates
D. OR gates

Answer: B

Explanation: Latches can also be implemented using NAND gates.

Fill in the Blank Questions

1. Latches are _____ sensitive, while flip-flops are edge sensitive.

Answer: level

2. The SR Latch has two inputs: _____ and Reset.
Answer: Set
3. When both S and R are 1, the SR Latch output is _____.
Answer: undefined
4. The D Latch eliminates the undefined state by using an _____ between S and R inputs.
Answer: inverter
5. The output of a D Latch follows the input D when the _____ signal is high.
Answer: enable

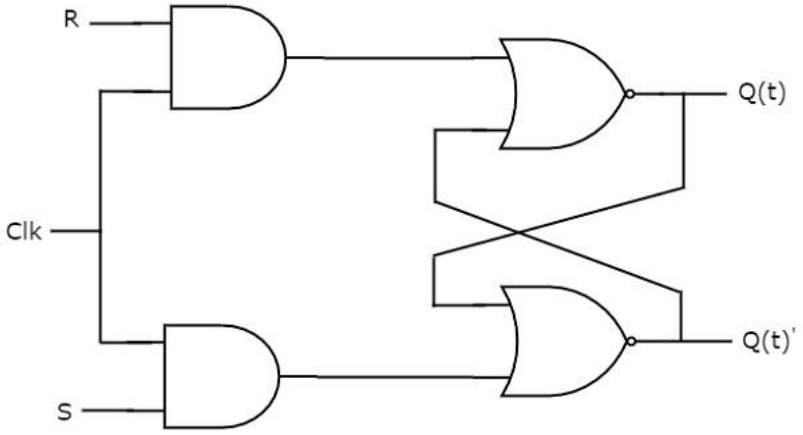
Unit 8 Flip-Flops

Previously, we discussed about Latches. Those are the basic building blocks of flip-flops. We can implement flip-flops in two methods.
In first method, **cascade two latches** in such a way that the first latch is enabled for every positive clock pulse and second latch is enabled for every negative clock pulse. So that the combination of these two latches become a flip-flop.
In second method, we can directly implement the flip-flop, which is edge sensitive. In this chapter, let us discuss the following **flip-flops** using second method.

- SR Flip-Flop
- D Flip-Flop
- JK Flip-Flop
- T Flip-Flop

SR Flip-Flop
SR flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal. The **circuit diagram** of SR flip-flop is shown in the following figure.

This circuit has two inputs S & R and two outputs Qt & Qt'. The operation



of SR flipflop is similar to SR Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.
The following table shows the **state table** of SR flip-flop.

S	**R**	**Q**t + 1
0	0	Qt
0	1	0
1	0	1
1	1	-

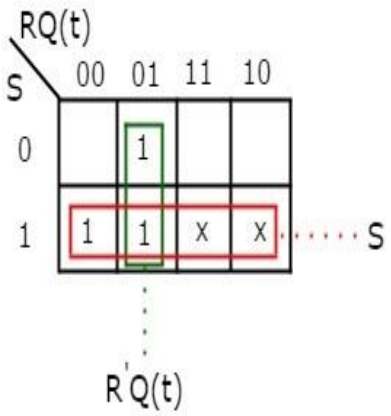
Here, Qt & Qt+1 are present state & next state respectively. So, SR flip-flop can be used for one of these three functions such as Hold, Reset &

Set based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of SR flip-flop.

Present Inputs		**Present State**	**Next State**
S	**R**	**Q**t	**Q**t + 1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

By using three variable K-Map, we can get the simplified expression for next state, Qt + 1. The **three variable K-Map** for next state, Qt + 1 is shown in the following figure.

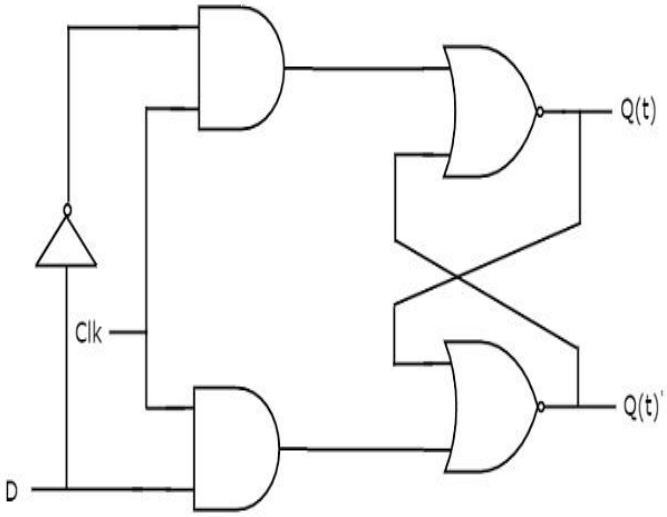
The maximum possible groupings of adjacent ones are already shown in



the figure. Therefore, the **simplified expression** for next state Qt + 1 is
Q (t + 1) = S + R′Q (t)

D Flip-Flop

D flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal. The **circuit diagram** of D flip-flop is shown in the following figure.



This circuit has single input D and two outputs Qt & Qt'. The operation of D flip-flop is similar to D Latch. But, this flip-flop affects the outputs only when positive transition of the clock signal is applied instead of active enable.

The following table shows the **state table** of D flip-flop.

D	**Qt + 1**
0	0
1	1

Therefore, D flip-flop always Hold the information, which is available on data input, D of earlier positive transition of clock signal. From the above state table, we can directly write the next state equation as

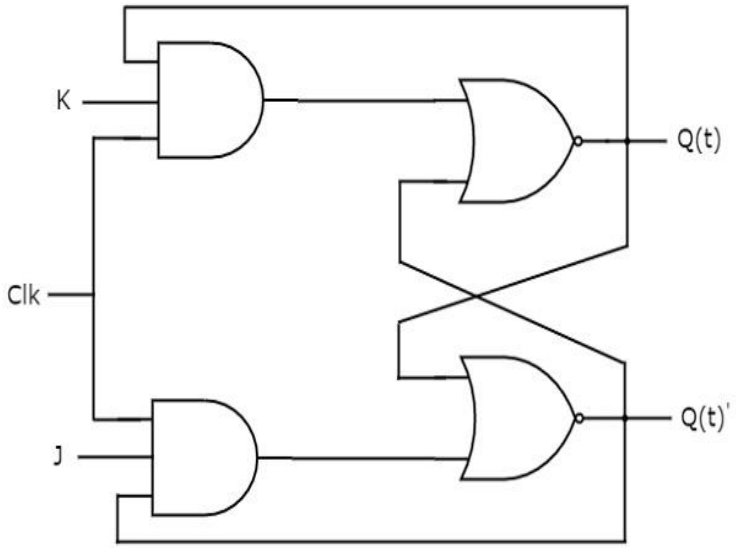
Qt + 1 = D

Next state of D flip-flop is always equal to data input, D for every positive transition of the clock signal. Hence, D flip-flops can be used in registers, **shift registers** and some of the counters.

JK Flip-Flop

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of JK flip-flop is shown in the following figure.

This circuit has two inputs J & K and two outputs Qt & Qt'. The operation of JK flip-flop is similar to SR flip-flop. Here, we considered the inputs of SR flip-flop as **S = J Qt'** and **R = KQ**t in order to utilize the modified SR flip-flop for 4 combinations of inputs.



The following table shows the **state table** of JK flip-flop.

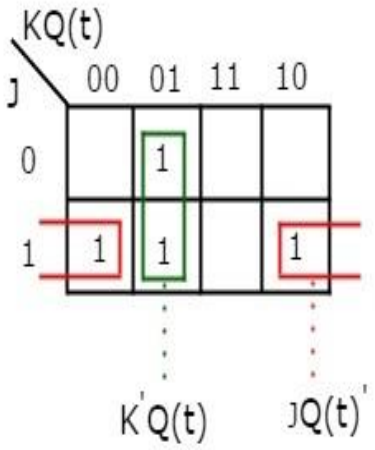
J	**K**	**Q**t + 1
0	0	Qt
0	1	0
1	0	1
1	1	Qt'

Here, Qt & Qt + 1 are present state & next state respectively. So, JK flip-flop can be used for one of these four functions such as Hold, Reset, Set & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of JK flip-flop.

Present Inputs		**Present State**	**Next State**
J	**K**	**Q**t	**Q**t+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

By using three variable K-Map, we can get the simplified expression for next state, Qt + 1. **Three variable K-Map** for next state, Qt + 1 is shown in the following figure.

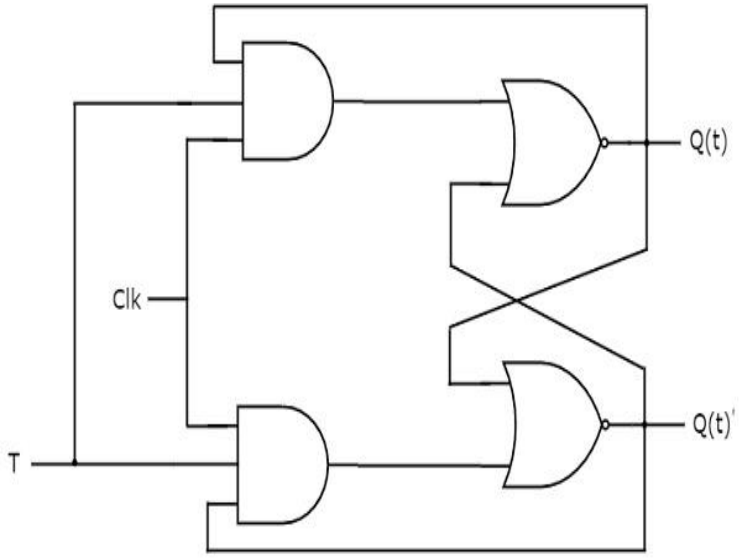
The maximum possible groupings of adjacent ones are already shown in



the figure. Therefore, the **simplified expression** for next state Qt + 1 is Q (t + 1) = JQ (t)′ + K′Q (t)

T Flip-Flop

T flip-flop is the simplified version of JK flip-flop. It is obtained by connecting the same input 'T' to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of T flip-flop is shown in the following figure



This circuit has single input T and two outputs Qt & Qt'. The operation of T flip-flop is same as that of JK flip-flop. Here, we considered the inputs of JK flip-flop as **J = T** and **K = T** in order to utilize the modified JK flip-flop for 2 combinations of inputs. So, we eliminated the other two

combinations of J & K, for which those two values are complement to each other in T flip-flop.
The following table shows the **state table** of T flip-flop.

D	**Q**t + 1
0	Qt
1	Qt’

Here, Qt & Qt + 1 are present state & next state respectively. So, T flip-flop can be used for one of these two functions such as Hold, & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the **characteristic table** of T flip-flop.

Inputs	**Present State**	**Next State**
T	**Q**t	**Q**t + 1
0	0	0
0	1	1
1	0	1
1	1	0

From the above characteristic table, we can directly write the **next state equation** as
$Q(t + 1) = T'Q(t) + TQ(t)'$
$\Rightarrow Q(t + 1) = T \oplus Q(t)$
The output of T flip-flop always toggles for every positive transition of the clock signal, when input T remains at logic High 11. Hence, T flip-flop can be used in **counters**.
In this chapter, we implemented various flip-flops by providing the cross coupling between NOR gates. Similarly, you can implement these flip-flops by using NAND gates.

Tutor Marked Assignment

1. Design a counter with sequences 0, 2, 3, 1, 0 using D-flip flops.
2. If both S and R inputs of an SR latch formed by cross-coupling two NOR gates are set to 0, the output is?
3. A MOD-16 ripple counter using J-K flip-flop has a current state 1001. What will the state be after 31 clock pulses?
4. A specific counter is using five S-R flip-flops. So what is the maximum number of states possible?
5. What is the maximum delay that can occur if four flip-flops are connected as a ripple counter and each flip-flop has propagation delays of t_{PHL} = 22 ns and t_{PLH} = 15 ns?

SELF-ASSSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What distinguishes flip-flops from latches?**

A. Flip-flops are level-sensitive
B. Flip-flops are edge-sensitive
C. Flip-flops use enable signals
D. Flip-flops operate without clock signals

E. **Answer:** B

Explanation: Flip-flops respond to clock edges, unlike latches which are level-sensitive.

2. **Which flip-flop is a modified version of the SR flip-flop?**

A. D flip-flop
B. T flip-flop
C. JK flip-flop
D. Master-slave flip-flop

Answer: C

Explanation: JK flip-flop is derived from the SR flip-flop with added logic to handle all input combinations.

3. **What is the next state equation for a D flip-flop?**

A. Qt + 1 = D
B. Qt + 1 = D′
C. Qt + 1 = Qt
D. Qt + 1 = D ⊕ Qt

E. **Answer:** A

Explanation: The next state of a D flip-flop is equal to the input D at the clock edge.

4. **What happens when both inputs of an SR flip-flop are 1?**

A. Output is 0
B. Output is 1
C. Output toggles
D. Output is undefined

Answer: D

Explanation: The SR flip-flop enters an undefined state when both S and R are 1.

5. **Which flip-flop toggles its output on every clock edge when input is high?**

A. SR flip-flop
B. D flip-flop
C. T flip-flop
D. JK flip-flop

Answer: C

Explanation: T flip-flop toggles its output when T = 1 and a clock edge occurs.

6. **What is the next state equation for a JK flip-flop?**

A. $Qt + 1 = J + K'$

B. $Qt + 1 = JQt' + K'Qt$

C. $Qt + 1 = J'Qt + KQt'$

D. $Qt + 1 = J \oplus K$

Answer: B

Explanation: This equation accounts for all input combinations and toggling behavior.

7. **Which flip-flop is derived by connecting the same input to both J and K of a JK flip-flop?**

A. SR flip-flop

B. D flip-flop

C. T flip-flop

D. Master-slave flip-flop

Answer: C

Explanation: T flip-flop is formed by tying J and K together.

8. **What is the next state equation for a T flip-flop?**

A. $Qt + 1 = T$

B. $Qt + 1 = T'Qt + TQt'$

C. $Qt + 1 = T \oplus Qt$

D. $Qt + 1 = Qt$

Answer: C

Explanation: The output toggles when T is high, represented by the XOR operation.

9. **Which flip-flop is best suited for use in counters?**

A. SR flip-flop

B. D flip-flop

C. T flip-flop

D. JK flip-flop

Answer: C

Explanation: T flip-flops toggle on each clock pulse, making them ideal for counting.

10. **What logic gates can be used to implement flip-flops besides NOR gates?**

A. XOR gates

B. NAND gates

C. AND gates

D. OR gates

Answer: B

Explanation: Flip-flops can also be constructed using NAND gates.

Fill in the Blank Questions

1. Flip-flops are _____ sensitive, responding to clock transitions.

Answer: edge

2. The SR flip-flop enters an _____ state when both S and R are 1.

Answer: undefined

3. The next state of a D flip-flop is equal to the input _____.

Answer: D

4. The T flip-flop toggles its output when input T is _____.

Answer: high

5. The next state equation for a T flip-flop is Q(t + 1) = _____.

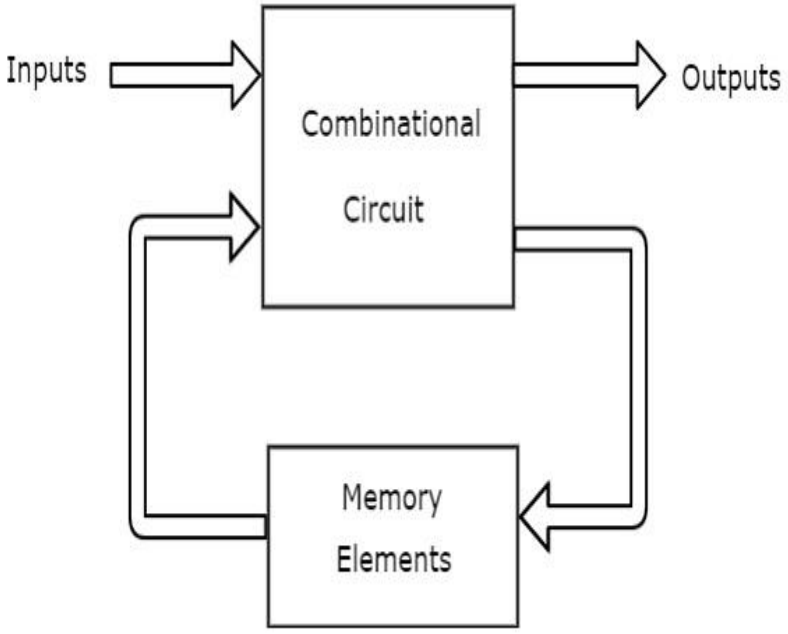
Answer: T $\oplus$ Q(t)

Module 4 Sequential Circuits

Unit 1 Sequential Circuits
Unit 2 Conversion of Flip-Flops
Unit 3 Registers
Unit 4 Counters

Unit 1: Sequential Circuits

We discussed various combinational circuits in earlier chapters. All these circuits have a set of outputs, which depends only on the combination of present inputs. The following figure shows the **block diagram** of sequential circuit.



This sequential circuit contains a set of inputs and outputs. The outputs of sequential circuit depends not only on the combination of present inputs but also on the previous outputs. Previous output is nothing but the **present state**. Therefore, sequential circuits contain combinational circuits along with memory storage elements. Some sequential circuits may not contain combinational circuits, but only memory elements.
Following table shows the **differences** between combinational circuits and sequential circuits.

Combinational Circuits	Sequential Circuits
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What distinguishes sequential circuits from combinational circuits?**

A. They use only logic gates
B. Their outputs depend only on current inputs
C. Their outputs depend on current inputs and previous states
D. They do not require memory

Answer: C

Explanation: Sequential circuits consider both present inputs and stored previous outputs.

2. **Which of the following is NOT a feature of combinational circuits?**

A. No feedback path
B. No memory elements
C. Clock signal required
D. Outputs depend only on present inputs

Answer: C

Explanation: Combinational circuits do not require a clock signal.

3. **What is the role of memory elements in sequential circuits?**

A. To store binary numbers
B. To hold previous output states
C. To convert analog signals
D. To generate clock pulses

Answer: B

Explanation: Memory elements store the present state, which influences future outputs.

4. **Which type of circuit includes a feedback path?**

A. Combinational circuit
B. Arithmetic circuit
C. Sequential circuit
D. Decoder circuit

Answer: C

Explanation: Feedback paths are a defining feature of sequential circuits.

5. **Which of the following is required for the operation of sequential circuits?**
A. Voltage regulator
B. Clock signal
C. Amplifier
D. Comparator
Answer: B
Explanation: Sequential circuits rely on clock signals to synchronize state changes.
6. **Which circuit type is generally easier to design?**
A. Sequential
B. Combinational
C. Analog
D. Hybrid
Answer: B
Explanation: Combinational circuits are simpler because they don't involve memory or timing.
7. **What does the present state in a sequential circuit refer to?**
A. The current input
B. The output of the last clock cycle
C. The voltage level
D. The logic gate used
Answer: B
Explanation: Present state is the stored output from the previous cycle.
8. **Which of the following is NOT typically found in a sequential circuit?**
A. Memory elements
B. Feedback paths
C. Clock signal
D. Only logic gates
Answer: D
Explanation: Sequential circuits include more than just logic gates—they also have memory and timing components.
9. **Which statement is true about combinational circuits?**
A. They require a clock signal
B. They store previous outputs
C. They are difficult to design
D. Their outputs depend only on current inputs
Answer: D
Explanation: Combinational circuits produce outputs based solely on current inputs.
10. **What type of circuit is used when output must reflect both current input and history?**
A. Decoder
B. Combinational
C. Sequential

D. Multiplexer

Answer: C

Explanation: Sequential circuits incorporate memory to reflect history in their outputs.

Fill in the Blank Questions

1. Sequential circuits require _____ elements to store previous outputs.

Answer: memory

2. Combinational circuits do not include a _____ path.

Answer: feedback

3. The output of a sequential circuit depends on present inputs and _____ state.

Answer: present

4. Sequential circuits are _____ to design compared to combinational circuits.

Answer: difficult

5. A _____ signal is necessary for the operation of sequential circuits.

Answer: clock

Unit 2 Conversion of Flip-Flops

In previous lectures, we discussed the four flip-flops, namely SR flip-flop, D flip-flop, JK flip-flop & T flip-flop. We can convert one flip-flop into the remaining three flip-flops by including some additional logic. So, there will be total of twelve **flip-flop conversions**.

Follow these **steps** for converting one flip-flop to the other.

- Consider the **characteristic table** of desired flip-flop.
- Fill the excitation values inputs of given flip-flop for each combination of present state and next state. The **excitation table** for all flip-flops is shown below.
- Get the **simplified expressions** for each excitation input. If necessary, use Kmaps for simplifying.

Present State	**Next State**	**SR flip-flop inputs**		**D flip-flop input**	**JK flip-flop inputs**		**T flip-flop input**
Qt	**Q**t + 1	**S**	**R**	**D**	**J**	**K**	**T**
0	0	0	x	0	0	x	0
0	1	1	0	1	1	x	1
1	0	0	1	0	x	1	1
1	1	x	0	1	x	0	0

- Draw the **circuit diagram** of desired flip-flop according to the simplified expressions using given flip-flop and necessary logic gates.

Now, let us convert few flip-flops into other. Follow the same process for remaining flipflop conversions.

SR Flip-Flop to other Flip-Flop Conversions

Following are the three possible conversions of SR flip-flop to other flip-flops.

- SR flip-flop to D flip-flop
- SR flip-flop to JK flip-flop
- SR flip-flop to T flip-flop

SR flip-flop to D flip-flop conversion

Here, the given flip-flop is SR flip-flop and the desired flip-flop is D flip-flop. Therefore, consider the following **characteristic table** of D flip-flop.

D flip-flop input	**Present State**	**Next State**
D	**Q**t	**Q**t + 1
0	0	0
0	1	0
1	0	1
1	1	1

D flip-flop input	Present State	Next State	SR flip-flop inputs	
D	**Q**t	**Q**t + 1	**S**	**R**
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0

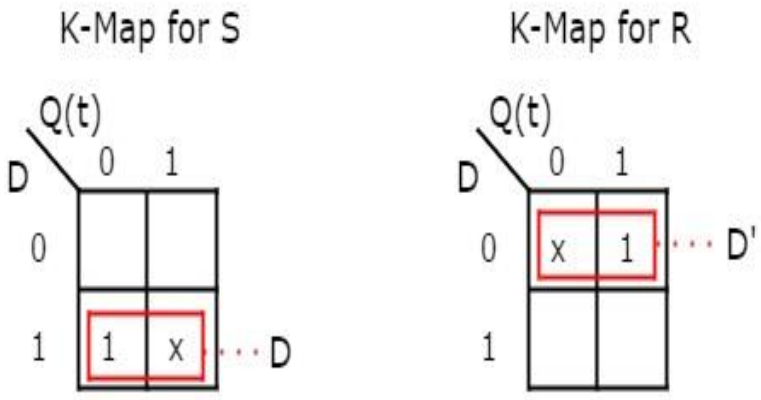
We know that SR flip-flop has two inputs S & R. So, write down the excitation values of SR flip-flop for each combination of present state and next state values. The following table shows the characteristic table of D flip-flop along with the **excitation inputs** of SR flip-flop.

From the above table, we can write the **Boolean functions** for each input as below.

$S = m_2 + d_3$

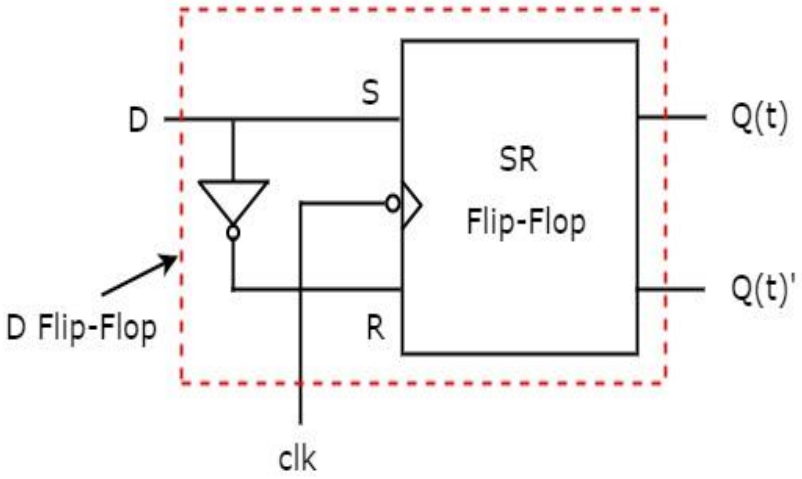
$R = m_1 + d_0$

We can use 2 variable K-Maps for getting simplified expressions for these inputs. The **k-Maps** for S & R are shown below.



So, we got S = D & R = D' after simplifying. The **circuit diagram** of D flip-flop is shown in the following figure.

This circuit consists of SR flip-flop and an inverter. This inverter produces an output, which is complement of input, D. So, the overall circuit has single input, D and two outputs Qt & Qt'. Hence, it is a **D flip-flop**. Similarly, you can do other two conversions.



D Flip-Flop to other Flip-Flop Conversions

Following are the three possible conversions of D flip-flop to other flip-flops.

- D flip-flop to T flip-flop
- D flip-flop to SR flip-flop
- D flip-flop to JK flip-flop

D flip-flop to T flip-flop conversion

Here, the given flip-flop is D flip-flop and the desired flip-flop is T flip-flop. Therefore, consider the following **characteristic table** of T flip-flop.

T flip-flop input	**Present State**	**Next State**
T	**Q**t	**Q**t + 1
0	0	0
0	1	1
1	0	1
1	1	0

T flip-flop input	**Present State**	**Next State**	**D flip-flop input**
T	**Q**t	**Q**t + 1	**D**

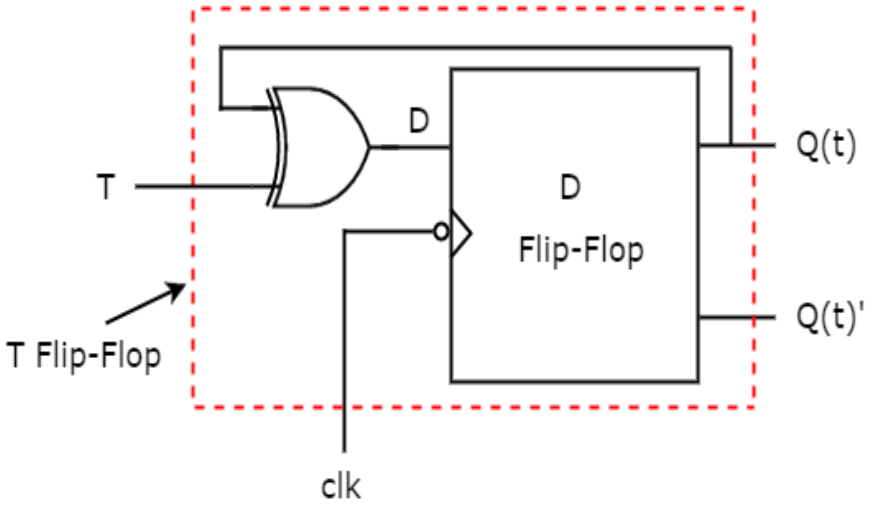
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

We know that D flip-flop has single input D. So, write down the excitation values of D flip-flop for each combination of present state and next state values. The following table shows the characteristic table of T flip-flop along with the **excitation input** of D flip-flop.

From the above table, we can directly write the **Boolean function** of D as below.

$D = T \oplus Q(t)$

So, we require a two input Exclusive-OR gate along with D flip-flop. The **circuit diagram** of T flip-flop is shown in the following figure.



This circuit consists of D flip-flop and an Exclusive-OR gate. This Exclusive-OR gate produces an output, which is Ex-OR of T and Qt. So, the overall circuit has single input, T and two outputs Qt & Qt'. Hence, it is a **T flip-flop**. Similarly, you can do other two conversions.

JK Flip-Flop to other Flip-Flop Conversions

Following are the three possible conversions of JK flip-flop to other flip-flops.

- JK flip-flop to T flip-flop
- JK flip-flop to D flip-flop
- JK flip-flop to SR flip-flop

JK flip-flop to T flip-flop conversion

Here, the given flip-flop is JK flip-flop and the desired flip-flop is T flip-flop. Therefore, consider the following **characteristic table** of T flip-flop.

T flip-flop input	**Present State**	**Next State**
T	**Q**t	**Q**t + 1
0	0	0
0	1	1
1	0	1
1	1	0

We know that JK flip-flop has two inputs J & K. So, write down the excitation values of JK flip-flop for each combination of present state and next state values. The following table shows the characteristic table of T

flip-flop along with the **excitation inputs** of JK flipflop.

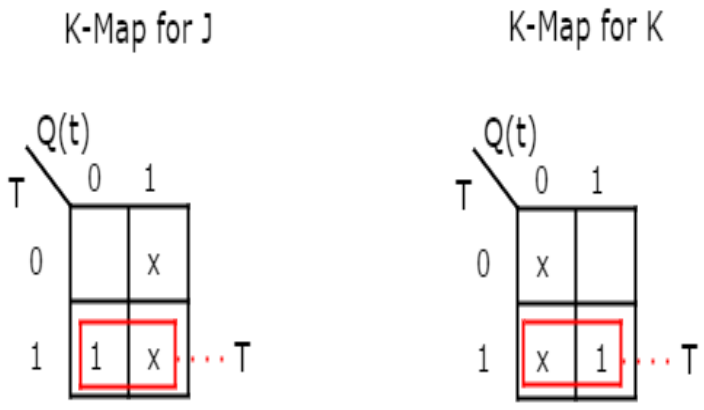
T flip-flop input	**Present State**	**Next State**	**JK flip-flop inputs**	
T	**Q**t	**Q**t + 1	**J**	**K**
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

From the above table, we can write the **Boolean functions** for each input as below.

$J = m_2 + d_1 + d_3$

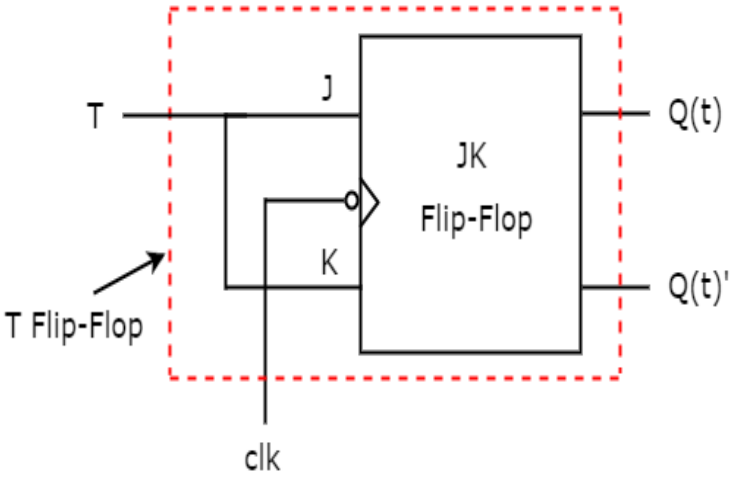
$K = m_3 + d_0 + d_2$

We can use 2 variable K-Maps for getting simplified expressions for these two inputs. The **k-Maps** for J & K are shown below.



So, we got, J = T & K = T after simplifying. The **circuit diagram** of T flip-flop is shown in the following figure.

This circuit consists of JK flip-flop only. It doesn't require any other



gates. Just connect the same input T to both J & K. So, the overall circuit has single input, T and two outputs Qt & Qt'. Hence, it is a **T flip-flop**. Similarly, you can do other two conversions.

T Flip-Flop to other Flip-Flop Conversions

Following are the three possible conversions of T flip-flop to other flip-flops.

- T flip-flop to D flip-flop
- T flip-flop to SR flip-flop
- T flip-flop to JK flip-flop

T flip-flop to D flip-flop conversion

Here, the given flip-flop is T flip-flop and the desired flip-flop is D flip-flop. Therefore, consider the characteristic table of D flip-flop and write down the excitation values of T flip-flop for each combination of present state and next state values. The following table shows the **characteristic table** of D flip-flop

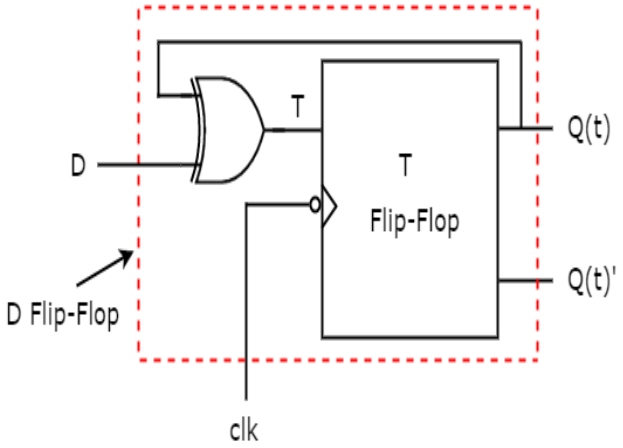
along with the **excitation input** of T flip-flop.

D flip-flop input	**Present State**	**Next State**	**T flip-flop input**	
D	**Q**t	**Q**t + 1	**T**	
0	0	0	0	
0	1	0	1	
1	0	1	1	
1	1	1	0	

From the above table, we can directly write the Boolean function of T as below.

$T = D \oplus Q(t)$

So, we require a two input Exclusive-OR gate along with T flip-flop. The **circuit diagram** of D flip-flop is shown in the following figure.



This circuit consists of T flip-flop and an Exclusive-OR gate. This Exclusive-OR gate produces an output, which is Ex-OR of D and Qt. So, the overall circuit has single input, D and two outputs Qt & Qt'. Hence, it is a **D flip-flop**. Similarly, you can do other two conversions.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions

1. What distinguishes sequential logic circuits from combinational ones?

A. They use only AND gates
B. Their output depends on current inputs and past states ✅
C. They do not use memory
D. They operate randomly

2. Which of the following is a basic sequential circuit?

A. Multiplexer
B. Decoder
C. Flip-flop ✅
D. Comparator

3. What is the main function of a flip-flop?

A. To perform arithmetic
B. To store a single bit of data ✅
C. To decode signals
D. To compare inputs

4. Which flip-flop has a toggle feature?

A. SR
B. JK ✅
C. D
D. T

5. What does the clock signal do in sequential circuits?

A. It powers the circuit

B. It synchronizes changes in state ✅
C. It stores data
D. It resets the system
6. Which flip-flop is commonly used for data storage?
A. T
B. D ✅
C. JK
D. SR
7. What is the output of a T flip-flop when the input is 1?
A. No change
B. Toggle ✅
C. Reset
D. Set
8. Which of the following is a type of sequential circuit?
A. Full adder
B. Counter ✅
C. Multiplexer
D. Decoder
9. What is the function of a register in sequential logic?
A. To perform logic operations
B. To store multiple bits of data ✅
C. To decode signals
D. To compare inputs
10. Which flip-flop is known for its simplicity and direct data input?
A. JK
B. D ✅
C. T
D. SR

Fill in the Blank Questions

1. Sequential circuits depend on current inputs and _____ states. → *past*
2. A _____ stores a single bit of data. → *flip-flop*
3. The _____ signal synchronizes changes in sequential circuits. → *clock*
4. A _____ is used to store multiple bits of data. → *register*
5. The _____ flip-flop toggles its output when the input is 1. → *T*

Unit 3 Registers

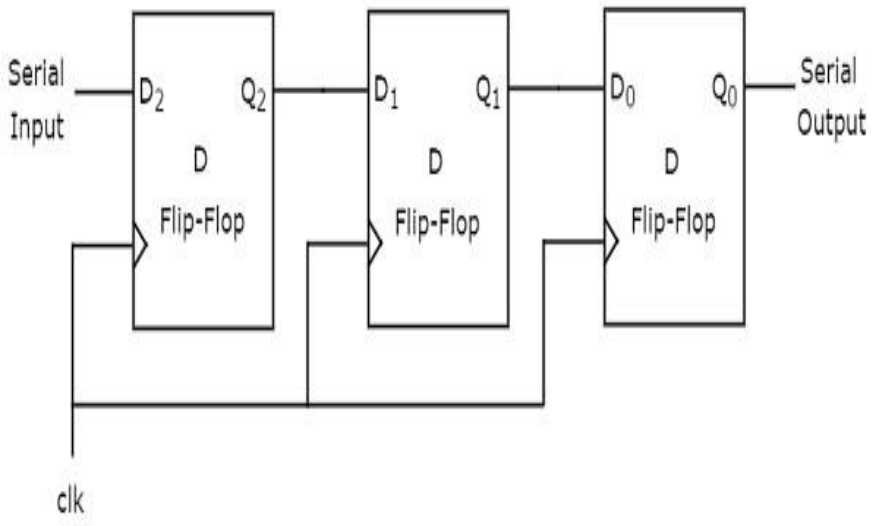
Shift Register

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

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

- Serial In − Serial Out shift register
- Serial In − Parallel Out shift register
- Parallel In − Serial Out shift register
- Parallel In − Parallel Out shift register

Serial In − Serial Out SISO Shift Register
The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out SISO shift register. The **block diagram** of 3-bit SISO shift register is shown in the following figure.



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

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

the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

Example

Let us see the working of 3-bit SISO shift register by sending the binary information **"011"** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is $Q_2Q_1Q_0$ = 000. We can understand the **working of 3-bit SISO shift register** from the following table.

No of positive edge of Clock	**Serial Input**	**Q_2**	**Q_1**	**Q_0**
0	-	0	0	0
1	1LSB	1	0	0
2	1	1	1	0
3	0MSB	0	1	1LSB
4	-	-	0	1
5	-	-	-	0MSB

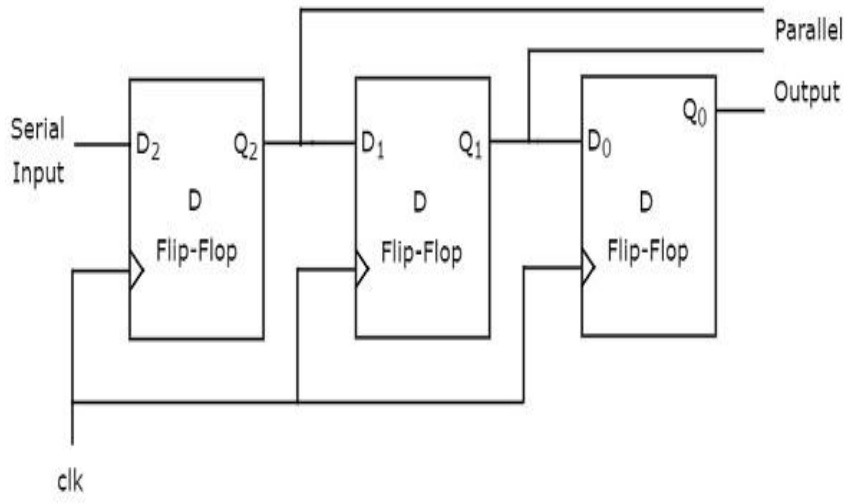
The initial status of the D flip-flops in the absence of clock signal is $Q_2Q_1Q_0$ = 000. Here, the serial output is coming from Q_0. So, the LSB 1 is received at 3rd positive edge of clock and the MSB 0 is received at 5th positive edge of clock.

Therefore, the 3-bit SISO shift register requires five clock pulses in order to produce the valid output. Similarly, the **N-bit SISO shift register** requires **2N-1** clock pulses in order to shift 'N' bit information.

Serial In - Parallel Out SIPO Shift Register

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out SIPO shift register. The **block diagram** of 3-bit SIPO shift register is shown in the following figure.

This circuit consists of three D flip-flops, which are cascaded. That



means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.
In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get **parallel outputs** from this shift register.

Example

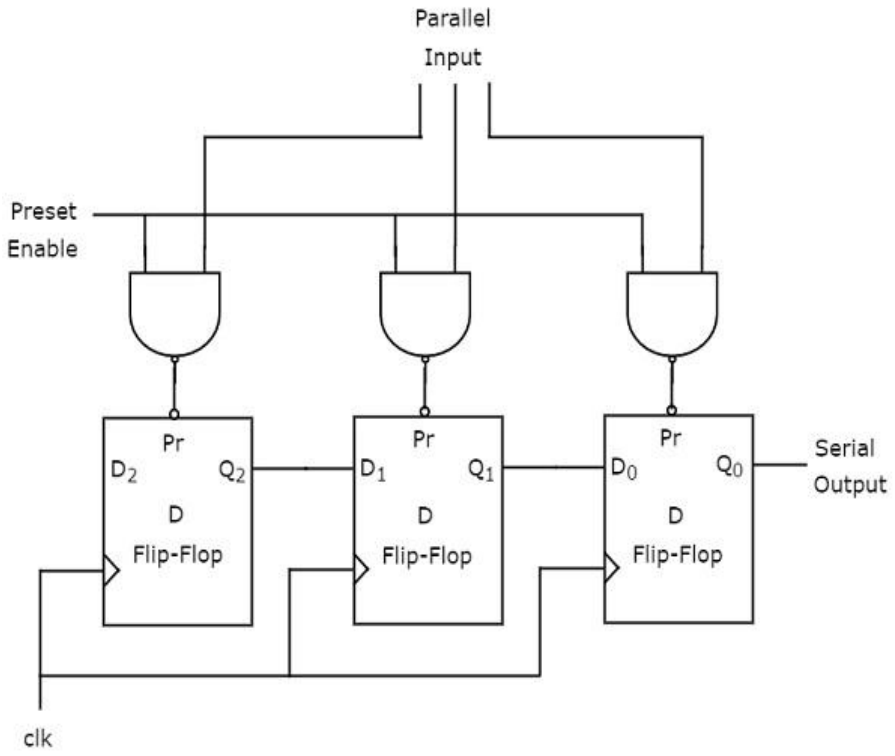
Let us see the working of 3-bit SIPO shift register by sending the binary information **"011"** from LSB to MSB serially at the input.
Assume, initial status of the D flip-flops from leftmost to rightmost is $Q_2Q_1Q_0 = 000$. Here, Q_2 & Q_0 are MSB & LSB respectively. We can understand the **working of 3-bit SIPO shift register** from the following table.

No of positive edge of Clock	**Serial Input**	**Q_2**MSB	**Q_1**	**Q_0**LSB
0	-	0	0	0
1	1LSB	1	0	0
2	1	1	1	0
3	0MSB	0	1	1

The initial status of the D flip-flops in the absence of clock signal is $Q_2Q_1Q_0 = 000$. The binary information **"011"** is obtained in parallel at the outputs of D flip-flops for third positive edge of clock.
So, the 3-bit SIPO shift register requires three clock pulses in order to produce the valid output. Similarly, the **N-bit SIPO shift register** requires **N** clock pulses in order to shift 'N' bit information.

Parallel In − Serial Out PISO Shift Register
The shift register, which allows parallel input and produces serial output is known as Parallel In − Serial Out PISO shift register. The **block diagram** of 3-bit PISO shift register is shown in the following figure.

This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-



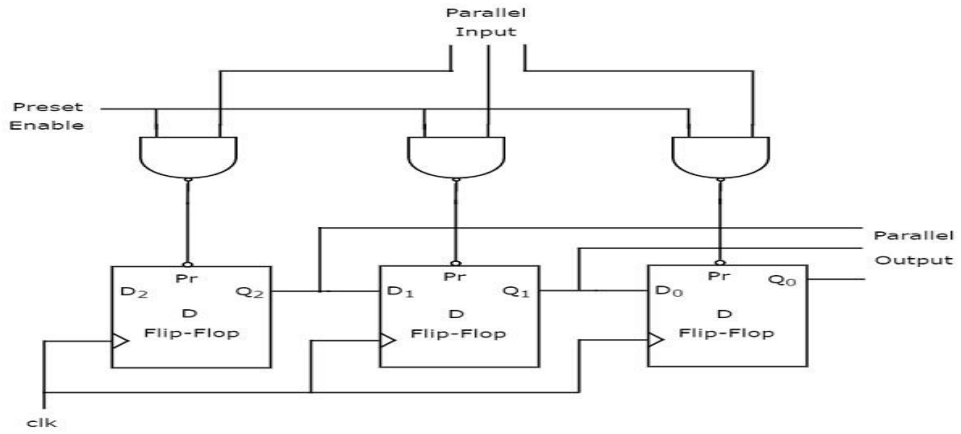
flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the **serial output** from the right most D flip-flop.

Example

Let us see the working of 3-bit PISO shift register by applying the binary information **"011"** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be $Q_2Q_1Q_0$ = 011. We can understand the **working of 3-bit PISO shift register** from the following table.



No of positive edge of Clock	Q_2	Q_1	Q_0
0	0	1	1LSB
1	-	0	1
2	-	-	0LSB

Here, the serial output is coming from Q0. So, the LSB 11 is received before applying positive edge of clock and the MSB 00 is received at 2^{nd} positive edge of clock.

Therefore, the 3-bit PISO shift register requires two clock pulses in order to produce the valid output. Similarly, the **N-bit PISO shift register** requires **N-1** clock pulses in order to shift 'N' bit information.

Parallel In - Parallel Out PIPO Shift Register

The shift register, which allows parallel input and produces parallel output is known as Parallel In − Parallel Out PIPO shift register. The **block diagram** of 3-bit PIPO shift register is shown in the following figure.
This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. We can apply the parallel inputs through

preset or clear. These two are asynchronous inputs. That means, the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition. So, we will get the **parallel outputs** from each D flip-flop.

Example

Let us see the working of 3-bit PIPO shift register by applying the binary information **"011"** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be $Q_2Q_1Q_0$ = 011. So, the binary information **"011"** is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.

Therefore, the 3-bit PIPO shift register requires zero clock pulses in order to produce the valid output. Similarly, the **N-bit PIPO shift register** doesn't require any clock pulse in order to shift 'N' bit information.

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What is the primary function of a register in digital circuits?**
A. To perform arithmetic operations
B. To store multiple bits of binary data
C. To decode binary inputs
D. To generate clock signals
Answer: B
Explanation: Registers are groups of flip-flops used to store binary data.
2. **Which type of shift register allows serial input and serial output?**
A. SIPO
B. PISO
C. SISO
D. PIPO
Answer: C
Explanation: SISO stands for Serial In – Serial Out.
3. **How many clock pulses are required for a 3-bit SISO shift register to produce valid output?**
A. 3
B. 4
C. 5

D. 6
Answer: C
Explanation: A 3-bit SISO shift register requires 2N–1 = 5 clock pulses.
4. **Which shift register allows serial input and parallel output?**
A. SISO
B. SIPO
C. PISO
D. PIPO
Answer: B
Explanation: SIPO stands for Serial In – Parallel Out.
5. **How many clock pulses are needed for a 3-bit SIPO shift register to produce valid output?**
A. 2
B. 3
C. 4
D. 5
Answer: B
Explanation: An N-bit SIPO shift register requires N clock pulses.
6. **Which shift register allows parallel input and serial output?**
A. SISO
B. SIPO
C. PISO
D. PIPO
Answer: C
Explanation: PISO stands for Parallel In – Serial Out.
7. **How many clock pulses are needed for a 3-bit PISO shift register to produce valid output?**
A. 2
B. 3
C. 4
D. 5
Answer: A
Explanation: An N-bit PISO shift register requires N–1 clock pulses.
8. **Which shift register allows parallel input and parallel output?**
A. SISO
B. SIPO
C. PISO
D. PIPO
Answer: D
Explanation: PIPO stands for Parallel In – Parallel Out.
9. **What kind of inputs are used in PIPO shift registers to load data?**
A. Serial inputs
B. Clock inputs
C. Asynchronous inputs

D. Enable inputs
Answer: C
Explanation: PIPO registers use asynchronous inputs like preset or clear.
10. **How many clock pulses are needed for a 3-bit PIPO shift register to produce valid output?**
A. 0
B. 1
C. 2
D. 3
Answer: A
Explanation: PIPO registers produce output immediately without clock pulses.

Fill in the Blank Questions

1. A shift register uses _____ flip-flops to store multiple bits.
Answer: D
2. The SISO shift register requires _____ clock pulses to shift N bits.
Answer: 2N–1
3. The SIPO shift register produces _____ output.
Answer: parallel
4. The PISO shift register uses _____ inputs to load data.
Answer: parallel
5. The PIPO shift register uses _____ inputs to load and output data without clock pulses.
Answer: asynchronous

Unit 4 Counters

In previous lectures, we discussed various shift registers & **counters using D flipflops**. Now, let us discuss various counters using T flip-flops. We know that T flip-flop toggles the output either for every positive edge of clock signal or for negative edge of clock signal.

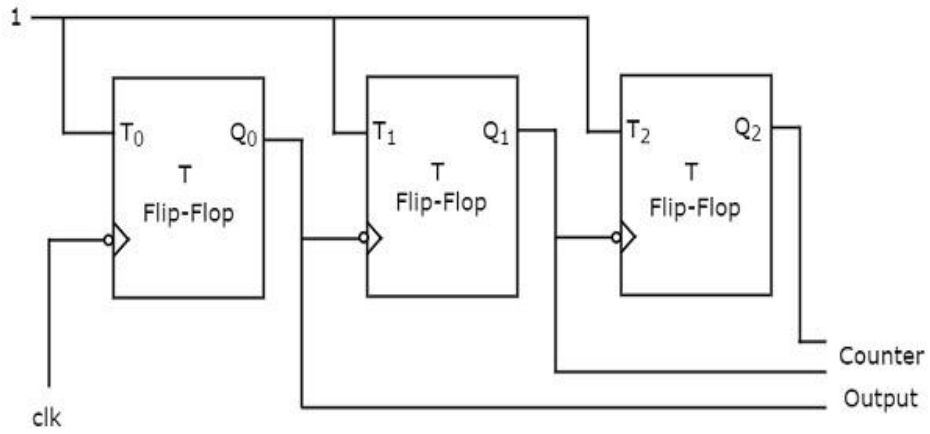
An 'N' bit binary counter consists of 'N' T flip-flops. If the counter counts from 0 to $2^N - 1$, then it is called as binary **up counter**. Similarly, if the counter counts down from $2^N - 1$ to 0, then it is called as binary **down counter**.
There are two **types of counters** based on the flip-flops that are connected in synchronous or not.

- Asynchronous counters
- Synchronous counters

Asynchronous Counters
If the flip-flops do not receive the same clock signal, then that counter is called as **Asynchronous counter**. The output of system clock is applied as clock signal only to first flip-flop. The remaining flip-flops receive the clock signal from output of its previous stage flip-flop. Hence, the outputs of all flip-flops do not change affect at the same time.
Now, let us discuss the following two counters one by one.

- Asynchronous Binary up counter
- Asynchronous Binary down counter

Asynchronous Binary Up Counter
An 'N' bit Asynchronous binary up counter consists of 'N' T flip-flops. It counts from 0 to $2^N - 1$. The **block diagram** of 3-bit Asynchronous binary up counter is shown in the following figure.
The 3-bit Asynchronous binary up counter contains three T flip-flops and



the T-input of all the flip-flops are connected to '1'. All these flip-flops are negative edge triggered but the outputs change asynchronously. The

clock signal is directly applied to the first T flip-flop. So, the output of first T flip-flop **toggles** for every negative edge of clock signal.

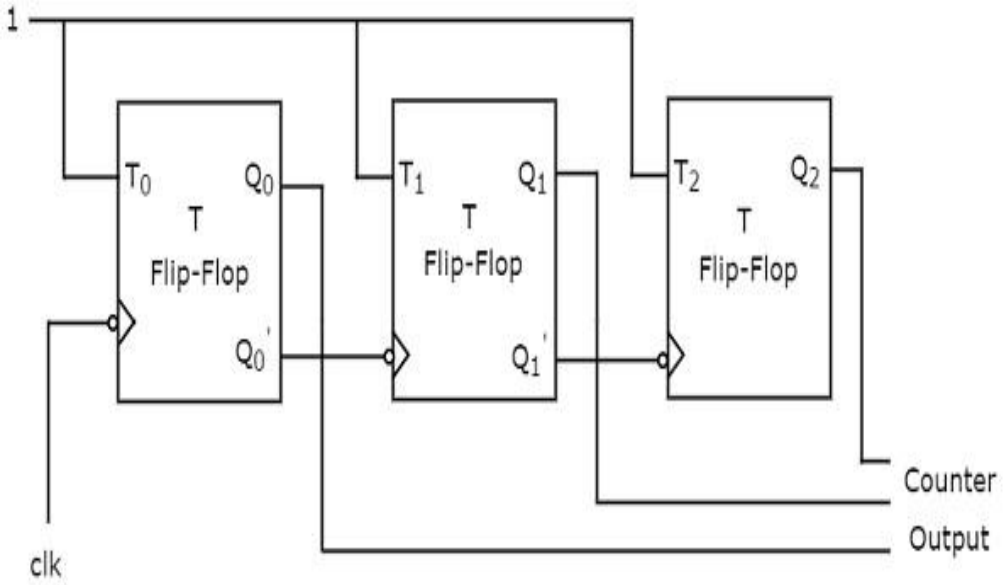
The output of first T flip-flop is applied as clock signal for second T flip-flop. So, the output of second T flip-flop toggles for every negative edge of output of first T flip-flop. Similarly, the output of third T flip-flop toggles for every negative edge of output of second T flip-flop, since the output of second T flip-flop acts as the clock signal for third T flip-flop.
Assume the initial status of T flip-flops from rightmost to leftmost is $Q_2Q_1Q_0 = 000$. Here, Q_2 & Q_0 are MSB & LSB respectively. We can understand the **working** of 3-bit asynchronous binary counter from the following table.

No of negative edge of Clock	**Q_0**LSB	**Q_1**	**Q_2**MSB
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Here Q_0 toggled for every negative edge of clock signal. Q_1 toggled for every Q_0 that goes from 1 to 0, otherwise remained in the previous state. Similarly, Q_2 toggled for every Q_1 that goes from 1 to 0, otherwise remained in the previous state.
The initial status of the T flip-flops in the absence of clock signal is $Q_2Q_1Q_0 = 000$. This is incremented by one for every negative edge of clock signal and reached to maximum value at 7^{th} negative edge of clock signal. This pattern repeats when further negative edges of clock signal are applied.

Asynchronous Binary Down Counter
An 'N' bit Asynchronous binary down counter consists of 'N' T flip-flops. It counts from $2^N - 1$ to 0. The **block diagram** of 3-bit Asynchronous binary down counter is shown in the following figure.

The block diagram of 3-bit Asynchronous binary down counter is similar



to the block diagram of 3-bit Asynchronous binary up counter. But, the only difference is that instead of connecting the normal outputs of one stage flip-flop as clock signal for next stage flip-flop, connect the **complemented outputs** of one stage flip-flop as clock signal for next stage flip-flop. Complemented output goes from 1 to 0 is same as the normal output goes from 0 to 1.

Assume the initial status of T flip-flops from rightmost to leftmost is $Q_2Q_1Q_0 = 000$. Here, Q_2 & Q_0 are MSB & LSB respectively. We can understand the **working** of 3-bit asynchronous binary down counter from the following table.

No of negative edge of Clock	**Q_0**LSB	**Q_1**	**Q_2**MSB
0	0	0	0
1	1	1	1
2	0	1	1
3	1	0	1
4	0	0	1
5	1	1	0
6	0	1	0
7	1	0	0

Here Q_0 toggled for every negative edge of clock signal. Q_1 toggled for every Q_0 that goes from 0 to 1, otherwise remained in the previous state. Similarly, Q_2 toggled for every Q_1 that goes from 0 to 1, otherwise remained in the previous state.

The initial status of the T flip-flops in the absence of clock signal is $Q_2Q_1Q_0 = 000$. This is decremented by one for every negative edge of clock signal and reaches to the same value at 8^{th} negative edge of clock signal. This pattern repeats when further negative edges of clock signal are applied.

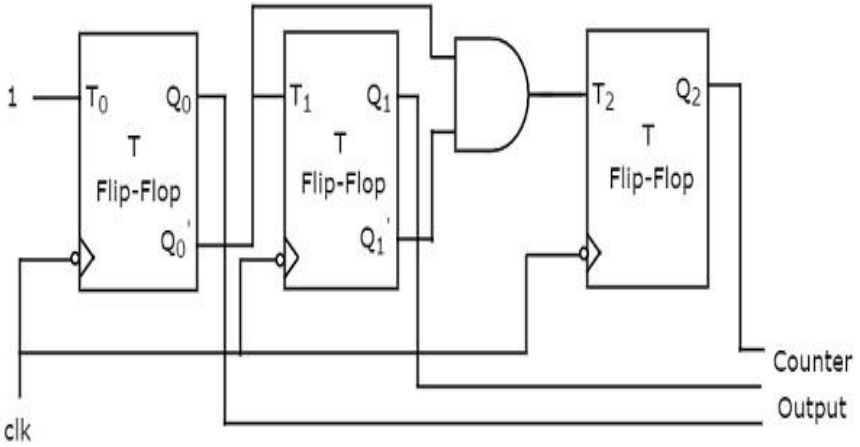
Synchronous Counters

If all the flip-flops receive the same clock signal, then that counter is called as **Synchronous counter**. Hence, the outputs of all flip-flops change affect at the same time.

Now, let us discuss the following two counters one by one.

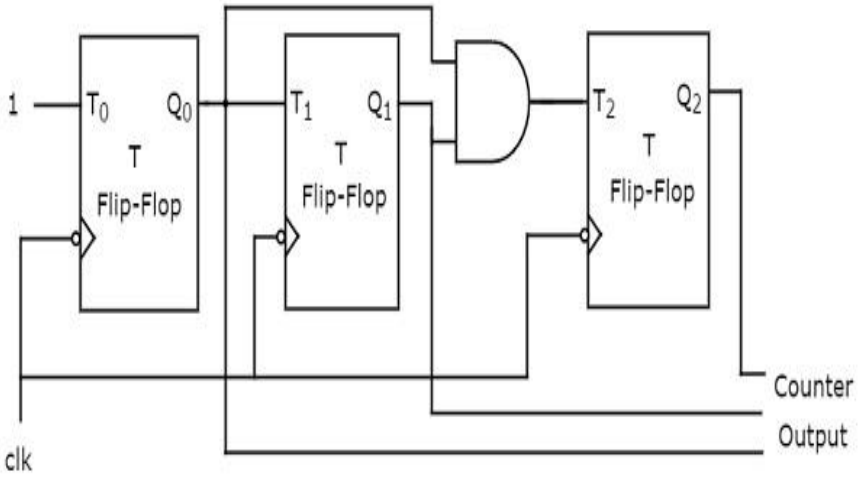
- Synchronous Binary up counter
- Synchronous Binary down counter

Synchronous Binary Up Counter



An 'N' bit Synchronous binary up counter consists of 'N' T flip-flops. It counts from 0 to $2^N - 1$. The **block diagram** of 3-bit Synchronous binary up counter is shown in the following figure.

The 3-bit Synchronous binary up counter contains three T flip-flops & one 2-input AND gate. All these flip-flops are negative edge triggered and



the outputs of flip-flops change affect synchronously. The T inputs of first, second and third flip-flops are 1, Q_0 & Q_1Q_0 respectively.
The output of first T flip-flop **toggles** for every negative edge of clock signal. The output of second T flip-flop toggles for every negative edge of clock signal if Q_0 is 1. The output of third T flip-flop toggles for every negative edge of clock signal if both Q_0 & Q_1 are 1.

Synchronous Binary Down Counter

An 'N' bit Synchronous binary down counter consists of 'N' T flip-flops. It counts from $2^N - 1$ to 0. The **block diagram** of 3-bit Synchronous binary down counter is shown in the following figure
The 3-bit Synchronous binary down counter contains three T flip-flops & one 2-input AND gate. All these flip-flops are negative edge triggered and the outputs of flip-flops change affect synchronously. The T inputs of first, second and third flip-flops are 1, Q_0' &' Q_1' Q_0' respectively.
The output of first T flip-flop **toggles** for every negative edge of clock signal. The output of second T flip-flop toggles for every negative edge of clock signal if Q_0' is 1. The output of third T flip-flop toggles for every negative edge of clock signal if both Q_1' & Q_0' are 1.

Tutor Marked Assignment

1. If the resolution of a digital-to-analog converter is approximately 0.4% of its full-scale range, then it is?
2. A bidirectional 4-bit shift register is storing the nibble 1101. Its input is HIGH. The nibble 1011 is waiting to be entered on the serial data-input line. After three clock pulses, the shift register is storing _____?
3. How many different states does a 3-bit asynchronous down counter have?
4. In a 3-bit asynchronous down counter, at the first negative transition of the clock, the counter content becomes _____?
5. For a 4 bit MOD-16 ripple counter using J-K flip-flop, the propagation delay of each flip flop is 50ns. What is the maximum clock frequency can be used?

SELF ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What type of flip-flop is commonly used in binary counters?**

A. D flip-flop
B. SR flip-flop
C. T flip-flop

D. JK flip-flop
Answer: C
Explanation: T flip-flops toggle their output on clock edges, making them ideal for counters.

2. **What is the range of a 3-bit binary up counter?**
A. 0 to 3
B. 0 to 7
C. 1 to 8
D. 0 to 15
Answer: B
Explanation: A 3-bit counter counts from 0 to 2^3 - 1 = 7.

3. **In an asynchronous counter, which flip-flop receives the system clock directly?**
A. First flip-flop
B. Second flip-flop
C. Last flip-flop
D. None
Answer: A
Explanation: Only the first flip-flop receives the system clock; others are triggered by previous outputs.

4. **What distinguishes asynchronous counters from synchronous counters?**
A. Use of D flip-flops
B. All flip-flops share the same clock
C. Flip-flops are triggered sequentially
D. They count in reverse
Answer: C
Explanation: In asynchronous counters, flip-flops are triggered by the output of the previous stage.

5. **What is the toggling condition for Q1 in a 3-bit asynchronous up counter?**
A. Every clock edge
B. When Q0 goes from 0 to 1
C. When Q0 goes from 1 to 0
D. When Q2 toggles
Answer: C
Explanation: Q1 toggles when Q0 transitions from 1 to 0.

6. **How is the clock signal routed in an asynchronous down counter?**
A. Directly to all flip-flops
B. Through complemented outputs of previous flip-flops
C. Through AND gates
D. Through serial input
Answer: B
Explanation: Complemented outputs are used to trigger the next flip-flop in down counters.

7. **What is the toggling condition for Q2 in a synchronous up counter?**

A. When Q0 is 1
B. When Q1 is 1
C. When Q0 and Q1 are both 1
D. Every clock edge
Answer: C

Explanation: Q2 toggles when both Q0 and Q1 are 1.

8. **Which counter type updates all flip-flops simultaneously?**

A. Asynchronous counter
B. Synchronous counter
C. Down counter
D. Ripple counter
Answer: B

E. **Explanation:** Synchronous counters use a common clock signal for all flip-flops.

9. **What is the toggling condition for Q1 in a synchronous down counter?**

A. When Q0 is 1
B. When Q0′ is 1
C. When Q1′ is 1
D. When Q2′ is 1
Answer: B

E. **Explanation:** Q1 toggles when Q0′ (complement of Q0) is 1.

10. **What is the output pattern of a 3-bit asynchronous down counter starting from 111?**

A. 111 → 110 → 101 → 100 → 011 → 010 → 001 → 000
B. 000 → 001 → 010 → 011 → 100 → 101 → 110 → 111
C. 111 → 000 → 111
D. 000 → 111 → 000

Answer: A
Explanation: The counter decrements from 7 (111) to 0 (000).

Fill in the Blank Questions

1. A 3-bit binary counter counts from 0 to _____.
Answer: 7

2. In asynchronous counters, flip-flops are triggered by the _____ of the previous flip-flop.
Answer: output

3. In synchronous counters, all flip-flops receive the same _____ signal.
Answer: clock

4. The T flip-flop toggles its output on every _____ edge of the clock signal.

5. **Answer:** negative
6. In a synchronous down counter, the third flip-flop toggles when both Q1′ and _____ are 1.
Answer: Q0′

Module 5 Memory Devices and Programmable Logic

Unit 1 Memory Devices and Classification
Unit 2 Programmable Logic Array (PLAs)
Unit 3 Programmable Logic Devices (PLDs)
Unit 4 Field-Programmable Gate Arrays (FGPAs)

Unit 1 Memory Devices and Classification

Memory structures are crucial in digital design. – ROM, PROM, EPROM, RAM, SRAM, (S) DRAM, RDRAM, …

➢ All memory structures have an address bus and a data bus – Possibly other control signals to control output etc. •E.g. 4 Bit Address bus with 5 Bit Data Bus ADDR DOUT

There are two types of memories that are used in digital systems:

- Random-access memory (RAM): perform both the write and read operations.
- Read-only memory (ROM): perform only the read operation.

The read-only memory is a programmable logic device. Other such units are the programmable logic array (PLA), the programmable array logic (PAL), and the field-programmable gate array (FPGA).

Random-Access Memory

A memory unit stores binary information in groups of bits called words.

- byte = 8 bits
- word = 2 bytes

The communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

- In random-access memory, the word locations may be thought of as being separated in space, with each word occupying one particular location.
- In sequential-access memory, the information stored in some medium is not immediately accessible, but is available only certain intervals of time. A magnetic disk or tape unit is of this type.
- In a random-access memory, the access time is always the same regardless of the particular location of the word.

- In a sequential-access memory, the time it takes to access a word depends on the position of the word with respect to the reading head position; therefore, the access time is variable.

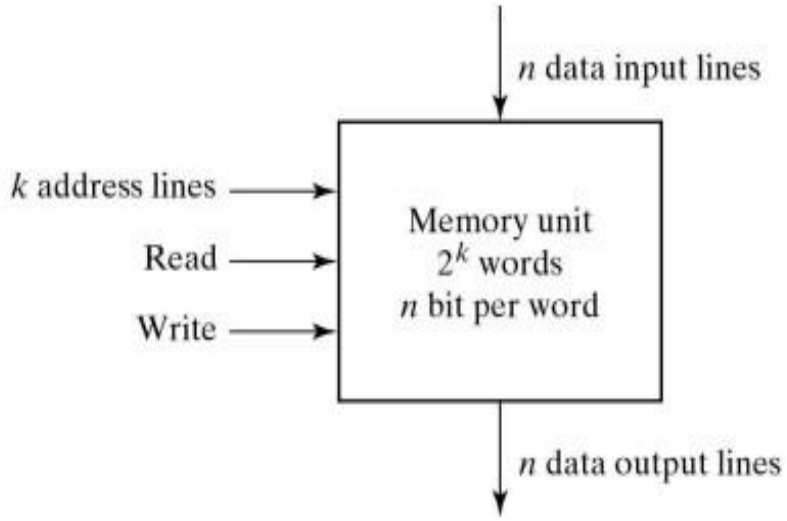


Fig. 7-2 Block Diagram of a Memory Unit

Static RAM

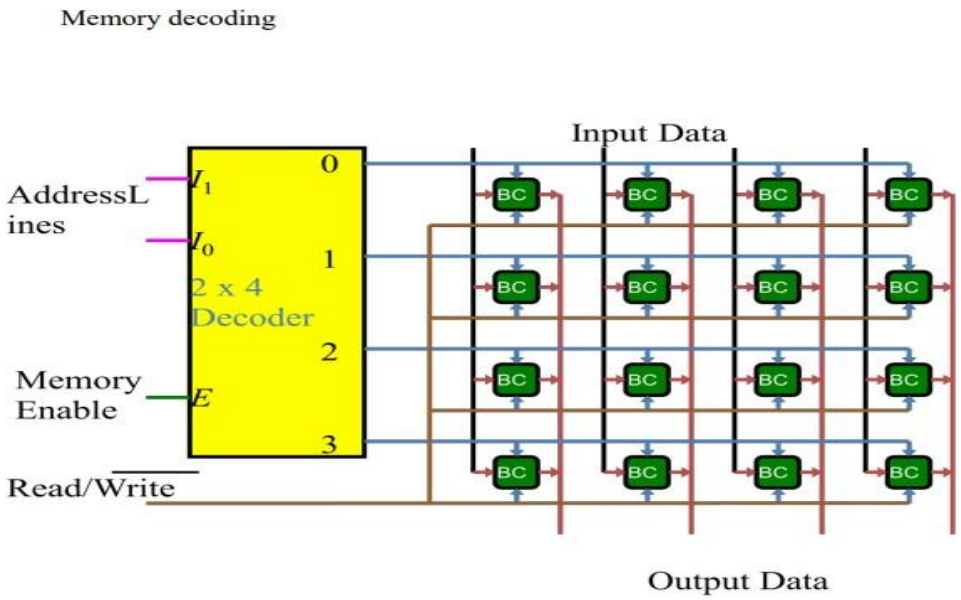
- SRAM consists essentially of internal latches that store the binary information.
- The stored information remains valid as long as power is applied to the unit.
- SRAM is easier to use and has shorter read and write cycles.
- Low density, low capacity, high cost, high speed, high power consumption.

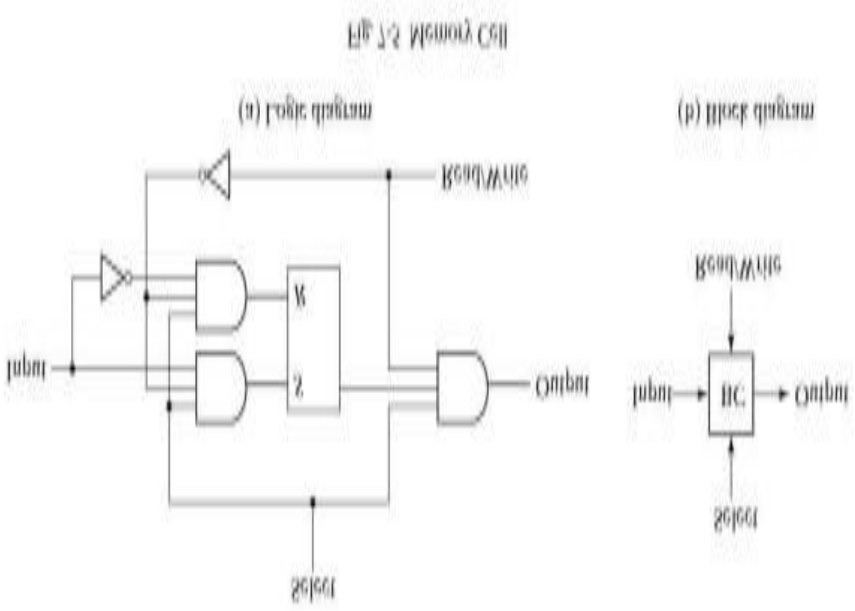
Dynamic RAM

- DRAM stores the binary information in the form of electric charges on capacitors.
- The capacitors are provided inside the chip by MOS transistors.
- the capacitors tends to discharge with time and must be periodically recharged by refreshing the dynamic memory.
- DRAM offers reduced power consumption and larger storage c High density, high capacity, low cost, low speed, low power consumption.
- The equivalent logic of a binary cell that stores one bit of information is apacity in a single memory chip.

shown below.

- Read/Write = 0, select = 1, input data to S-R latch
- Read/Write = 1, select = 1, output data from S-R latch





SELF-ASSESMENT EXERCISES

Multiple Choice Questions

1. What is the primary function of memory in digital systems?

A. To perform calculations

B. To store data and instructions ✅

C. To display output

D. To generate clock signals

2. Which type of memory is volatile?
A. ROM
B. Flash
C. RAM ✅
D. EEPROM
3. What does ROM stand for?
A. Random Output Memory
B. Read Only Memory ✅
C. Real Operating Module
D. Rapid Online Memory
4. Which memory retains data even when power is off?
A. RAM
B. ROM ✅
C. Cache
D. Register
5. What type of memory is used for temporary data storage?
A. ROM
B. RAM ✅
C. Flash
D. Hard Disk
6. Which of the following is a non-volatile memory?
A. RAM
B. Cache
C. ROM ✅
D. DRAM
7. What is the function of cache memory?
A. Long-term storage
B. High-speed temporary storage ✅
C. Permanent data retention
D. Data encryption
8. Which memory type is programmable and erasable?
A. RAM
B. PROM
C. EPROM ✅
D. Cache
9. What does EEPROM stand for?
A. Electrically Erasable Programmable Read Only Memory ✅
B. Enhanced Erasable Programmed Output Module
C. Electronic Erased Primary ROM
D. External Erasable Processing Memory
10. Which memory type is used to store BIOS in computers?
A. RAM
B. ROM ✅
C. Cache
D. DRAM

Fill in the Blank Questions

1. _____ memory loses its contents when power is turned off. → *RAM*
2. _____ stands for Read Only Memory. → *ROM*
3. _____ memory is used for high-speed temporary storage. → *Cache*
4. _____ is a type of memory that can be electrically erased and reprogrammed. → *EEPROM*
5. The BIOS in a computer is stored in _____. → *ROM*

Unit 2 Programmable Logic Array

- The decoder in PROM is replaced by an array of AND gates that can be programmed to generate any product term of the input variables.
- The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.
- The output is inverted when the XOR input is connected to 1 (since x⊕1 = x'). The output doesn't change and connect to 0 (since x⊕0 = x).

F1 = AB'+AC+A'BC'
F2 = (AC+BC)'

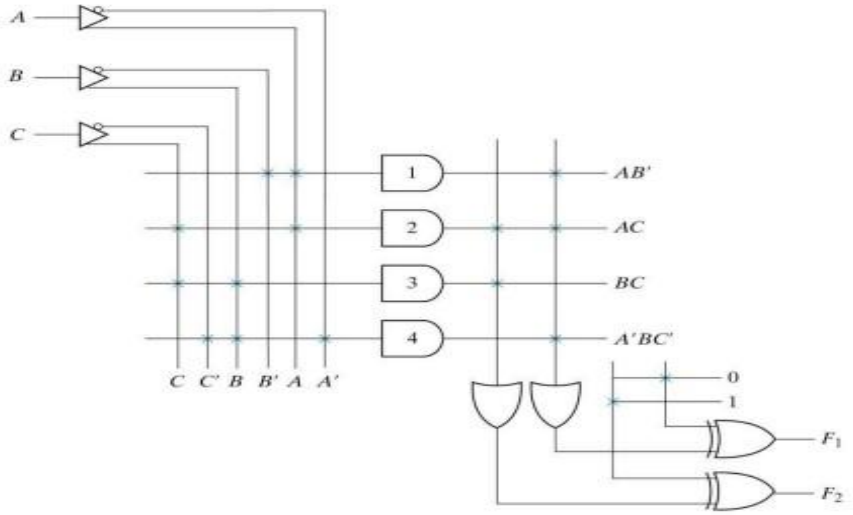


Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

Implement the following two Boolean functions with a PLA:

- $F_1(A, B, C) = \sum(0, 1, 2, 4)$
- $F_2(A, B, C) = \sum(0, 5, 6, 7)$

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

C

$F_1 = A'B' + A'C' + B'C'$

$F_1 = (AB + AC + BC)'$

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

C

$F_2 = AB + AC + A'B'C'$

$F_2 = (A'C + A'B + AB'C')'$

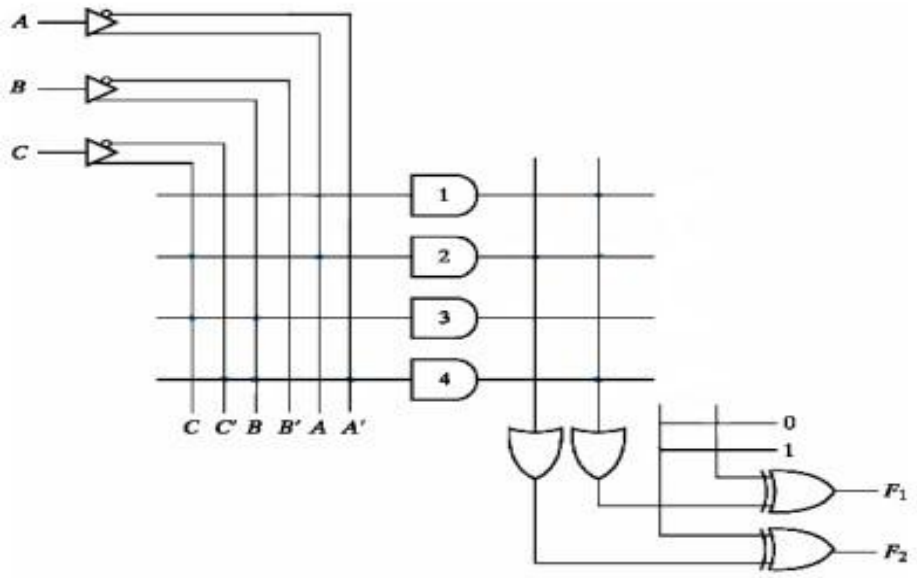
- Both the true and complement of the functions are simplified in sum of products.
- We can find the same terms from the group terms of the functions of F_1, F_1', F_2 and F_2' which will make the minimum terms.

$$F1 = (AB + AC + BC)'$$
$$F2 = AB + AC + A'B'C'$$

PLA programming table

	Product term	Inputs A	B	C	Outputs (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

Fig. 7-15 Solution to Example 7-2



SELF ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What component replaces the decoder in a PLA compared to a PROM?**

A. OR gates
B. XOR gates
C. AND gates
D. Flip-flops

Answer: C

Explanation: In a PLA, the decoder is replaced by a programmable array of AND gates to generate product terms.

2. **What is the role of the OR gates in a PLA?**

A. To invert the output
B. To generate product terms
C. To combine product terms into sum-of-products expressions
D. To store binary data

Answer: C

Explanation: OR gates in a PLA combine the product terms to form the final output functions.

3. **What happens when the XOR input in a PLA is connected to 1?**

A. The output is unchanged
B. The output is inverted
C. The input is disabled
D. The output is multiplied

Answer: B

Explanation: Connecting XOR input to 1 inverts the output since x \oplus 1 = x'.

4. **Which of the following is a valid product term for $F_1 = AB' + AC + A'BC'$?**

A. AB
B. AC
C. A′B′C
D. BC

Answer: B

Explanation: AC is one of the product terms used in the sum-of-products expression for F_1.

5. **How many product terms are needed to implement $F_1 = AB' + AC + A'BC'$?**

A. 2
B. 3
C. 4

D. 5
Answer: B
Explanation: F_1 uses three product terms: AB′, AC, and A′BC′.
6. **What is the simplified Boolean expression for $F_2 = (AC + BC)'$?**
A. AC + BC
B. A′C′ + B′C′
C. A′ + B′
D. F_2 = A′B′ + A′C′ + B′C′
Answer: D
Explanation: The complement of AC + BC is simplified using DeMorgan's laws and K-map techniques.
7. **What does the PLA programming table specify?**
A. Truth table of the circuit
B. Input-output voltage levels
C. Mapping of product terms to inputs and outputs
D. Clock timing
Answer: C
Explanation: The PLA programming table shows how product terms are formed and which outputs they affect.
8. **How many inputs are used in the PLA shown in the example?**
A. 2
B. 3
C. 4
D. 5
Answer: B
Explanation: The PLA uses three inputs: A, B, and C.
9. **What is the output of a PLA when the XOR input is connected to 0?**
A. The output is inverted
B. The output is unchanged
C. The output is disabled
D. The output is multiplied
Answer: B
Explanation: When XOR input is 0, the output remains unchanged since x \oplus 0 = x.
10. **Which of the following Boolean functions is implemented using a PLA in the example?**
A. $F_1 = AB + AC$
B. $F_2 = AB + AC + A'B'C'$
C. $F_1 = AB' + AC + A'BC'$
D. F_2 = AC + BC
Answer: C
Explanation: F_1 is implemented as AB′ + AC + A′BC′ in the PLA example.
Fill in the Blank Questions

1. In a PLA, the decoder is replaced by a programmable array of _____ gates.

Answer: AND

2. The output of a PLA is inverted when the XOR input is connected to _____.

Answer: 1

3. The Boolean function $F_1 = AB' + AC + A'BC'$ is implemented using _____ product terms.

Answer: three

4. The PLA programming table maps product terms to _____ and _____.

Answer: inputs, outputs

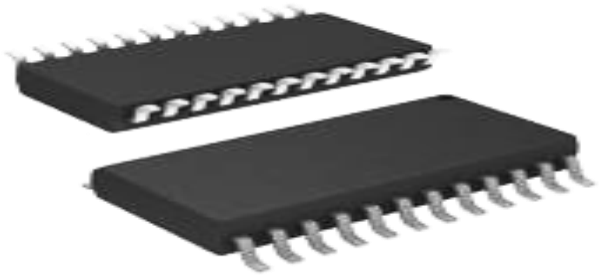
5. The simplified expression for $F_2 = (AC + BC)'$ is derived using _____ maps.

Answer: Karnaugh

Unit 3 Programmable Logic Device

What is PLD?

A PLD is a class of integrated circuit that may be configured to carry out a number of different digital logic operations. Common uses for them include simple logic circuits, state machines, and counters, which only need a few logic gates to function.



PLD programming

A number of techniques can be used to program PLDs, including:

- **In-circuit programming (ICP)**: PLDs can be programmed while they are inserted in a circuit using a technique called in-circuit programming (ICP). Typically, production applications adopt this technique.
- **External programming (EPROM)**: PLDs can be programmed using external programming (EPROM), which involves removing the PLD from the circuit and using an external programmer. Applications in development and prototype are frequently employed with this technique.
- **Flash programming (Flash)**: Flash programming is a similar technique to EPROM programming for PLDs, however it allows for numerous PLD reprogramming. This approach is frequently employed in situations when it may be necessary to modify the logic function.

PLDs are often programmed using a software tool that creates the PLD's programming data. Typically, the software tool has a graphical user interface that enables the user to design the PLD's logic function.

The software tool generates the PLD programming data when the logic function has been created. Then, through one of the aforementioned techniques, the programming data is downloaded to the PLD.

SELF ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What does PLD stand for?**

A. Programmable Logic Decoder

B. Programmable Logic Device

C. Peripheral Logic Driver
D. Parallel Logic Design
Answer: B
Explanation: PLD stands for Programmable Logic Device.

2. **Which of the following is a typical use for PLDs?**
A. Image processing
B. Simple logic circuits
C. Audio amplification
D. Data storage
Answer: B
Explanation: PLDs are commonly used for simple logic circuits, state machines, and counters.

3. **What is the main advantage of PLDs?**
A. Fixed functionality
B. High power consumption
C. Configurability for different logic operations
D. Analog signal processing
Answer: C
Explanation: PLDs can be configured to perform various digital logic operations.

4. **Which programming method allows PLDs to be programmed while still in the circuit?**
A. EPROM
B. Flash
C. In-circuit programming (ICP)
D. External programming
Answer: C
Explanation: ICP allows PLDs to be programmed without removing them from the circuit.

5. **Which programming method is commonly used for prototypes and development?**
A. ICP
B. EPROM
C. Flash
D. ROM
Answer: B
Explanation: EPROM programming is often used in development and prototyping.

6. **What is a key benefit of flash programming for PLDs?**
A. It is permanent
B. It allows multiple reprogrammings
C. It requires no software
D. It uses analog signals
Answer: B
Explanation: Flash programming enables PLDs to be reprogrammed multiple times.

7. **What kind of interface do most PLD software tools provide?**
A. Command-line only
B. Graphical user interface
C. Text editor
D. Voice-controlled interface
Answer: B
Explanation: Most PLD tools offer a GUI for designing logic functions.
8. **What does the software tool generate for PLD programming?**
A. Clock signals
B. Logic gates
C. Programming data
D. Binary counters
Answer: C
Explanation: The software tool generates the programming data for the PLD.
9. **Which method involves removing the PLD from the circuit for programming?**
A. ICP
B. Flash
C. EPROM
D. RAM
Answer: C
Explanation: EPROM programming requires the PLD to be removed from the circuit.
10. **Which technique is best suited for production environments?**
A. EPROM
B. Flash
C. ICP
D. Manual wiring
Answer: C
Explanation: ICP is typically used in production applications.
Fill in the Blank Questions
1. PLD stands for _____ Logic Device.
Answer: Programmable
2. PLDs are commonly used in _____ logic circuits and counters.
Answer: simple
3. _____ programming allows PLDs to be programmed while still in the circuit.
Answer: In-circuit
4. _____ programming allows PLDs to be reprogrammed multiple times.
Answer: Flash
5. A _____ user interface is typically used in PLD software tools.
Answer: graphical

Unit 4 Field-Programmable Gate Arrays

What is a FPGA Board?

FPGA is the most versatile type of PLD. They are made up of many logic components that can be coupled in many ways. They are therefore perfect for applications that need a lot of flexibility and customization. FPGAs are frequently used in programs for embedded systems, signal processing, networking, cryptography, industrial automation, and other things.

FPGA programming

A programmable routing network can be used to connect the grid of logic cells that make up FPGAs. This makes it possible to arrange FPGAs to carry out a wide range of functions.
A number of techniques can be used to program FPGAs, including:

- **High-level synthesis (HLS)**: High-level synthesis (HLS) is a technique for programming FPGAs that creates the programming information for the FPGA using a high-level programming language, such C or C++. Usually, complicated applications that demand a high level of performance adopt this approach.
- **Verilog HDL**: An FPGA's logic functions are described using Verilog HDL, a hardware description language. Usually, medium-to-complex applications adopt this approach.
- **VHDL**: Similar to Verilog HDL, VHDL is a hardware description language. The majority of the time, complicated applications adopt this technique.

A software tool that produces the programming data for the FPGA is commonly used to program FPGAs. Typically, the software tool has a graphical user interface that enables the user to design the FPGA's logic function.
The software tool generates the FPGA programming data when the logic function has been created. A programming tool, such as a USB programmer, is then used to download the programming data to the FPGA.

Tutor Marked Assignment

1. Design a Full Adder using ROM and PLA
2. Design a 5X2 RAM using D Flip-flop
3. How many 16K * 4 RAMs are required to achieve a memory with a capacity of 64K and a word length of 8 bits?
4. The complex programmable logic device contains several PLD blocks and _____?
5. The difference between a PAL & a PLA is _____?

SELF-ASSESMENT EXERCISES

Multiple Choice Questions (MCQs)

1. **What does FPGA stand for?**

A. Flexible Programmable Gate Array
B. Field-Programmable Gate Array
C. Fixed-Performance Gate Array
D. Fast Processing Gate Architecture

Answer: B

Explanation: FPGA stands for Field-Programmable Gate Array.

2. **Which of the following best describes an FPGA?**

A. A fixed-function processor
B. A memory storage device
C. A customizable logic device
D. A digital-to-analog converter

Answer: C

Explanation: FPGAs are highly flexible and customizable logic devices.

3. **Which application is NOT typically associated with FPGAs?**

A. Cryptography
B. Industrial automation
C. Word processing
D. Signal processing

Answer: C

Explanation: Word processing is typically handled by general-purpose CPUs, not FPGAs.

4. **What connects the logic cells in an FPGA?**

A. Serial bus
B. Clock signal
C. Programmable routing network

D. USB interface
Answer: C
Explanation: A programmable routing network links the logic cells in an FPGA.
5. **Which programming method uses high-level languages like C or C++?**
A. VHDL
B. Verilog HDL
C. High-Level Synthesis (HLS)
D. Assembly
Answer: C
Explanation: HLS allows FPGA programming using high-level languages.
6. **Which hardware description language is similar to Verilog HDL?**
A. Python
B. VHDL
C. Java
D. C++
Answer: B
Explanation: VHDL is another hardware description language used for FPGA programming.
7. **What is the role of the software tool in FPGA programming?**
A. To generate clock signals
B. To create logic gates
C. To produce programming data for the FPGA
D. To store binary data
Answer: C
Explanation: The software tool designs the logic and generates programming data.
8. **What is typically used to download programming data to an FPGA?**
A. HDMI cable
B. USB programmer
C. Ethernet cable
D. Serial port
Answer: B
Explanation: A USB programmer is commonly used to load data into the FPGA.
9. **Which method is most suitable for medium-to-complex FPGA applications?**
A. VHDL
B. Verilog HDL
C. HLS
D. Python
Answer: B

Explanation: Verilog HDL is widely used for medium-to-complex FPGA designs.

10. **What kind of interface do most FPGA software tools provide?**

A. Command-line only
B. Graphical user interface
C. Text editor
D. Voice-controlled interface

Answer: B

Explanation: Most FPGA tools offer a GUI for designing logic functions.

Fill in the Blank Questions

1. FPGA stands for _____ Programmable Gate Array.
Answer: Field
2. FPGAs are made up of a grid of _____ cells.
Answer: logic
3. _____ is a high-level programming technique used for FPGA design.
Answer: High-Level Synthesis (HLS)
4. _____ and VHDL are hardware description languages used to program FPGAs.
Answer: Verilog HDL
5. A _____ is commonly used to transfer programming data to an FPGA.
Answer: USB programmer