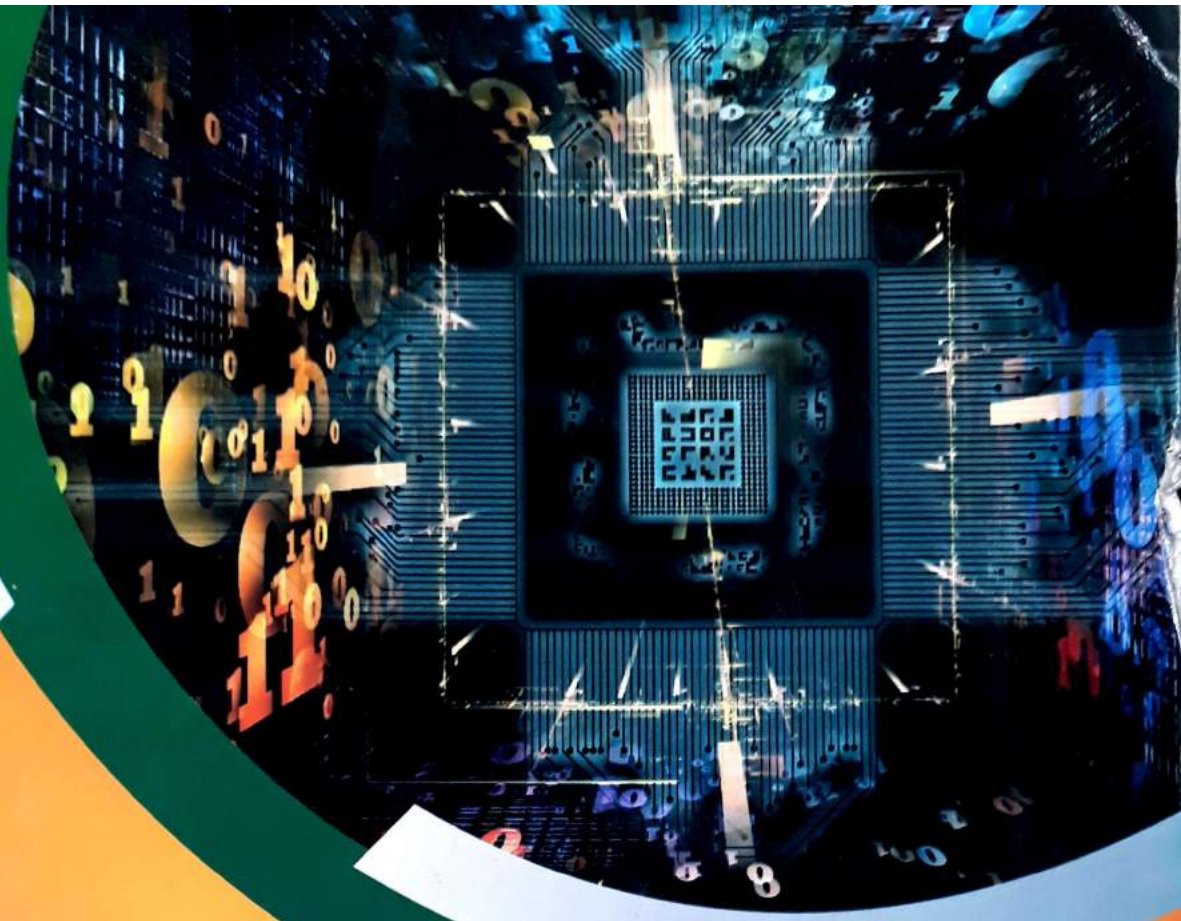


Strictly as per the New Revised Syllabus (Rev - 2016) of

Mumbai University

w.e.f. academic year 2017-2018

(As per Choice Based Credit and Grading System)

MU

Digital Logic Design and Analysis

(Code : CSC302)

Semester III - Computer Engineering

J. S. Katre **Vaishali S. Joshi**

Tech-Max Publications, Pune
Innovation Throughout Engineering Division

MO202A Price ₹ 455/-

Scanned by CamScanner

Digital Logic Design and Analysis

(Code : CSC302)

Semester III - Computer Engineering
(Mumbai University)

Strictly as per New Choice Based Credit and Grading System Syllabus (Revise 2016) of Mumbai University with effective from Academic Year 2017-2018

J. S. Katre
M.E. (Electronics and Telecommunication)
Formerly, Assistant Professor
Department of Electronics Engineering
Vishwakarma Institute of Technology (V.I.T.), Pune.
Maharashtra, India

Vaishali S. Joshi
B.E. (Computer Science and Engineering)

Tech-Max Publications, Pune
Innovation Throughout Engineering Division

JAGDISH BOOK DEPOT
N0202A

Scanned by CamScanner

Digital Logic Design and Analysis : (Code : CSC302)
(Semester III, Computer Engineering, Mumbai University)
J. S. Katre, V. S. Joshi.

Copyright © by Authors. All rights reserved. No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

First Printed in India : January 2002
First Edition : July 2017 **(For Mumbai University)**

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : Image Offset, Dugane Ind. Area Survey No. 28/25, Dhayari, Near Pari Company, Pune – 41, Maharashtra State, India. E-mail : rahulshahimage@gmail.com

ISBN : 978-93-86742-07-0

Published by
Tech-Max Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner, Pune - 411 009. Maharashtra State, India
Ph : 91-20-24225065, 91-20-24217965, Fax 020-24228978.
Email : info@techmaxbooks.com,
Website : www.techmaxbooks.com

[CSC302] (FID : TM337) (Book Code : MO202A)

Preface

Dear students,

We are extremely happy to present the book of **"Digital Logic Design and Analysis"** for you. We have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

We are thankful to Shri. Pradeep Lunawat and Shri. Sachin Shah for the encouragement and support that they have extended. We are also thankful to the staff members of Tech-Max Publications and others for their efforts to make this book as good as it is. We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let us know, because that will help us to improve further.

We are also thankful to our family members and friends for their patience and encouragement.

- Authors

□□□

Scanned by CamScanner

Syllabus...

Digital Logic Design and Analysis : Sem. III, (Computer Engg. (MU))

Course Code	Course Name	Credits
CSC302	Digital Logic Design and Analysis	4

Course Objectives :

1. To introduce the fundamental concepts and methods for design of digital circuits and a pre-requisite for computer organization and architecture, microprocessor systems.
2. To provide the concept of designing Combinational and sequential circuits.
3. To provide basic knowledge of how digital building blocks are described in VHDL.

Course Outcomes :

At the end of the course student should be able :

1. To understand different number systems and their conversions.
2. To analyze and minimize Boolean expressions.
3. To design and analyze combinational circuits.
4. To design and analyze sequential circuits
5. To understand the basic concepts of VHDL.
6. To study basics of TTL and CMOS Logic families.

Module 1

Number Systems and Codes :

Introduction to number system and conversions : Binary, Octal, Decimal and Hexadecimal number systems, Binary arithmetic : Addition, Subtraction (1's and 2's complement), Multiplication and division, Octal and Hexadecimal arithmetic : Addition and subtraction (7's and 8's complement method for octal) and (15's and 16's complement method for Hexadecimal), Codes : Gray code, BCD code, Excess – 3 code, ASCII code, Error detection and correction : Hamming codes. **(Refer chapters 1, 2 and 3)**

Module 2

Boolean Algebra and Logic Gates :

Theorems and properties of Boolean algebra, Boolean functions, Boolean function reduction using Boolean laws, Canonical forms, standard SOP and POS form, Basic digital gates : NOT, AND, OR, NAND, NOR, EX-OR, EX-NOR, positive and negative logic, K-map method : 2 Variable, 3 Variable, 4 Variable, Don't care condition, Quine Mc-Clusky method, NAND-NOR realization. **(Refer chapters 4 and 5)**

Module 3

Combinational Logic Design :

Introduction, Half and Full adder, Half subtractor, Full subtractor, Four Bit ripple adder, Look ahead carry adder, 4 bit adder subtractor, One digit BCD Adder, Multiplexer, Multiplexer tree, Demultiplexer, Demultiplexer tree, Encoders : Priority encoder, Decoders, One bit, Two bit, 4-bit Magnitude comparator, ALU IC 74181. **(Refer chapter 6)**

Module 4

Sequential Logic Design :

Introduction : SR latch, Concepts of Flip Flops : SR, D, J-K, T, Truth Tables and Excitation Tables of all types, Race around condition, Master Slave J-K Flip Flops, Timing Diagram, Flip-flop conversion, State machines, state diagrams, State table, concept of Moore and Mealy machine, Counters : Design of Asynchronous and Synchronous Counters, Modulus of the Counters, UP-DOWN counter, Shift Registers : SISO, SIPO, PIPO, PISO, Bidirectional Shift Register, Universal Shift Register, Ring and twisted ring/Johnson Counter, sequence generator. **(Refer chapters 7, 8 and 9)**

Module 5

Introduction to VHDL :

Introduction : Fundamental building blocks Library, Entity, Architecture, Modeling Styles, Concurrent and sequential statements, simple design examples for combinational circuits and sequential circuits. **(Refer chapter 10)**

Module 6

Digital Logic Families :

Introduction : Terminologies like Propagation Delay, Power Consumption, Fan in and Fan out, current and voltage parameters, noise margin with respect to TTL and CMOS Logic and their comparison. **(Refer chapter 11)**

□□□

Scanned by CamScanner

Module 1

Chapter 1 : Number Systems 1-1 to 1-29

1.1 Introduction 1-1
1.1.1 Signals 1-1
1.1.2 Types of Signals 1-1
1.1.3 Digital Signals 1-1
1.2 Digital Systems 1-2
1.3 Binary Logic and Logic Levels 1-2
1.3.1 Positive Logic 1-2
1.3.2 Negative Logic 1-2
1.4 Number Systems 1-3
1.4.1 Important Definitions Related to All Numbering Systems 1-3
1.4.2 Various Numbering Systems 1-4
1.5 The Decimal Number System 1-4
1.5.1 Characteristics of a Decimal System 1-4
1.5.2 Most Significant Digit (MSD) 1-5
1.5.3 Least Significant Digit (LSD) 1-5
1.6 The Binary Number System 1-5
1.6.1 Binary Numbers from $(0)_{10}$ to $(15)_{10}$ 1-6
1.6.2 Binary Number Formats 1-6
1.6.3 Disadvantages of the Binary System 1-6
1.7 Octal Number System 1-7
1.7.1 Applications of Octal System 1-7
1.8 Hexadecimal Number System 1-8
1.9 Relation between Binary, Decimal, Octal and Hexadecimal Numbers 1-9
1.10 Conversion of Number Systems 1-10
1.11 Conversions Related to Decimal System 1-10
1.11.1 Conversion from Radix r to Decimal 1-10
1.11.2 Conversion from Decimal to Other Systems 1-11
1.11.2.1 Successive Division for Integer Part Conversion 1-11
1.11.2.2 Successive Multiplication for Fractional Part Conversion 1-13
1.11.2.3 Conversion of Mixed Decimal Number to Any Other Radix 1-15
1.12 Conversion from Binary to Other Systems 1-20
1.12.1 Conversion from Binary to Decimal 1-20
1.12.2 Binary to Octal Conversion 1-20

1.12.3 Binary to Hex Conversion 1-21
1.13 Conversion from Other Systems to Binary System 1-21
1.13.1 Conversion from Decimal to Binary 1-21
1.13.2 Octal to Binary Conversion 1-22
1.13.3 Hex to Binary Conversion 1-22
1.14 Conversion from Octal to Other Systems 1-22
1.14.1 Octal to Hex Conversion 1-23
1.14.2 Conversion from Other Systems to Octal 1-27
1.15 Conversions Related to Hexadecimal System 1-27
1.15.1 Other Systems to Hex 1-27
1.15.2 Hex to Other Systems 1-27

Module 1

Chapter 2 : Arithmetic in Number Systems 2-1 to 2-33

2.1 Introduction 2-1
2.1.1 Binary Addition 2-1
2.1.2 Sum and Carry 2-2
2.1.3 Addition of Large Binary Numbers 2-2
2.1.4 Binary Subtraction 2-2
2.1.5 Subtraction and Borrow 2-3
2.1.6 Subtraction of Larger Binary Numbers 2-3
2.1.7 Binary Multiplication 2-4
2.1.8 Binary Division 2-5
2.2 Unsigned Binary Numbers 2-6
2.2.1 Important Features of Unsigned Numbers 2-6
2.3 Sign-Magnitude Numbers 2-7
2.3.1 Range of Sign-Magnitude Numbers 2-7
2.4 Complements 2-8
2.4.1 Types of Complements 2-8
2.4.2 1's Complement 2-8
2.4.3 2's Complement 2-8
2.5 Binary Subtraction using 1's and 2's Complements 2-9
2.5.1 Subtraction using 1's Complement 2-9
2.5.2 Binary Subtraction using 2's Complement Method 2-11
2.6 Octal Arithmetic 2-17
2.6.1 Octal Addition 2-17

Scanned by CamScanner

2.6.2 Subtraction of Octal Numbers 2-17
2.6.3 Method 1 : Direct Subtraction of Octal Numbers 2-18
2.6.4 Method 2 : Subtraction using 7's Complement 2-18
2.6.5 Method 3 : Subtraction using 8's Complement 2-20
2.6.6 Octal Multiplication 2-20
2.6.7 Octal Division 2-21
2.7 Hexadecimal Arithmetic 2-22
2.7.1 Hex Addition 2-22
2.7.2 Hex Subtraction 2-23
2.7.3 Method 1 : Direct Subtraction 2-23
2.7.4 Hex Subtraction using 15's Complement 2-24
2.7.5 Hex Subtraction using 16's Complement 2-26
2.7.6 Hexadecimal Multiplication 2-28
2.7.7 Hex Division 2-29
2.8 Solved University Examples 2-29

Module 1

Chapter 3 : Codes **3-1 to 3-40**

3.1 Concept of Coding 3-1
3.1.1 Advantages of Binary Codes 3-1
3.1.2 Applications of Binary Codes 3-1
3.2 Classification of Codes 3-1
3.2.1 Weighted Binary Codes 3-2
3.2.2 Non Weighted Codes 3-2
3.2.3 Reflective Codes 3-2
3.2.4 Sequential Codes 3-3
3.2.5 Alphanumeric Codes 3-3
3.2.6 Error Detecting and Correcting Codes 3-3
3.3 Binary Coded Decimal (BCD) Code 3-3
3.3.1 Comparison with Binary 3-4
3.3.2 Advantages of BCD Codes 3-5
3.3.3 Disadvantages 3-5
3.4 BCD Arithmetic 3-6
3.4.1 BCD Addition 3-6
3.4.2 BCD Subtraction 3-9
3.5 Non – weighted Codes 3-14

3.6 Excess – 3 Code 3-14
3.7 Gray Code 3-16
3.7.1 Reflective Property 3-17
3.7.2 Application of Gray Code 3-18
3.7.3 Advantages of Gray Code 3-18
3.7.4 Gray-to-Binary Conversion 3-19
3.7.5 Binary to Gray Conversion 3-19
3.8 Alphanumeric Codes 3-21
3.8.1 ASCII - (American Standard Code for Information Interchange) 3-22
3.8.2 Extended ASCII Characters 3-23
3.9 Code Conversions 3-24
3.9.1 Binary to BCD Conversion 3-24
3.9.2 BCD to Binary Conversion 3-25
3.9.3 BCD to Excess – 3 3-25
3.9.4 Excess – 3 to BCD Conversion 3-26
3.10 Error Detection and Correction Codes 3-27
3.10.1 Parity 3-28
3.10.2 Use of Parity Bit to Decide Parity 3-28
3.10.3 Error Detection using ASCII Code 3-30
3.11 Hamming Code (Error Correcting Code) 3-31
3.11.1 7-Bit Hamming Code 3-31
3.11.2 Minimum Number of Parity Bits 3-31
3.11.3 Deciding the Values of Parity Bits 3-32
3.11.4 Deciding the Parity Bits for a 7 Bit Code 3-32
3.11.5 Detection and Correction of Errors 3-36
3.11.6 Limitation of Hamming Code 3-39

Module 2

Chapter 4 : Logic Gates and Boolean Algebra **4-1 to 4-36**

4.1 Introduction 4-1
4.1.1 Basic Logical Operations (Logic Variables) 4-1
4.1.2 Logic Gates 4-1
4.1.3 Gates, Symbols and Boolean Expression 4-2
4.2 Basic Definitions 4-3
4.2.1 Postulates 4-3
4.3 Axiomatic Definition of Boolean Algebra 4-4

Scanned by CamScanner

4.3.1 Boolean Postulates and Laws 4-5
4.3.2 Differences between Boolean and Ordinary Algebra 4-6
4.4 Two Valued Boolean Algebra 4-6
4.5 Basic Theorems and Properties of Boolean Algebra 4-8
4.5.1 Duality 4-8
4.5.2 Basic Theorems 4-9
4.5.3 De-Morgan's Theorems 4-11
4.6 Boolean Expression and Boolean Function 4-12
4.6.1 Truth Table Formation 4-12
4.6.2 Examples on Reducing the Boolean Expression 4-13
4.6.3 Complement of a Function 4-20
4.7 Logic Gates 4-20
4.7.1 Classification of Logic Gates 4-21
4.8 NOT Gate or Inverter 4-21
4.8.1 Operation for Pulsed Input 4-21
4.8.2 Standard Package (IC Gate) 4-22
4.9 AND Gate 4-22
4.9.1 Standard Package (IC Gate) 4-22
4.10 The OR Gate 4-22
4.10.1 Standard Package (IC Gate) 4-23
4.11 Special Type of Gates or Derived Gates 4-23
4.11.1 The EX-OR Gate 4-23
4.11.2 The EX-NOR Gate 4-24
4.11.3 Realization of Switching Functions 4-24
4.11.4 To Draw a Logic Circuit from Boolean Equation 4-24
4.11.5 To Write a Boolean Expression for a Logic Circuit 4-25
4.12 Universal Gates 4-27
4.12.1 The NAND Gate 4-28
4.12.2 Universality of NAND Gate 4-28
4.12.3 The NOR Gate 4-30
4.12.4 Universality of NOR Gate 4-31
4.12.5 Solved Examples 4-33

Module 2

Chapter 5 : Logic Minimization and Reduction Techniques ..... 5-1 to 5-80

5.1 System or Circuit 5-1

5.1.1 Digital Systems 5-1
5.1.2 Input Output Relation 5-1
5.1.3 Types of Digital Systems 5-2
5.1.4 Combinational Circuit Design 5-3
5.2 Standard Representations for Logical Functions 5-3
5.2.1 Sum-of-Products (SOP) Form 5-4
5.2.2 Product of the Sums Form (POS) 5-4
5.2.3 Standard or Canonical SOP and POS Forms 5-5
5.2.4 Conversion of a Logic Expression to Standard SOP or POS Form 5-6
5.3 Concepts of Minterm and Maxterm 5-8
5.3.1 How are the Minterm and Maxterm of Table 5.3.1 Obtained ? 5-8
5.3.2 Representation of Logical Expressions using Minterms and Maxterms 5-9
5.3.3 Writing SOP and POS Forms for a Given Truth Table 5-9
5.3.4 To Write Standard SOP Expression for a Given Truth Table 5-10
5.3.5 To Write a Standard POS Expression for a Given Truth Table 5-11
5.3.6 Conversion from SOP to POS and Vice Versa 5-11
5.4 Methods to Simplify the Boolean Functions 5-13
5.4.1 Algebraic Simplification 5-13
5.4.2 Disadvantages of Algebraic Method of Simplification 5-15
5.5 Karnaugh-Map Simplification (The Map Method) 5-15
5.5.1 K-map Structure 5-16
5.5.2 K-map Boxes and Associated Product Terms 5-17
5.5.3 Alternative Way to Label the K-map 5-18
5.5.4 Truth Table to K-map 5-18
5.5.5 Representation of Standard SOP form on K-map 5-20
5.6 Simplification of Boolean Expressions using K-map 5-20
5.6.1 How does Simplification Takes Place ? 5-21
5.6.2 Way of Grouping (Pairs, Quads and Octets) 5-21
5.6.3 Grouping Two Adjacent One's (Pairs) 5-22
5.6.4 Grouping Four Adjacent Ones (Quad) 5-23
5.6.5 Grouping Eight Adjacent Ones (Octet) 5-25
5.6.6 Summary of Rules Followed for K-Map Simplification 5-26
5.7 Minimization of SOP Expressions (K Map Simplification) 5-26
5.7.1 Elimination of a Redundant Group 5-29
5.7.2 Minimization of Logic Functions not Specified in Standard SOP Form 5-31
5.7.3 Don't Care Conditions 5-33

Scanned by CamScanner

5.7.4 Disadvantages of K-Map Technique 5-38
5.8 NAND-NAND Implementation 5-38
5.8.1 NAND - NAND Implementation using Alternative Representation 5-39
5.9 Product of Sum (POS) Simplification 5-42
5.9.1 K-map Representation of POS Form 5-42
5.9.2 Representation of Standard POS Form on K-map 5-43
5.9.3 Simplification of Standard POS Form using K-map 5-44
5.10 The NOR - NOR Implementation 5-48
5.11 Quine Mc-Cluskey Minimization Technique (Tabular Method) 5-57
5.11.1 Important Definitions 5-58

Module 3

Chapter 6 : Combinational Logic Design **6-1 to 6-95**

6.1 Introduction to Combinational Circuits 6-1
6.1.1 Analysis of a Combinational Circuit 6-2
6.1.2 Design of Combinational Logic using Statements 6-2
6.2 Code Converters 6-8
6.2.1 BCD to Excess 3 Converter 6-8
6.2.2 BCD to Gray Code Converter 6-10
6.2.3 Binary to Gray Code Converter 6-11
6.2.4 Gray to BCD Code Converter 6-13
6.3 Binary Adders and Subtractors 6-14
6.3.1 Types of Binary Adders 6-14
6.3.2 Half Adder 6-15
6.3.3 Full Adder 6-16
6.3.4 Full Adder using Half Adder 6-18
6.3.5 Applications of Full Adder 6-19
6.3.6 Binary Subtractors 6-19
6.3.7 Half Subtractor 6-19
6.3.8 Full Subtractor 6-21
6.3.9 Full Subtractor using Half Subtractors 6-22
6.4 The n-Bit Parallel Adder 6-24
6.4.1 A Four Bit Parallel Adder Using Full Adders 6-24
6.4.2 Propagation Delay in Parallel Adder 6-25
6.4.3 Look Ahead – Carry Adder 6-26
6.4.4 Four Bit Fast Adder with Look-Ahead Carry 6-28

6.4.5 Fast Adder IC 74 LS 83 / 74 LS 283 6-28
6.4.6 Four Bit Binary Adder using IC 7483 6-29
6.4.7 Cascading of Adders 6-29
6.5 n-bit Parallel Subtractor (Use of Adder as Subtractor) 6-30
6.5.1 4 Bit Parallel Subtractor using 2's Complement 6-30
6.5.2 4-Bit Binary Parallel Adder/Subtractor (Using 2's Complement) 6-30
6.5.3 4-Bit Binary Parallel Adder/Subtractor (Using 1's Complement) 6-32
6.6 BCD Adder 6-32
6.6.1 Block Diagram of BCD Adder 6-33
6.6.2 Design of Combinational Circuit 6-33
6.6.3 8-Bit BCD Adder using IC 74283 6-35
6.7 Magnitude Comparators 6-36
6.7.1 1-Bit Binary Comparator 6-36
6.7.2 A 2-Bit Comparator 6-38
6.7.3 A 4-Bit Magnitude Comparator (IC 7485) 6-40
6.7.4 Cascading of IC 7485 6-42
6.8 Arithmetic Logic Unit (ALU) 6-45
6.8.1 Cascading of two 74181 ALUs 6-48
6.9 Multiplexer (Data Selector) 6-48
6.9.1 Necessity of Multiplexers 6-49
6.10 Advantages of Multiplexers 6-49
6.11 Types of Multiplexers 6-49
6.11.1 2 : 1 Multiplexer 6-50
6.11.2 A 4 : 1 Multiplexer 6-50
6.11.3 8 : 1 Multiplexer 6-51
6.11.4 16 : 1 MUX 6-52
6.11.5 Applications of a Multiplexer 6-53
6.12 Study of Different Multiplexer ICs 6-53
6.13 Multiplexer Tree 6-54
6.14 Use of Multiplexers in Combinational Logic Design 6-56
6.14.1 Implementation of a Logical Expression in the Non-standard SOP Form 6-63
6.14.2 Implementing a Standard POS Expression using Multiplexer 6-63
6.14.3 Implementation of Boolean SOP Expression with Don't Care Conditions 6-66
6.15 Demultiplexers 6-74
6.15.1 Demultiplexer Principle 6-75
6.16 Types of Demultiplexers 6-75

Scanned by CamScanner

6.16.1 1 : 2 Demultiplexer 6-75
6.16.2 1 : 4 Demultiplexer 6-76
6.16.3 1 : 8 Demultiplexer 6-78
6.16.4 IC 74138 as 1 : 8 DE-MUX 6-78
6.17 Demultiplexer Tree 6-79
6.17.1 Comparison of Multiplexer and Demultiplexer 6-82
6.17.2 Use of DEMUX in Combinational Logic Design 6-83
6.18 Encoders 6-84
6.18.1 Types of Encoders 6-84
6.19 Priority Encoder 6-85
6.19.1 Octal to Binary Encoder 6-86
6.20 Decoder 6-87
6.20.1 2 to 4 Line Decoder 6-87
6.20.2 Difference Between Decoder and Demultiplexer 6-88
6.20.3 Demultiplexer as Decoder 6-88
6.20.4 3 to 8 Line Decoder 6-89
6.20.5 1 : 8 DEMUX Operated as 3:8 Decoder 6-89
6.20.6 IC 74138 as 3 : 8 Decoder 6-90
6.20.7 Implementation of Boolean Function using Decoder 6-91

Module 4

Chapter 7 : Flip Flops **7-1 to 7-76**

7.1 Introduction 7-1
7.1.1 Clock Signal 7-2
7.1.2 Clock Skew 7-2
7.1.3 Comparison of Combinational and Sequential Circuits 7-2
7.1.4 1-Bit Memory Cell (Basic Bistable Element) 7-3
7.1.5 Latch 7-3
7.2 S-R Latch using NOR Gates 7-4
7.2.1 Operation of S-R Latch 7-5
7.2.2 Symbol and Truth Table of S-R Latch 7-6
7.2.3 Characteristic Equation 7-6
7.2.4 NAND Latch [S-R Latch using NAND Gates] 7-7
7.3 Triggering Methods 7-8
7.3.1 Concept of Level Triggering 7-8
7.3.2 Types of Level Triggered Flip-flops 7-9

7.3.3 Concept of Edge Triggering 7-9
7.3.4 Types of Edge Triggered Flip Flops 7-9
7.4 Gated Latches (Level Triggered SR Flip Flop) 7-10
7.4.1 Types of Level Triggered (Clocked) Flip Flops 7-10
7.5 The Gated S-R Latch (Level Triggered S-R Flip Flop) 7-10
7.5.1 Negative Level Triggered SR Flip Flop 7-11
7.5.2 Disadvantage of S-R latch 7-12
7.5.3 Application of S-R latch 7-12
7.6 The Gated D Latch (Clocked D Flip Flop) 7-12
7.7 Gated JK Latch (Level Triggered JK Flip Flop) 7-13
7.7.1 Race Around Condition in JK Latch 7-14
7.7.2 Difference between Latch and Flip-flop 7-16
7.8 Edge Triggered Flip Flops 7-16
7.8.1 Positive Edge Triggered S-R Flip Flop 7-17
7.8.2 Negative Edge Triggered S - R Flip Flop 7-18
7.9 Edge Triggered D Flip Flop 7-19
7.9.1 Positive Edge Triggered D Flip Flop 7-20
7.9.2 Negative Edge Triggered D Flip Flop 7-21
7.9.3 Applications of D Flipflop 7-21
7.10 Edge Triggered J-K Flip Flop 7-22
7.10.1 Positive Edge Triggered JK Flip Flop 7-22
7.10.2 Characteristic Equation of JK Flip Flop 7-25
7.10.3 How does an Edge Triggered JK FF Avoid Race Around Condition ? 7-25
7.10.4 Negative Edge Triggered JK Flip-Flop 7-26
7.11 Toggle Flip Flop (T Flip Flop) 7-26
7.11.1 Negative Edge Triggered T Flip Flop 7-28
7.11.2 Application of T F/F 7-29
7.12 Master Slave (MS) JK Flip Flop 7-29
7.13 Preset and Clear Inputs 7-31
7.13.1 Synchronous Preset and Clear Inputs 7-32
7.13.2 JK Flip Flop with Preset and Clear Inputs 7-33
7.13.3 Applications of JK Flip Flop 7-33
7.14 Design of a Clocked Flip Flop 7-33
7.14.1 Design Procedure 7-33
7.14.2 Design Example 7-34

Scanned by CamScanner

7.15 Excitation Table of Flip-Flop 7-35
7.15.1 Excitation Table of SR Flip Flops 7-35
7.15.2 Excitation Table of D Flip Flop 7-36
7.15.3 Excitation Table of JK Flip Flop 7-37
7.15.4 Excitation Table of T Flip Flop 7-37
7.16 Conversion of Flip Flops 7-37
7.16.1 Conversion from S-R Flip Flop to D Flip Flop 7-38
7.16.2 Conversion of JK FF to T FF 7-39
7.16.3 SR Flip Flop to T Flip Flop 7-40
7.16.4 SR Flip Flop to JK Flip Flop 7-41
7.16.5 Conversion of D Flip Flop to T Flip Flop 7-42
7.16.6 T Flip Flop to D Flip Flop Conversion 7-43
7.16.7 JK Flip Flop to D Flip Flop Conversion 7-43
7.16.8 JK Flip Flop to SR Flip Flop Conversion 7-44
7.16.9 D FF to SR FF Conversion 7-45
7.16.10 T FF to SR FF Conversion 7-46
7.16.11 Conversion from D FF to JK FF 7-46
7.17 Applications of Flip Flops 7-51
7.17.1 Application of SR Latch for Elimination of Keyboard Debounce 7-51
7.18 Sequential Circuits 7-52
7.18.1 Classification of Sequential Circuits 7-52
7.19 Analysis of Clocked Sequential Circuits 7-53
7.19.1 State Table 7-53
7.19.2 State Diagram 7-54
7.19.3 State Equation 7-54
7.19.4 Flip Flop Input Equations 7-57
7.20 Analysis Procedure for Circuits Containing Flip-Flops 7-58
7.20.1 Analysis with D Flip Flops 7-58
7.20.2 Analysis Using JK Flip-Flops 7-61
7.20.3 Analysis using T Flip Flops 7-63
7.20.4 Models for Synchronous Sequential Circuits 7-65
7.21 Moore Circuit 7-65
7.21.1 Example of Moore Circuit 7-65
7.22 Mealy Circuit 7-66
7.22.1 Example of Mealy Circuit 7-67
7.22.2 Comparison of Moore and Mealy Models 7-67

7.23 Analysis of Clocked Sequential Circuits 7-68

Module 4

Chapter 8 : Counters ..... 8-1 to 8-87

8.1 Introduction 8-1
8.1.1 Types of Counters 8-1
8.1.2 Classification of Counters 8-2
8.2 Asynchronous/Ripple Up Counters 8-4
8.2.1 Two Bit Asynchronous Up Counter using JK Flip-Flops 8-4
8.2.2 3 Bit Asynchronous Up Counter 8-6
8.2.3 4 Bit Asynchronous up Counter 8-7
8.2.4 State Diagram of a Counter 8-8
8.3 Asynchronous Down Counters 8-8
8.3.1 3- Bit Asynchronous Down Counter 8-11
8.4 UP / DOWN Counters 8-11
8.4.1 UP/DOWN Ripple Counters 8-12
8.4.2 3-bit and 4-bit Up Down Ripple Counters 8-14
8.5 Modulus of the Counter (MOD-N Counter) 8-22
8.5.1 Frequency Division Taking Place in Asynchronous Counters 8-23
8.6 Decade (BCD) Ripple Counter 8-25
8.7 Problems Faced by Ripple Counters 8-25
8.7.1 Glitch 8-26
8.7.2 Problem of Propagation Delay 8-26
8.7.3 Maximum Operating Frequency 8-26
8.8 Synchronous Counters 8-27
8.8.1 2-Bit Synchronous up Counter 8-28
8.8.2 3-Bit Synchronous Binary up Counter 8-30
8.8.3 Design of the 3 Bit Synchronous Counter 8-33
8.8.4 Four Bit Synchronous Up Counter 8-34
8.9 Modulo – N Synchronous Counters 8-34
8.9.1 Synchronous Decade Counter 8-42
8.10 UP / DOWN Synchronous Counter 8-42
8.10.1 3-bit Up/Down Synchronous Counter 8-43
8.10.2 Advantages of Synchronous Counter 8-44
8.10.3 Comparison of Synchronous and Asynchronous Counters 8-47
8.11 Lock Out Condition 8-47

Scanned by CamScanner

8.11.1 Bushless Circuit 8-48
8.12 Bush Diagram 8-52
8.13 Applications of Counters 8-57
8.14 State Reduction and Assignments 8-57
8.14.1 Row Elimination Method 8-57
8.14.2 Example on State Reduction 8-58
8.15 State Assignment 8-60
8.16 Design of Clocked Synchronous State Machine using State Diagram 8-62
8.16.1 Design using Unused States 8-64
8.17 Sequence Generator 8-68
8.17.1 Sequence Generator using Shift Register 8-68
8.17.2 Sequence Generator using Counters 8-72
8.18 Sequence Detector 8-83
8.19 University Questions and Answers 8-87

Module 4

Chapter 9 : Shift Registers **9-1 to 9-25**

9.1 Introduction 9-1
9.2 Data Formats 9-1
9.3 Classification of Registers 9-2
9.4 Buffer Registers 9-2
9.5 Shift Registers 9-3
9.5.1 Serial Input Serial Output (Shift Left Mode) 9-4
9.5.2 Serial In Serial Out (Shift Right Mode) 9-6
9.5.3 Applications of Serial Operation 9-8
9.6 Serial In Parallel Out (SIPO) 9-8
9.7 Parallel In Serial Out Mode (PISO) 9-9
9.8 Parallel In Parallel Out (PIPO) 9-10
9.9 Bidirectional Shift Register 9-11
9.9.1 A 3-bit Bidirectional Register using the JK Flip Flops 9-12
9.10 Universal Shift Register 9-13
9.11 Applications of Shift Registers 9-16
9.11.1 Serial to Parallel Converter 9-16
9.11.2 Parallel to Serial Converter 9-16
9.11.3 Ring Counter 9-16
9.11.4 Johnson's Counter (Twisted / Switch Tail Ring Counter) 9-20

9.11.5 Johnson's Counter using D Flip-flops 9-22
9.11.6 Sequence Generator 9-23

Module 5

Chapter 10 : Introduction to VHDL **10-1 to 10-85**

10.1 Introduction to VHDL 10-1
10.1.1 Features of VHDL 10-1
10.2 Structure of VHDL Module / VHDL Design Units 10-2
10.2.1 Package 10-2
10.2.2 Entity 10-3
10.2.2.1 Entity Declaration 10-3
10.2.2.2 Port Modes in VHDL 10-3
10.2.3 Architecture Body 10-4
10.2.3.1 Configuration Declaration 10-5
10.2.4 Important Points to Remember while Designing any Module using VHDL 10-6
10.3 Library 10-6
10.4 Identifiers 10-7
10.5 VHDL Operators 10-7
10.5.1 Logical Operators 10-7
10.5.2 Relational Operators 10-8
10.5.3 Adding Operators 10-8
10.5.4 Multiplying Operators 10-8
10.5.5 Miscellaneous Operators 10-9
10.5.6 Precedence of Operators in VHDL 10-9
10.6 VHDL Data Objects 10-10
10.6.1 Classes of Data Objects 10-10
10.6.1.1 Signals 10-10
10.6.1.2 Variable 10-10
10.6.1.3 Constant 10-10
10.6.1.4 File 10-11
10.6.2 Comparison between Signal and Variable 10-11
10.7 VHDL Data Types 10-11
10.8 Classification of Data Types 10-11
10.8.1 Scalar 10-12
10.8.1.1 Enumeration Types 10-12
10.8.1.2 Integer Types 10-12
..... 10-13

Scanned by CamScanner

10.8.1.3 Physical Types 10-13
10.8.1.4 Real (Floating Point) Type 10-13
10.8.2 Composite Types 10-13
10.8.2.1 Array Types 10-14
10.8.2.2 Record Types 10-14
10.8.3 Access Types 10-14
10.8.4 File Types 10-14
10.9 Modeling Styles in VHDL 10-14
10.9.1 Structural Modeling / Gate Level Modeling 10-15
10.9.2 Dataflow Modeling 10-16
10.9.3 Behavioral Style of Modeling 10-17
10.9.4 Mixed Style of Modeling 10-18
10.9.5 Comparison of Modeling Styles 10-19
10.10 VHDL Statements 10-19
10.10.1 Variable Assignment Statements 10-20
10.10.2 Signal Assignment Statements 10-21
10.11 Concurrent Statements 10-21
10.11.1 Process Statement 10-21
10.11.2 Block Statement 10-21
10.11.3 Generate Statement 10-22
10.12 Sequential Statements 10-23
10.12.1 With Select when Statement 10-23
10.12.2 Wait Statement 10-24
10.12.3 Conditional Statements 10-25
10.12.4 If Statement 10-25
10.12.5 Case Statements 10-26
10.12.6 Comparison between If and Case Statements 10-27
10.12.7 Loop Statements 10-27
10.12.8 Null Statements 10-28
10.12.9 Exit Statements 10-29
10.12.10 Next Statement 10-29
10.12.11 Assertion Statement 10-30
10.13 Comparison between Concurrent and Sequential Statements 10-30
10.14 VHDL Attributes 10-30
10.14.1 Types of Attributes 10-30
10.14.2 User Defined Attributes 10-30

10.14.3 Pre-defined Attributes 10-31
10.14.3.1 Value Attributes 10-31
10.14.3.2 Function Attributes 10-31
10.14.3.3 Signal Attributes 10-32
10.14.3.4 Type Attributes 10-33
10.14.3.5 Range Attributes 10-33
10.15 VHDL For Combinational and Sequential Circuits 10-33
10.16 VHDL For Basic Logic Gates 10-33
10.16.1 Implementation of NOT Gate 10-33
10.16.2 Implementation of AND Gate 10-34
10.16.3 Implementation of OR Gate 10-35
10.16.4 Implementation of NAND Gate 10-35
10.16.5 Implementation of NOR Gate 10-36
10.16.6 Implementation of XOR Gate 10-37
10.16.7 Implementation of XNOR Gate 10-38
10.17 VHDL For Combinational Circuits 10-38
10.17.1 Implementation of Adders 10-38
10.17.1.1 Implementation of Half Adder 10-38
10.17.1.2 Implementation of Full Adder 10-40
10.17.1.3 Implementation of Full Adder Using Half Adders 10-42
10.17.1.4 Implementation of 4-Bit Full Adder 10-43
10.17.2 Implementation of Subtractor 10-44
10.17.2.1 Implementation of Half Subtractor 10-44
10.17.2.2 Implementation of Full Subtractor 10-45
10.17.2.3 Implementation of 4 bit Adder-Subtractor 10-47
10.17.3 Implementation of Comparator 10-48
10.17.4 Implementation of Multiplexer 10-49
10.17.4.1 Implementation of 4 : 1 Multiplexer 10-49
10.17.4.2 Implementation of 8 : 1 MUX 10-52
10.17.5 Implementation of De-Multiplexers 10-55
10.17.5.1 Implementation of 1 : 4 De-Multiplexer 10-55
10.17.6 Implementation of Encoders 10-56
10.17.6.1 Implementation of 8 : 3 (Octal to Binary) Encoder 10-56
10.17.6.2 Implementation of 4 : 2 Priority Encoders using If-Then-Else Statement 10-57
10.17.7 Implementation of Decoders 10-58
10.17.7.1 VHDL Code for 2 : 4 Decoder 10-58

Scanned by CamScanner

10.17.7.2 Implementation of 3 : 8 Binary Decoder 10-59
10.17.7.3 VHDL Code for BCD to 7 Segment Decoder 10-60
10.17.8 Implementation of Parity Circuits 10-61
10.17.8.1 VHDL Code for Even Parity Generator 10-62
10.17.8.2 VHDL Code for 8 bit Parity Generator using for Loop and Generic Statement . 10-62
10.17.8.3 VHDL Code for 4-bit Parity Checker 10-62
10.17.9 Implementation of Code Converters 10-63
10.17.9.1 VHDL Code for Binary to Gray Code Converter 10-63
10.17.9.2 VHDL Code for BCD to Excess-3 Converter 10-65
10.17.9.3 VHDL Code for Gray to Binary Code Converter 10-66
10.18 VHDL for Sequential Circuits 10-68
10.18.1 Implementation of Flip-Flops 10-68
10.18.1.1 VHDL Code for D Flip-Flop 10-68
10.18.1.2 Implementation of JK Flip Flop 10-69
10.18.1.3 Implementation of T Flip Flop 10-70
10.18.1.4 Implementation of SR Latch 10-72
10.18.1.5 Implementation of SR Flip-Flop 10-72
10.18.2 Implementation of Asynchronous Counters 10-73
10.18.2.1 VHDL code for 4-bit Binary Up Counter with Asynchronous Clear 10-73
10.18.2.2 VHDL Code for 4-Bit Binary Down Counter with Asynchronous Preset 10-75
10.18.2.3 VHDL Code for 4-Bit Binary Up-Down Counter 10-76
10.18.2.4 VHDL Code for 4-Bit BCD Up Asynchronous Counter 10-76
10.18.2.5 VHDL Code for BCD Up-Down Asynchronous Counter 10-77
10.18.3 Implementation of Synchronous Counters 10-78
10.18.3.1 VHDL Code for Synchronous Mod-6 Counter 10-78
10.18.3.2 VHDL Code for 4-Bit Synchronous Counter 10-79
10.18.3.3 VHDL code for 3-Bit Synchronous Up/Down Counter With Asynchronous Clear Input 10-80
10.18.4 Implementation of Shift Registers 10-80
10.18.4.1 VHDL Code for 4-bit Buffer Register 10-80
10.18.4.2 VHDL Code For Serial Input Serial Output Shift Register 10-82
10.18.4.3 VHDL Code for Serial In Parallel Out Shift Register 10-82
10.18.4.4 VHDL Code for Parallel In Serial Out Shift Register 10-83
10.18.4.5 VHDL Code for Parallel In Parallel Out Shift Register 10-83
.19 Advantages of VHDL 10-84
10.19.1 Disadvantages of VHDL 10-84

10.19.2 Applications of VHDL 10-84

Module 6

Chapter 11 : Digital Logic Families ..... 11-1 to 11-42
11.1 Introduction 11-1
11.1.1 Classification Based on Circuit Complexity 11-1
11.2 Classification of Logic Families 11-2
11.2.1 Classification Based on Devices Used 11-2
11.3 Characteristics of Digital ICs 11-3
11.3.1 Voltage and Current Parameters 11-3
11.3.2 Fan-in and Fan-out 11-5
11.3.3 Noise Margin 11-5
11.3.4 Propagation Delay (Speed of Operation) 11-5
11.3.5 Power Dissipation 11-6
11.3.6 Operating Temperature 11-7
11.3.7 Figure of Merit (Speed Power Product SPP) 11-7
11.3.8 Invalid Voltage Levels 11-7
11.3.9 Current Sourcing and Current Sinking 11-7
11.4 TTL Logic 11-8
11.4.1 The Multiple Emitter Transistor 11-8
11.4.2 Two Input TTL-NAND Gate (Totempole Output) 11-8
11.4.3 Totem-pole (Active Pull up) Output Stage 11-11
11.4.4 Unconnected Inputs 11-13
11.4.5 Clamping Diodes 11-13
11.4.6 TTL NOR Gate 11-14
11.4.7 5400 Series 11-15
11.5 Sourcing and Sinking in TTL 11-15
11.5.1 Current Sinking Action 11-15
11.5.2 Current Sourcing Action 11-16
11.5.3 Three Input TTL NAND Gate 11-16
11.6 Wired AND Connection (TTL) 11-17
11.7 Open Collector Outputs (TTL) 11-18
11.7.1 Disadvantages of Open Collector Output 11-19
11.7.2 Advantage 11-19
11.7.3 Wired ANDing 11-20
11.7.4 Comparison of Totem-pole and Open Collector Outputs 11-21
11.8 Tristate (Three state) TTL Devices 11-21
11.8.1 Advantages of Tristate 11-22
11.8.2 Tristate Buffers 11-22

Scanned by CamScanner

11.8.3 Applications of Tristate Buffers (Bus Organization) 11-23
11.8.4 A TRI-STATE Inverter 11-24
11.9 Standard TTL Characteristics 11-25
11.9.1 Advantages of TTL 11-26
11.9.2 Disadvantages of TTL 11-26
11.10 MOS - Logic Family 11-26
11.11 CMOS Logic 11-26
11.11.1 CMOS Inverter 11-26
11.11.2 CMOS NOR Gate 11-28
11.11.3 CMOS NAND Gate 11-30
11.11.4 CMOS Series 11-32
11.12 CMOS Characteristics 11-32
11.12.1 Power Supply Voltage 11-32
11.12.2 Logic Voltage Levels 11-32
11.12.3 Noise Margins 11-33
11.12.4 Power Dissipation 11-33
11.12.5 Fan Out 11-33
11.12.6 Switching Speed 11-34
11.12.7 Unconnected Inputs 11-34
11.12.8 Static Sensitivity 11-34
11.12.9 Latch Ups 11-34
11.13 Advantages of CMOS 11-34
11.14 Disadvantages of CMOS 11-35
11.15 Tristate Logic Output 11-35
11.16 Interfacing 11-35
11.16.1 TTL to CMOS Interfacing 11-35
11.16.2 TTL Driving High Voltage CMOS 11-36
11.16.3 Interfacing using Level-Shifter (TTL to High Voltage CMOS) 11-37
11.16.4 CMOS to TTL Interface 11-37
11.16.5 CMOS Driving TTL in the HIGH State 11-38
11.16.6 CMOS Driving TTL in the LOW State 11-38
11.16.7 High Voltage CMOS Driving TTL 11-39
11.16.8 Interfacing CMOS with ECL and TTL 11-39
11.17 Comparison of CMOS and TTL 11-39
11.17.1 Comparison of CMOS, TTL and ECL 11-40

CHAPTER 1

Module 1

Number Systems

Syllabus :

Introduction to number system and conversions : Binary, Octal, Decimal and Hexadecimal number systems.

1.1 Introduction :

- In the modern world of electronics, the term "digital" is generally associated with a computer.
- This is because, the term "digital" is derived from the way computers perform operation, by counting digits.
- For many years, the main application of digital electronics was only in the computer systems.
- But today, the digital electronics is used in many other applications such as : TV, Radar, Military systems, Medical equipments, Communication systems, Industrial process control and Consumer electronics.

1.1.1 Signals :

We can define "signal" as a physical quantity, which contains some information and which is a function of one or more independent variables.

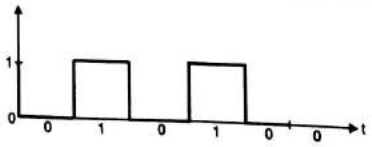
1.1.2 Types of Signals :

The signals can be of two types :

1. Analog signals 2. Digital signals

1.1.3 Digital Signals :

- A **digital signal** is defined as the signal which has only a finite number of distinct values. Digital signals are not continuous signal. They are discrete signals as shown in Fig. 1.1.1.
- **Binary signal :** If a digital signal has only two distinct values, i.e. 0 and 1 then it is called as a binary signal.



(B-2163) **Fig. 1.1.1 : Binary signal (Digital signal)**

Scanned by CamScanner

- **Octal signal :** A digital signal having eight distinct values is called as an octal signal.
- **Hexadecimal signal :** A digital signal having sixteen distinct values is called as the hexadecimal number.

Type of digital signal	Number of distinct values
Binary	2
Octal	8
Hexadecimal	16

1.2 Digital Systems :

- These are the circuits that operate on the digital signals to produce digital signals.
- The input signal to a digital system is digital and its output signal is also digital.

(B-1706) **Fig. 1.2.1 : Digital circuit**

Examples of digital systems :

The examples of digital circuits are adders, subtractors, registers, flip-flops, counters, microprocessors, digital calculators, computers etc.

1.3 Binary Logic and Logic Levels :

A logic statement, is defined as a statement which is true if some condition is satisfied and false if that condition is not satisfied. For example, a bulb turns ON, if we close the switch, otherwise it is OFF.

1.3.1 Positive Logic :

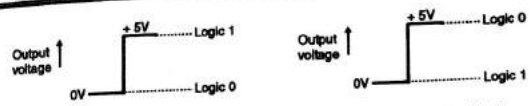
- A "LOW" voltage level represents "logic 0" state and a comparatively "HIGH" output voltage level represents "logic 1" state, as shown in Fig. 1.3.1(a).
- For example, 0 Volt represent a logic 0 state and + 5 V represent logic 1. This is called as "positive logic".

Positive Logic : Logic 0 (LOW) = 0 V.
Logic 1 (HIGH) = + 5 V.

1.3.2 Negative Logic :

- A "LOW" voltage level represents "logic 1" state and a "HIGH" output voltage level represents "logic 0" state, as shown in Fig. 1.3.1(b).
- For example, 0 Volt represent a "logic 1" state and + 5 V represent "logic 0" state. This is called as "negative logic".

Negative Logic : Logic 0 (LOW) = + 5 V
Logic 1 (HIGH) = 0 V.



(a) Positive logic (B-440) (b) Negative logic

(B-439) **Fig. 1.3.1**

Note : In this chapter, we are going to consider only the positive logic. Also we will assume the logic 0 level corresponds to 0 Volt and logic 1 level corresponds to + 5 V.

1.4 Number Systems :

- A number system defines a set of values used to represent a quantity. We talk about the number of people attending class, the number of modules taken per student and also use numbers to represent grades obtained by students in tests.
- Quantifying values and items in relation to each other is helpful for us to make sense of our environment. We do this at an early age; figuring out if we have more toys to play with, more presents, more dollies and so on.
- The study of number systems is not just limited to computers. We apply numbers every day and knowing how numbers work will give us an insight into how a computer manipulates and stores numbers.
- Mankind through the ages has used signs or symbols to represent numbers.

1.4.1 Important Definitions Related to All Numbering Systems :

All the numbering systems have a few common elements as follows :

Radix or Base :

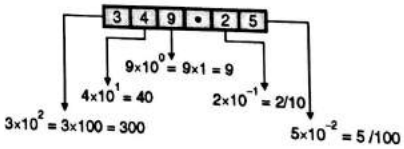
1. The number of values that a digit (one character) can have is equal to the base of the system. It is also called as the Radix of the system.

   For example for a decimal system, the base is "10" because every digit can have 10 distinct values. (0, 1, 2, ..., 9).

2. The largest value of a digit is always one less than the base :

   For example, the largest digit in a decimal system is 9. (One less than the base 10).

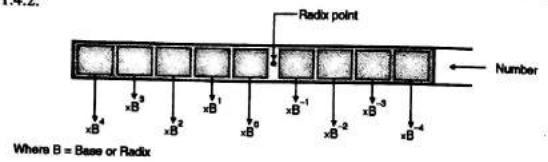
3. The position (place) of every digit represents a different multiple of base. i.e. the numbers have positional importance. For example consider the decimal number $(349.25)_{10}$ shown in Fig. 1.4.1.



(C-5) **Fig. 1.4.1 : Numbers have positional importance**

Scanned by CamScanner

- Hence we can implement a common rule for all the numbering systems as follows. For a general number, we have to multiply each of digit by some power of **base (B) or radix** as shown in Fig. 1.4.2.



(C-6) **Fig. 1.4.2**

4. **Column numbers :**
   - The column number is the number assigned to the digits placed in relation with the decimal point.
   - Column numbers to the left of the decimal number start with 0 and go up (0, 1, 2, ...) as shown in Fig. 1.4.2.
   - Column numbers to the right of the decimal number start from – 1 and become more and more negative (– 1, – 2, – 3, – 4, ...) as shown in Fig. 1.4.2.

1.4.2 Various Numbering Systems :

Various numbering systems used in practice and their bases are as shown in Table 1.4.1.

Table 1.4.1 : Various number systems and their bases

Name of number system	Base
Binary	2
Octal	8
Decimal	10
Duodecimal	12
Hexadecimal	16

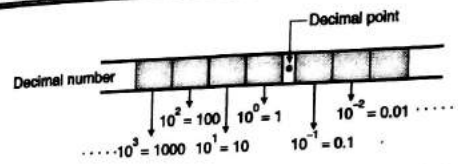
1.5 The Decimal Number System :

- The number system we have been taught to use since school is called the decimal numbering system.
- We are all familiar with counting and mathematics that uses this system.
- Looking at its make up will help us to understand other numbering systems.

1.5.1 Characteristics of a Decimal System :

Some of the important characteristics of a decimal system are :

1. It uses the base of 10.
2. The largest value of a digit is 9.
3. Each place (column number) represents a different multiple of 10. These multiples are also called as **weighted values.** The weighted values of each position are as shown in Fig. 1.5.1.



(C-8) **Fig. 1.5.1 : Positions and corresponding weighted values for a decimal system**

1.5.2 Most Significant Digit (MSD) :

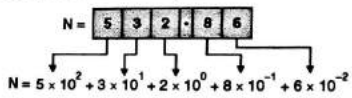
The leftmost digit having the highest weight is called as the most significant digit of a number.

1.5.3 Least Significant Digit (LSD) :

The rightmost digit having the lowest weight is called as the least significant digit of a number.

Ex. 1.5.1 : Represent the decimal number 532.86 in terms of powers of 10.

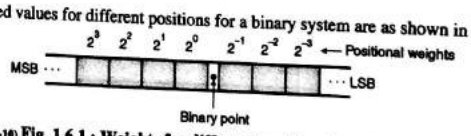
Soln. : The required representation is shown in Fig. P. 1.5.1.



(C-9) **Fig. P. 1.5.1**

1.6 The Binary Number System :

- Most modern computer systems use the binary logic for their operation. A computer cannot operate on the decimal number system.
- A binary number system uses only two digits namely 0 and 1.
- The binary number system works like the decimal number system except one change. **It uses the base 2.**
- Hence the largest value of a digit is 1 and the number of values a digit can assume is two i.e. 0 and 1.
- The weighted values for different positions for a binary system are as shown in Fig. 1.6.1.



(C-10) **Fig. 1.6.1 : Weights for different positions for a binary system**

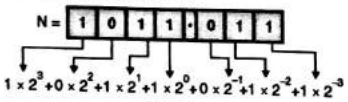
- The binary digits (0 and 1) are also called as **bits.** Thus binary system is a two bit system.
- The leftmost bit in a given binary number with the highest weight is called as **Most Significant Bit (MSB)** whereas the rightmost bit in a given number with the lowest weight is called as **Least Significant Bit (LSB).**

Scanned by CamScanner

Ex. 1.6.1 : Express the binary number 1011.011 in terms of powers of 2.

Soln. :

Step 1 : Express the given number in powers of 2 :



(C-11) **Fig. P. 1.6.1**

1.6.1 Binary Numbers from $(0)_{10}$ to $(15)_{10}$:

The binary numbers representing the decimal numbers from 0 to 15 are shown in the Table 1.6.1.

Table 1.6.1 : Binary numbers upto $(15)_{10}$

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

1.6.2 Binary Number Formats :

- We typically write binary numbers as a sequence of bits (bits is short for binary digits). We have defined boundaries for these bits. These boundaries are :

Table 1.6.2 : Binary number formats

Name	Size (bits)	Example
Bit	1	1
Nibble	4	0101
Byte	8	0000 0101
Word	16	0000 0000 0000 0101
Double Word	32	0000 0000 0000 0000 0000 0000 0000 0101

1.6.3 Disadvantages of the Binary System :

- The binary number system has an important drawback. It requires a very long string of 1's and 0's to represent a decimal number. For example,

$$(128)_{10} = (10000000)_2$$

- So we require 8 bits to represent $(128)_{10}$. Imagine what will happen if $(26275)_{10}$ is to be converted.
- To overcome this the other numbering systems were developed. Let us discuss them one by one.

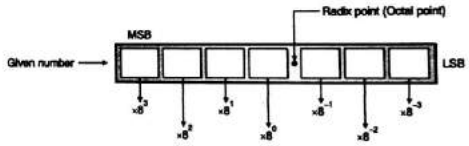
1.7 Octal Number System :

The important features of the octal number systems are as follows :

1. **Base :** The Base used for octal number system is 8.
2. **The number of values assumed by each digit :** Each digit in the octal system will assume 8 different values from 0 to 7 (0, 1, 2,....., 6, 7).
3. **The largest value of a digit :** The largest value of a digit in the octal system will be 7. That means the octal number higher than 7 will not be 8, instead of that it will be 10. Table 1.7.1 gives you a clear idea about this.

Table 1.7.1 : Octal numbers

0	10	20	30	40
1	11	21	31	41
2	12	22	32	42
3	13	23	33	43
4	14	24	34	44
5	15	25	35	45
6	16	26	36	46
7	17	27	37	47



(C-13) **Fig. 1.7.1 : Weights for different positions for an octal system**

4. Each digit has a different multiple of base. This is as shown in Fig. 1.7.1.

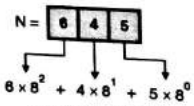
1.7.1 Applications of Octal System :

1. In order to represent large numbers, we will need a huge string of 0's and 1's if the binary system is used.
2. So the octal system can be used for reducing the size of the large binary number.
3. But it is important to note that the digital circuits and computers internally work strictly with the binary structure, and not with the octal structure.
4. The octal structure is used only as the convenience of the user.

Ex. 1.7.1 : Represent the octal number 645 in power of 8.

Soln. :

Step 1 : Representation in power of 8 :



(C-14) **Fig. P. 1.7.1**

Scanned by CamScanner

1.8 Hexadecimal Number System :

The important features of a hexadecimal number system are as follows :

- **Base** : The base of hexadecimal system is 16.
- **Number of values assumed by each digit** : The number of values assumed by each digit is 16. The values include digits 0 through 9 and letters A, B, C, D, E, F. Hence the sixteen possible values are : 0 1 2 3 4 5 6 7 8 9 A B C D E F
- 0 represents the least significant digit whereas F represents the most significant digit.
- The base of 16 needs 16 digits. Hence it borrows 0 through 9 but needs another 6. For these the first six letters A, B, C, D, E, F are used. Here A represents 10, B represents 11 and so on. The hexadecimal digits and their values are as shown in Table 1.8.1.

Table 1.8.1 : Hexadecimal digits and their values

Hexadecimal digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 1.8.2 : Hexadecimal numbers

0	10	20	30	.....	90	A0	.....	F0
1	11	21	31	.....	91	A1	.....	F1
2	12	22	32	.....	92	A2	.....	F2
3	13	23	33	.....	93	A3	.....	F3
4	14	24	34	.....	94	A4	.....	F4
. . . .	. . . .	. . . .	. . . .		. . . .	. . . .		. . . .
D	1D	2D	3D	.....	9D	AD	.....	FD
E	1E	2E	3E	.....	9E	AE	.....	FE
F	1F	2F	3F	.....	9F	AF	.....	FF

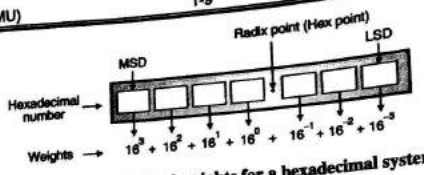
When dealing with large values, binary numbers quickly become too unwidely. The hexadecimal (base 16) numbering system solves this problem. Hexadecimal numbers offer the two features :

1. Hex numbers are very compact.
2. It is easy to convert from hex to binary and binary to hex.

Largest value of a digit : The largest value of a digit in the hexadecimal number system is 15 and it is represented by F. The hexadecimal number higher than F will be 10. Table 1.8.2 gives you a clear idea about hexadecimal numbers.

The largest two digit hexadecimal number is FF which corresponds to 255 decimal. The next higher number after FF is 100.

Positional weights : The positional weights for a hexadecimal number to the left and right of decimal point are as shown in Fig. 1.8.1.



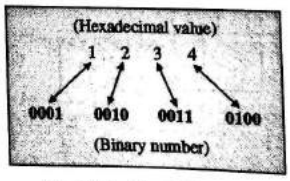
(C-15) **Fig. 1.8.1 : Positional weights for a hexadecimal system**

1.9 Relation between Binary, Decimal, Octal and Hexadecimal Numbers :

- Hexadecimal numbers are compact and easy to read. It is very easy to convert numbers from binary system to hexadecimal system and vice-versa, every nibble (4 bits) can be converted to a hexadecimal digit using this Table 1.9.1.

Table 1.9.1 : Relation between decimal, binary, octal and hexadecimal number

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



(C-16)

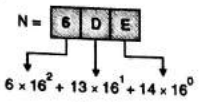
Fig. 1.9.1 : Hex to binary conversion

Ex. 1.9.1 : Represent the hexadecimal number 6DE in the powers of 16.

Scanned by CamScanner

Soln. :

Representation in the powers of 16 :



(C-17) **Fig. P. 1.9.1**

1.10 Conversion of Number Systems :

- The conversion of a number in base "r" to decimal is done by expanding the given number in a power series and adding all the terms.
- In the subsequent sections we are going to present a general procedure for decimal to any base (radix) conversion.
- If the given number includes the radix point, then it is necessary to separate the number into an integer part and a fraction part. Then each part should be converted by considering separately.

1.11 Conversions Related to Decimal System :

In this section we will perform the following conversions related to the decimal system :

1. Decimal to other systems. 2. Other systems to decimal.

1.11.1 Conversion from Radix r to Decimal :

- The general procedure for conversion from binary to decimal is as given below :

Steps to be followed :

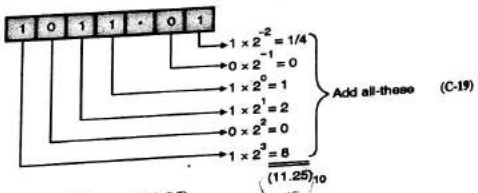
Step 1 : Note down the given number.
Step 2 : Write down the weights corresponding to different positions.
Step 3 : Multiply each digit in the given number with the corresponding weight to obtain product numbers.
Step 4 : Add all the product numbers to get the decimal equivalent.

- The following example demonstrates the conversion of a binary, octal and hex number to its decimal equivalent.

Ex. 1.11.1 : Convert the binary number 1 0 1 1 . 0 1 into its decimal equivalent.

Soln. :

Steps 1, 2 and 3 :



(C-19)

Step 4 : Addition :

$\therefore \quad (1011.01)_2 = (11.25)_{10}$ ...Ans.

Ex. 1.11.2 : Convert the octal number $(314)_8$ into its decimal equivalent.

Soln. :

Step 1 : Get the octal number :
Step 2 : Write corresponding weights :
Step 3 : Multiply (column wise) :
Step 4 : Add contents of row-3 :

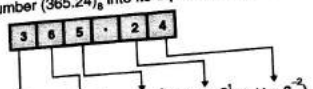
3	1	4
8^2	8^1	8^0
192	8	4

$192 + 8 + 4 = 204$

$\therefore \quad (314)_8 = (204)_{10}$ ...Ans.

Ex. 1.11.3 : Convert the octal number $(365.24)_8$ into its equivalent decimal number. (C-20)

Soln. :

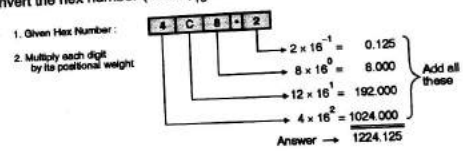


Decimal number, $D = (3 \times 8^2) + (6 \times 8^1) + (5 \times 8^0) + (2 \times 8^{-1}) + (4 \times 8^{-2})$
$= 192 + 48 + 5 + 0.25 + 0.0625$

$\therefore \; D = (245.3125)_{10}$

Ex. 1.11.4 : Convert the hex number $(4C8.2)_{16}$ into its equivalent decimal number.

Soln. :



$\therefore (4C8.2)_{16} = (1224.125)_{10}$

(C-21) **Fig. P. 1.11.4**

1.11.2 Conversion from Decimal to Other Systems :

- If the given decimal number consists of a decimal point, then we have to first separate out the integer and fractional parts.
- Then convert them separately to the desired radix, and combine the converted parts to obtain the complete converted number.
- The procedures for converting the integer part and the fractional part are completely different from each other.

1.11.2.1 Successive Division for Integer Part Conversion :

- Follow the procedure given below to convert the integer part of the given decimal number into any radix "r".

Steps to be followed :

Step 1 : Divide the integer part of given decimal number by the base and note down the remainder.
Step 2 : Continue to divide the quotient by the base (r) until there is nothing left, noting the remainders from each step.
Step 3 : List the remainder values in reverse order from the bottom to top to find the equivalent.

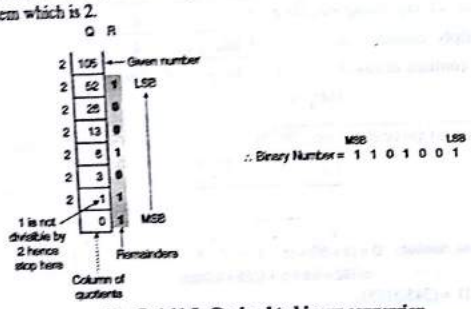
- This procedure will be best understood by going through the following illustrative examples.

Scanned by CamScanner

Ex. 1.11.5 : Convert $(105)_{10}$ to the equivalent binary number.

Soln. :

Refer Fig. P. 1.11.5 which shows a simpler method. We divide the given number by the radix or base of binary system which is 2.



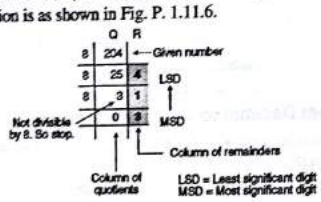
(C-3650) **Fig. P. 1.11.5 : Decimal to binary conversion**

Thus $(105)_{10} = (1101001)_2$ **...Ans.**

Ex. 1.11.6 : Convert $(204)_{10}$ into its equivalent octal number.

Soln. :

- We divide the decimal number by the radix (base) of octal system which is 8.
- The required conversion is as shown in Fig. P. 1.11.6.



$\therefore (204)_{10} = (314)_8$ **...Ans.**

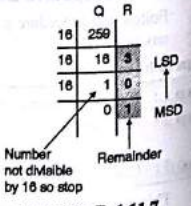
(C-24) **Fig. P. 1.11.6 : Decimal to octal conversion**

Ex. 1.11.7 : Convert the decimal number 259 into its hex equivalent.

Soln. : The conversion takes place as follows :

The base is 16 so we divide the given number by 16.

$\therefore (259)_{10} = (103)_{16}$ **...Ans.**

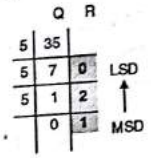


(C-25(a)) **Fig. P. 1.11.7**

Ex. 1.11.8 : Convert the decimal number 35 into radix 5 equivalent.

Soln. :

The base is 5, so we will divide the given number by 5 as shown in Fig. P. 1.11.8.



(C-3341) **Fig. P. 1.11.8 : Decimal to radix 5**

Hence the answer is : $(35)_{10} = (120)_5$. **...Ans.**

1.11.2.2 Successive Multiplication for Fractional Part Conversion :

- Now let us see how to convert the fractional part of a decimal number into any other radix number.
- The general procedure for such a conversion is as follows :

Steps to be followed :

Step 1 : Multiply the given fractional decimal number by the base (radix) r.

Step 2 : Note down the carry generated in this multiplication as MSD.

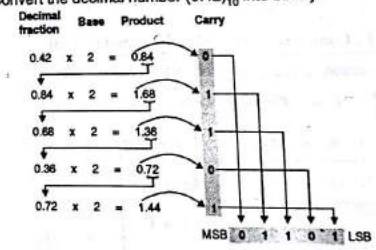
Step 3 : Multiply only the fractional number of the product in step 2 by the base, and note down the carry as the next bit to MSD.

Step 4 : Repeat steps 2 and 3 upto the end. The last carry will represent the LSD of equivalent i.e. converted number.

Let us solve some examples to understand fraction conversion clearly.

Ex. 1.11.9 : Convert the decimal number $(0.42)_{10}$ into binary.

Soln. :



(C-27)

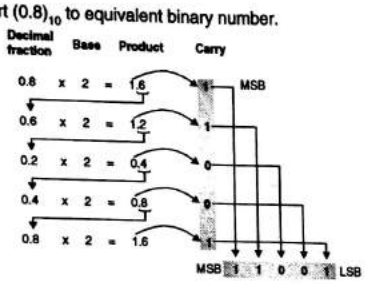
So $(0.42)_{10} = (0.01101)_2$ **...Ans.**

Note : We could have continued further. But the conversion is generally carried out only upto 5 digits.

Scanned by CamScanner

1.11.10 : Convert $(0.8)_{10}$ to equivalent binary number.

Soln. :

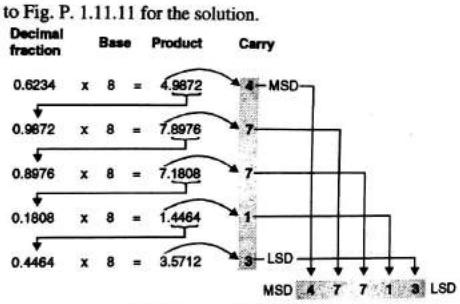


(C-28)

$\therefore \ (0.8)_{10} = (0.11001)_2$ ...Ans.

1.11.11 : Convert $(0.6234)_{10}$ into its equivalent octal number.

Soln. : Refer to Fig. P. 1.11.11 for the solution.



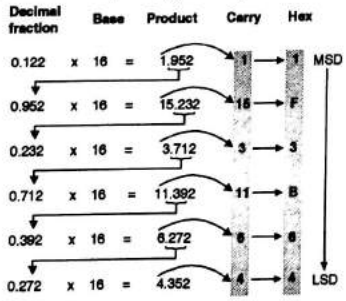
(C-29)

$\therefore \ (0.6234)_{10} = (0.47713)_8$ ...Ans.

Fig. P. 1.11.11 : Conversion of fractional decimal to octal

1.11.12 : Convert the decimal fraction $(0.122)_{10}$ to its equivalent hex number.

Soln. :



(C-30)

$\therefore \ (0.122)_{10} = (0.1F3B64)_{16}$ ...Ans.

1.11.2.3 Conversion of Mixed Decimal Number to Any Other Radix :

- We have seen the conversion of the integer and fractional parts of a decimal number into desired radix number separately.
- Now let us see the conversion of mixed decimal number containing the integer and fractional parts into any other radix number.
- Follow the procedure given below for such a conversion.

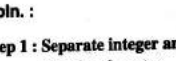
Steps to be followed :

Step 1 : Separate the integer and fractional parts of the given decimal number.
Step 2 : Convert the integer part into desired radix.
Step 3 : Convert the fractional part into desired radix.
Step 4 : Combine the results of steps 2 and 3 to get the final answer.

Ex. 1.11.13 : Convert $(85.63)_{10}$ into its equivalent binary number.

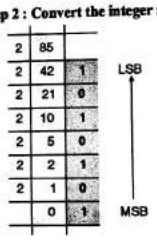
Soln. :

Step 1 : Separate integer and fractional parts :



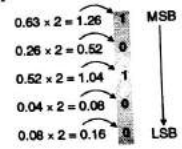
(C-31)

Step 2 : Convert the integer :



$\therefore \ (85)_{10} = (1\,0\,1\,0\,1\,0\,1)_2$

Step 3 : Convert the fractional part :



$\therefore \ (0.63)_{10} = (0.10100)_2$

(C-32)

Step 4 : Combine the results of steps 2 and 3 :

$\therefore \ (85.63)_{10} = (1010101.10100)_2$ ...Ans.

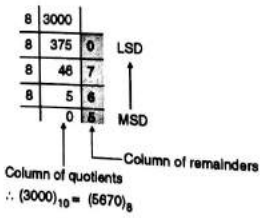
Ex. 1.11.14 : Convert $(3000.45)_{10}$ into its equivalent octal number.

Soln. :

Step 1 : Separate the integer and fractional parts :

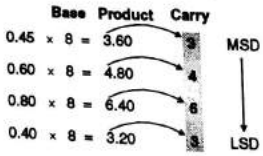
Integer part = 3000 and fractional part = 0.45.

Step 2 : Convert the integer part :



$\therefore \ (3000)_{10} = (5670)_8$

Step 3 : Convert the fraction part :



$\therefore \ (0.45)_{10} = (0.3463)_8$

(C-33) **Fig. P. 1.11.14**

Scanned by CamScanner

Step 4 : Combine the results of steps 2 and 3 :

Combining the results of steps 2 and 3 we get the answer.

$$\therefore \quad (3000.45)_{10} = (5670.3463)_8$$

Ex. 1.11.15 : Convert $(2003.31)_{10}$ into its equivalent hex number.

Soln. :

Step 1 : Separate the integer and fractional parts :

2003.31

Integer → Fractional

(C-34)

Step 2 : Convert integer part :

			Hex
16	2003		
16	125	3	3
16	7	13	D
	0	7	7

LSD ↑ MSD

$\therefore (2003)_{10} = (7D3)_{16}$ (C-35)

Step 3 : Convert the fractional part into hex :

Decimal fraction	×	Base	=	Product	Carry Decimal	Carry Hex
0.31	×	16	=	4.96	4	4
0.96	×	16	=	15.36	15	F
0.36	×	16	=	5.76	5	5
0.76	×	16	=	12.16	12	C
0.16	×	16	=	2.56	2	2

MSD ↓ LSD

$\therefore \quad (0.31)_{10} = (0.4F5C2)_{16}$ (C-36)

Step 4 : Combine the results of steps 2 and 3 :

Combining the results of steps 2 and 3 we get,

$$\therefore \quad (2003.31)_{10} = (7D3.4F5C2)_{16}$$ ...Ans.

Ex. 1.11.16 : Convert $(54.45)_{10}$ into binary, base 4, octal and hexadecimal number systems.

May 12, 4 Marks

Soln. :

Step 1 : Convert decimal to binary :

Separate integer and fractional parts :

54 · 45

Integer Fractional

(C-3291)

Convert the integer :

2	54	
2	27	0
2	13	1
2	6	1
2	3	0
2	1	1
	0	1

LSB ↑ MSB

$(54)_{10} = (110110)_2$

Convert the fractional part :

Decimal fraction	Base	Product	Carry
0.45 × 2 =		0.9	0
0.9 × 2 =		1.8	1
0.8 × 2 =		1.6	1
0.6 × 2 =		1.2	1
0.2 × 2 =		0.4	0
0.4 × 2 =		0.8	0

MSB ↓ LSB

(C-3292)

$\therefore (0.45)_{10} = (0.011100)_2$

$$\therefore \quad (54.45)_{10} = (110110.011100)_2$$ ...Ans.

Step 2 : Convert decimal to base 4 :

Convert the integer part :

4	54	
4	13	2
4	3	1
	0	3

LSD ↑ MSD

$\therefore (54)_{10} = (312)_4$

Convert the fractional part :

0.45 × 4 = 1.8 → 1

0.8 × 4 = 3.2 → 3

0.2 × 4 = 0.8 → 0

0.8 × 4 = 3.2 → 3

MSD ↓ LSD

(C-3293)

$\therefore (0.45)_{10} = (0.1303)_4$

$$\therefore \quad (54.45)_{10} = (312.1303)_4$$ ...Ans.

Step 3 : Convert decimal to octal :

Convert the obtained binary number to octal :

Binary number : $(110110.011100)_2$

Binary number : 1 1 0 1 1 0 · 0 1 1 1 0 0 ← Group of 3 bits

Octal number : 6 6 · 3 4

(C-3293(a))

$$\therefore \quad (54.45)_{10} = (66.34)_8$$ ...Ans.

Step 4 : Convert decimal to hexadecimal :

Binary number – $(110110.011100)_2$

Add 2 zeros to extreme left and right to get,

Binary number : 0 0 1 1 0 1 1 0 · 0 1 1 1 0 0 0 0 ← Group of 4 bits

Hex number : 3 6 · 7 0

(C-3293(b))

$$\therefore \quad (54.45)_{10} = (36.70)_{16}$$ ...Ans.

Ex. 1.11.17 : Convert $(126)_{10}$ to Octal, Hexcode.

May 15, 2 Marks

Soln. :

Step 1 : **Convert decimal to octal :**

8	126	6
8	15	7
8	1	1
	0	

LSB ↑ MSB

(C-5189)

$\therefore (126)_{10} = (176)_8$ ...Ans.

Step 2 : **Convert decimal to hexadecimal :**

16	126	14	→ E
16	7	7	→ 7
	0		

LSB ↑ MSB

(C-5190)

$\therefore (126)_{10} = (7E)_{16}$ ...Ans.

Scanned by CamScanner

Ex. 1.11.18 : Convert $(214.32)_{10}$ to binary.

May 15, 2 Marks

Soln. :

Step 1 : **Convert integer and fractional parts :**

214 · 32

Integer — Fractional part

(C-5191)

Step 2 : **Convert integer :**

2	214	0	LSB
2	107	1	
2	53	1	
2	26	0	
2	13	1	
2	6	0	
2	3	1	
2	1	1	
	0		MSB

$\therefore (214)_{10} = (11010110)_2$

Step 3 : Convert fractional part :

Decimal fraction	Base	Product	Carry
0.32 × 2 =		0.64	0 (MSB)
0.64 × 2 =		1.28	1
0.28 × 2 =		0.56	0
0.56 × 2 =		1.12	1
0.12 × 2 =		0.24	0 (LSB)

$(0.32)_{10} = (0.01010)_2$

Step 4 : **Combine results of steps 2 and 3 :**

$(214.32)_{10} = (11010110.01010)_2$

(C-5192)

...Ans.

Ex. 1.11.19 : Convert decimal number 199.375 into binary, octal, hexadecimal system.

Dec. 15, 2 Marks

Soln. :

$(199.375)_{10} = (?)_2 = (?)_8 = (?)_{16}$

Step 1 : **Convert decimal to binary :**

Separate integer and fractional part :

199 · 375

Integer — Fractional part

(C-5262)

Convert the integer :

2	199	1	LSB
2	99	1	
2	49	1	
2	24	0	
2	12	0	
2	6	0	
2	3	1	
2	1	1	
	0		MSB

$\therefore (199)_{10} = (11000111)_2$

Convert the fractional part :

Decimal	Base	Product	Carry
0.375 × 2 =		0.75	0 (MSB)
0.75 × 2 =		1.5	1
0.5 × 2 =		1.0	1
0.0 × 2 =		0.0	0 (LSB)

$\therefore (0.375)_{10} = (0.0110)_2$

(C-5263)

...Ans.

$\therefore (199.375)_{10} = (11000111.0110)_2$

Step 2 : **Convert decimal to octal :**

Convert the integer :

8	199	7	LSB
8	24	0	
8	3	3	
	0		MSB

$\therefore (199)_{10} = (307)_8$

Convert the fractional part :

Decimal	Base	Product	Carry
0.375 × 8 =		3.0	3 (MSB)
0.0 × 8 =		0.0	0 (LSB)

$\therefore (0.375)_{10} = (0.30)_8$

(C-5264)

$\therefore (199.375)_{10} = (307.30)_8$

...Ans.

Step 3 : **Convert decimal to hexadecimal :**

Convert the integer :

			Hex	
16	199	7	→ 7	LSB
16	12	12	→ C	
	0			MSB

$\therefore (199)_{10} = (C7)_{16}$

Convert the fractional part :

Decimal	Base	Product	Carry
0.375 × 16 =		6.0	6 (MSB)
0.0 × 16 =		0.0	0 (LSB)

$\therefore (0.375)_{10} = (0.60)_{16}$

(C-5265)

$\therefore (199.375)_{10} = (C7.60)_{16}$

...Ans.

Ex. 1.11.20 : Convert decimal number 151.33 into binary, base-4, octal, hexadecimal system.

Dec. 16, 4 Marks

Soln. :

1. **Decimal to binary :**

Step 1 : **Separate integer and fractional parts :**

151 · 33

Integer — Fractional

(C-5714)

Step 2 : **Convert the integer :**

2	151	1	LSB
2	75	1	
2	37	1	
2	18	0	
2	9	1	
2	4	0	
2	2	0	
2	1	1	
	0		MSB

$\therefore (151)_{10} = (10010111)_2$

(C-5715)

Step 3 : Convert the fractional part :

	Base	Product	Carry
0.33 × 2 =		0.66	0 (MSB)
0.66 × 2 =		1.32	1
0.32 × 2 =		0.64	0
0.64 × 2 =		1.28	1
0.28 × 2 =		0.56	0 (LSB)

$\therefore (0.33)_{10} = (0.01010)_2$

Scanned by CamScanner

Step 4 : Combine the results of steps 2 and 3 :

$$\therefore (151.33)_{10} = (10010111.01010)_2$$

Decimal to base-4 :

Convert the integer part :

4	151	3	LSD
4	37	1	
4	9	1	
2	2	0	
2	1	1	
	0		MSD

$\therefore (151)_{10} = (10113)_4$ (C-5716)

Convert the fractional part :

Base				Product	Carry	
0.33	×	4	=	1.32	1	MSD
0.32	×	4	=	1.28	1	
0.28	×	4	=	1.12	1	
0.12	×	4	=	0.48	1	
0.48	×	4	=	1.92	1	LSD

$\therefore (0.33)_{10} = (0.11111)_2$

$$\therefore (151.33)_{10} = (10113.11111)_4$$ ...Ans.

Decimal to octal :

Convert the obtained binary number to octal :

Binary : 010 010 111 · 010 100 – Group of 3 bits

Octal : 2 2 7 · 2 4 (C-5717)

$$\therefore (151.33)_{10} = (227.24)_8$$ ...Ans.

Decimal to hexadecimal :

Binary : 1001 0111 · 0101 0000 – Group of 4 bits

Hexadecimal : 9 7 · 5 0 (C-5718)

$$\therefore (151.33)_{10} = (97.5)_{16}$$ ...Ans.

.12 Conversion from Binary to Other Systems :

.12.1 Conversion from Binary to Decimal :

We have already discussed this.

.12.2 Binary to Octal Conversion :

For converting the given binary number into an equivalent octal number, follow the procedure ven below :

Step 1 : Divide the binary bits into groups of 3 starting from the LSB.

Step 2 : Convert each group into its equivalent decimal. As the number of bits in each group is restricted to 3, the decimal number will be same as octal number.

Ex. 1.12.1 : Convert the binary number $(1\,1\,0\,1\,0\,0\,1\,0)_2$ into its equivalent octal number.

Soln. :

Additional zero to make groups of three

1. The given number is → 0 1 1 0 1 0 0 1 0
2. Make group of 3 → Group 3, Group 2, Group 1
3. Convert into octal → 3 2 2

(C-37)

$$\therefore (11010010)_2 = (322)_8$$

Note : In the third group, there are only 2 bits. Hence we have assumed the number to be 011 instead of 11. Always add the extra zeros on the MSB side, not on LSB side.

1.12.3 Binary to Hex Conversion :

It is easy to convert from an integer binary number to hex. This is accomplished by :

Step 1 : Divide the binary number into 4-bit sections from the LSB to the MSB.

Step 2 : Convert each 4-bit binary number to its hex equivalent.

Ex. 1.12.2 : Convert the binary number 1010 1111 1011 0010 into the equivalent hex number.

Soln. :

1. Split the given number into groups of 4 bits.
2. Convert each group into corresponding hex.

1010	1111	1011	0010
A	F	B	2

$$\therefore (1010\ 1111\ 1011\ 0010)_2 = (AFB2)_{16}$$ ...Ans.

1.13 Conversion from Other Systems to Binary System :

1.13.1 Conversion from Decimal to Binary :

We have already discussed this earlier.

1.13.2 Octal to Binary Conversion :

To get the binary equivalent of the given octal number we have to convert each octal digit into its equivalent 3-bit binary number. This is as explained in the following example.

Ex. 1.13.1 : Convert the octal number $(364)_8$ into equivalent binary number.

Soln. :

Step 1 : Given octal number :

Step 2 : Convert each digit to binary :

3	6	4
011	110	100

$$\therefore (364)_8 = (0\,1\,1\,1\,1\,0\,1\,0\,0)_2$$ ...Ans.

Scanned by CamScanner

Ex. 1.13.2 : Convert $(364.25)_8$ into its equivalent binary number.

Soln. : Follow the same procedure explained in the previous example.

Step 1 : Given octal number :

Step 2 : Convert each digit into binary :

3	6	4	·	2	5
011	110	100	·	010	101

$\therefore \quad (364.25)_8 = (011\ 110\ 100 \cdot 010\ 101)_2$ ...Ans.

1.13.3 Hex to Binary Conversion :

It is also easy to convert from an integer hex number to binary. This is accomplished by :

Step 1 : Convert each hex digit to its 4-bit binary equivalent.
Step 2 : Combine the 4-bit sections by removing the spaces.

Ex. 1.13.3 : Convert the hex number AFB2 into equivalent binary number.

Soln. :

Each digit in the given hex number is converted into 4-bit binary numbers as shown in Fig. P. 1.13.3.

Given hex number	A	F	B	2
Each digit converted to its four bit binary equivalent	1010	1111	1011	0010

(C-38) **Fig. P. 1.13.3 : Hex to binary conversion**

Hence $(A\,F\,B\,2)_{16} = (1010\ 1111\ 1011\ 0010)_2$ ...Ans.

1.14 Conversion from Octal to Other Systems :

We have already discussed the following two conversions :

1. Octal to decimal 2. Octal to binary.

1.14.1 Octal to Hex Conversion :

For converting octal to hex, follow the steps given below :

Step 1 : Convert the given octal number into equivalent binary.
Step 2 : Then convert this binary number into hex.

The octal to hex conversion is demonstrated in Fig. 1.14.1.

Given octal number = (436)

Step 1 : Convert octal to binary :

Given octal number :

4	3	6
↓	↓	↓
100	011	110

Binary equivalent

Step 2 : Convert binary to hex :

Binary $\rightarrow (1\,0\,0\,0\,1\,1\,1\,1\,0)_2$

Add three zeros on extreme left (on MSB side) to get, $(0001\ 0001\ 1110)_2$

Binary number :

0001	0001	1110
↓	↓	↓
1	1	D

Groups of 4 bits

Hex number :

$\therefore \quad (436)_8 = (1\,1\,D)_{16}$ ...Ans.

Fig. 1.14.1

1.14.2 Conversion from Other Systems to Octal :

We have already discussed the following two conversions :

1. Decimal to octal 2. Binary to octal.

Hex to octal conversion :

For the hex to octal conversion follow the steps given below :

Step 1 : Represent each hex digit by a 4-bit binary number.
Step 2 : Combine these 4-bit binary sections by removing the spaces.
Step 3 : Now group these binary bits into groups of 3 bits, starting from the LSB side.
Step 4 : Then convert each of this 3-bit group into an octal digit.

Ex. 1.14.1 : Convert the hex number 4CA into its equivalent octal form.

Soln. :

Step 1 : Convert $(4CA)_{16}$ into binary :

4	C	A
0100	1100	1010

Step 2 : Combine the 4 bit binary sections by removing spaces :

$\therefore \quad (4\,C\,A)_{16} = (0100\ 1100\ 1010)_2$

Step 3 : Group these binary bits into groups of 3 bits :

(C-39)

$\therefore \quad \underbrace{010}_{2}\ \underbrace{011}_{3}\ \underbrace{001}_{1}\ \underbrace{010}_{2}$

Each 3 bit group is converted to an octal digit.

Step 4 :

$\therefore \quad (4\,C\,A)_{16} = (2\,3\,1\,2)_8$ ...Ans.

Fractional hex to octal conversion :

To convert the fractional hex number into octal we use the following steps :

Step 1 : Convert the given fractional hex number into its equivalent binary number.
Step 2 : Group the binary bits into groups of 3 bits.
Step 3 : Convert each group of 3 bits into an octal digit.

Ex. 1.14.2 : Convert $(0.12E)_{16}$ into equivalent octal number.

Soln. :

Step 1 : Convert each hex digit into 4-bit binary word :

(C-40)

Given hex number	·	1	2	E
Binary number	·	0001	0010	1110

Scanned by CamScanner

Step 2, 3 : Group the binary bits into groups of 3 bits and convert each group into an octal number :

$$\cdot\ \underbrace{000}_{0}\ \underbrace{100}_{4}\ \underbrace{101}_{5}\ \underbrace{110}_{6} \leftarrow \text{Groups of 3 bits.}$$

(C-41)

$\cdot\ 0\ \ 4\ \ 5\ \ 6 \leftarrow$ Octal digits

$\therefore\ (0.12\,E)_{16} = (0.0456)_8$ ...Ans.

Note : The bits are grouped starting from the fractional point and moving towards right.

Conversion of mixed hex number to octal :

The procedure to be followed for conversion of mixed hex number is same as the one discussed for fractional hex conversion.

Ex. 1.14.3 : Convert the hex number $(68.4B)_{16}$ into equivalent octal number.

Soln. :

1. Given hex number :

6	8	.	4	B

2. Convert to binary :

0110	1000	.	0100	1011

3. Now put additional zeros to extreme left and right

(C-42)

$$(0)\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ \cdot\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ (0)$$

Additional zeros

4. Form groups of 3 bits :

$$\underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{000}_{0}\ \cdot\ \underbrace{010}_{2}\ \underbrace{010}_{2}\ \underbrace{110}_{6}$$

(C-43)

5. Convert into octal :

$\therefore\ (68.4B)_{16} = (150.226)_8$ ...Ans.

Ex. 1.14.4 : Convert $(243.63)_8$ to decimal, binary and hexadecimal. **May 11, Dec. 12, 5 Marks**

Soln. :

1. **Conversion to decimal :**

(C-2164)

N = | 2 | 4 | 3 | . | 6 | 3 | Octal

$$2 \times 8^2 + 4 \times 8^1 + 3 \times 8^0 + 6 \times 8^{-1} + 3 \times 8^{-2}$$

$= 128 + 32 + 3 + 0.75 + 0.0468 = 163.7968$

$\therefore\ (243.63)_8 = (163.7968)_{10}$ ...Ans.

2. **Conversion to binary :**

N = | 2 | 4 | 3 | . | 6 | 3 | Octal

010 100 011 . 110 011 Binary

$\therefore\ (243.63)_8 = (010100011.110011)_2$ ...Ans.

3. **Conversion to hex :**

Obtained binary number : $(010100011.110011)_2$

Add two zero's to extreme right (on LSB side to get (10100011.11001100).

Binary number : | 1010 | 0011 | . | 1100 | 1100 | Group of 4 bits

Hex number : A 3 . C C

$\therefore\ (243.63)_8 = (A3.CC)_{16}$ ...Ans.

Ex. 1.14.5 : Convert $(650.17)_8$ into decimal, binary and hex. **Dec. 11, 6 Marks**

Soln. :

1. **Conversion to decimal :**

(C-2165)

Given octal number : | 6 | 5 | 0 | . | 1 | 7 |

$$(6 \times 8^2) + (5 \times 8^1) + (0 \times 8^0) + (1 \times 8^{-1}) + (7 \times 8^{-2})$$

$= 384 + 40 + 0 + 0.125 + 0.109$

$\therefore\ N = (424.234)_{10}$

2. **Conversion to binary :**

N = | 6 | 5 | 0 | . | 1 | 7 |

110 101 000 . 001 111

$\therefore\ (650.17)_8 = (110101000.001111)_2$

3. **Conversion to hex :**

Obtained binary number : 110101000.001111

Add three zero's to extreme left (MSB side) and two zero's to extreme right (LSB side) to get $(0001\ 1010\ 1000.0011\ 1100)_2$

Binary number : | 0001 | 1010 | 1000 | . | 0011 | 1100 | Group of 4 bits

Hex number : 1 A 8 . 3 C

$\therefore\ (650.17)_8 = (1A8.3C)_H$

Ex. 1.14.6 : Convert the $(761.514)_8$ to binary and hexadecimal. **Dec. 13, 5 Marks**

Scanned by CamScanner

Soln. :

$(761.514)_8 = (?)_2 = (?)_H$:

Step 1 : Convert octal to binary :

Given octal number

7	6	1	.	5	1	4
111	110	001	.	101	001	100

$\therefore\ (761.514)_8 = (111\ 110\ 001 \cdot 101\ 001\ 100)_2$ ...Ans.

Step 2 : Convert octal to hex :

$(761.514)_8 = (111\ 110\ 001 \cdot 101\ 001\ 100)_2$

Make group of 4 bits.

Binary number :	0001	1111	0001	.	1010	0110	0000
Hex number :	1	F	1		A	6	0

(C-4199)

$\therefore\ (761.514)_8 = (1F1.A60)_H$ ...Ans.

Ex. 1.14.7 : Convert the hex number $(67.4A)_{16}$ into equivalent octal number. Dec. 14. 2 Marks

Soln. :

1. Given hex number : 6 7 . 4 A
2. Convert to binary : 0110 0111 . 0100 1010
3. Now put additional zeros to extreme left and right and make groups of 3 bits.

001	100	111	.	010	010	100
1	4	7	.	1	2	4

(C-4957)

$\therefore\ (67.4A)_{16} = (147.124)_8$ ...Ans.

Ex. 1.14.8 : Convert $(532.125)_8$ into decimal, binary and hexadecimal. May 16. 3 Marks

Soln. :

1. Conversion to decimal :

N = 5 3 2 . 1 2 5

$(5 \times 8^2) + (3 \times 8^1) + (2 \times 8^0) \cdot (1 \times 8^{-1}) + (2 \times 8^{-2}) + (5 \times 8^{-3})$ (C-5511)

$= 320 + 24 + 2 + 0.125 + 0.03125 + 0.009765 = 346.166015$ ...Ans.

$\therefore\ (532.125)_8 = (346.166015)_{10}$

2. Conversion to binary :

N =	5	3	2	.	1	2	5
	101	011	010	.	001	010	101

$\therefore\ (532.125)_8 = (101011010 \cdot 001010101)_2$ ...Ans.

3. Conversion to hexadecimal :

Obtained binary number : 101011010 · 001010101

Add 3 zeros to extreme left and right side.

Group of 4 bits :	0001	0101	1010	.	0010	1010	1000
Hex number :	1	5	A	.	2	A	8

$\therefore\ (532.125)_8 = (15A \cdot 2A8)_{16}$ ...Ans.

1.15 Conversions Related to Hexadecimal System :

1.15.1 Other Systems to Hex :

- We are supposed to discuss the following conversions :
  1. Decimal to Hex. 2. Binary to Hex. 3. Octal to Hex.
- We have already discussed them.

1.15.2 Hex to Other Systems :

- We are supposed to discuss the following conversions :
  1. Hex to Decimal. 2. Hex to Binary 3. Hex to Octal.
- We have already discussed them.

Ex. 1.15.1 : Convert the decimal number 2048 to base 6.

Soln. :

Division	Quotient	Remainder
2048/6	= 341	2 LSD
341/6	= 56	5
56/6	= 9	2
9/6	= 1	3
1/6	= 0	1 MSD

(C-51)

$\therefore\ (2048)_{10} = (13252)_6$

...Ans.

Scanned by CamScanner

Ex. 1.15.2 : Convert decimal 214.32 into base 7.

Soln. :

N = 214 · 32 (C-52)

Integer Fractional

Step 1 : Convert the integer part

Division	Quotient	Remainder
214/7	= 30	4 LSD
30/7	= 4	2
4/7	= 0	4 MSD

$\therefore (214)_{10} = (424)_7$

Step 2 : Convert the fractional part (C-53)

Fraction	Base	Product	Carry
0.32	× 7 =	2.24	2 MSD
0.24	× 7 =	1.68	1
0.68	× 7 =	4.76	4 LSD

$\therefore (0.32)_{10} = (0.214)_7$

Step 3 : Combine :

$$(214.32)_{10} = (424.214)_7$$ ....Ans.

Ex. 1.15.3 : Determine the values of base x if :

1. $(211)_x = (152)_8$
2. $(193)_x = (623)_8$

Soln. :

1. $(211)_x = (152)_8$:

$$2 \times x^2 + 1 \times x^1 + 1 \times x^0 = 1 \times 8^2 + 5 \times 8^1 + 2 \times 8^0$$
$$2x^2 + x + 1 = 64 + 40 + 2$$
$$2x^2 + x + 1 = 106$$
$$2x^2 + x - 105 = 0 \quad \text{... (1)}$$

Solving Equation (1) we get,

$$x = 7$$
$$\therefore (211)_7 = (152)_8$$ ...Ans.

2. $(193)_x = (623)_8$:

$$1 \times x^2 + 9 \times x^1 + 3 \times x^0 = 6 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$$
$$x^2 + 9x + 3 = 384 + 16 + 3$$
$$x^2 + 9x + 3 = 403$$
$$x^2 + 9x - 400 = 0 \quad \text{... (2)}$$

Solving Equation (2) we get,

$$x = 16$$ ...Ans.
$$\therefore (193)_{16} = (623)_8$$ ...Ans.

Ex. 1.15.4 : Convert decimal (215.32) into base '7'. Dec. 14, 4 Marks

Soln. :

N = 215 · 32 (C-4958)

Integer Fractional

Step 1 : Convert the integer part

Division	Quotient	Remainder
215 / 7	30	5 LSD
30 / 7	4	2
4 / 7	0	4 MSD

$\therefore (215)_{10} = (425)_7$

Step 2 : Convert the fractional part : (C-4959)

Fraction	Base	Product	Carry
0.32	× 7 =	2.24	2 MSD
0.24	× 7 =	1.68	1
0.68	× 7 =	4.76	4 LSD

$\therefore (0.32)_{10} = (0.214)_7$

Step 3 : Combine :

$$(215{\cdot}32)_{10} = (425{\cdot}214)_7$$ ...Ans.

Ex. 1.15.5 : Convert $(121.2)_3$ into base 10. May 15, 2 Marks

Soln. :

Step 1 : **Convert $(121.2)_3$ into decimal :**

Given number : 1 2 1 · 2

Decimal : $(1 \times 3^2) + (2 \times 3^1) + (1 \times 3^0) + (2 \times 3^{-1})$ (C-5186)

$$\therefore N = 9 + 6 + 1 + 0.66 = 16.66$$
$$\therefore (121.2)_3 = (16.66)_{10}$$ ...Ans.

Review Questions

Q. 1 Define base of radix of a number system.

Q. 2 Explain the following :
(a) Bit (b) Nibble (c) Word (d) Double word

Q. 3 What are the disadvantages of a binary system ?

Q. 4 Describe the hexadecimal system.

Q. 5 Write a short note on : Octal system

Q. 6 Explain the conversion of hex to binary.

Q. 7 Explain the conversion of binary to other systems.

Q. 8 Describe the conversion of decimal to other systems.

Scanned by CamScanner

CHAPTER 2

Module 1

Arithmetic in Number Systems

Syllabus :

Binary arithmetic : Addition, Subtraction (1's and 2's complement), Multiplication and division, Octal and Hexadecimal arithmetic : Addition and subtraction (7's and 8's complement method for octal) and (15's and 16's complement method for Hexadecimal.)

2.1 Introduction :

- In the previous chapter, we have learnt about the various number system. Now let us develop various rules for carrying out the arithmetic operations such as addition, subtraction, multiplication and division.
- Binary arithmetic is essential in all the digital computers and many other digital systems.

2.1.1 Binary Addition :

- Binary addition is the key for binary multiplication, subtraction and division. The four most basic cases of binary addition are shown in Table 2.1.1.

Table 2.1.1 : Four cases of binary addition

	A		B		Addition	Comment
Case 1	0	+	0	=	0	Same as decimal addition
Case 2	0	+	1	=	1	
Case 3	1	+	0	=	1	
Case 4	1	+	1	=	10	$\bullet + \bullet = \bullet\bullet = (10)_2$

- For cases 1, 2 and 3 of Table 2.1.1, the binary addition takes place by following the rules of decimal addition.
- But concentrate on case 4. Addition of binary 1 + 1 represents the combining of one pebble and one pebble to obtain a total of two pebbles.

$$1 + 1 = \bullet\bullet \text{ two pebbles}$$

- Since binary 10 stands for $\bullet\bullet$ two pebbles, the result of binary addition 1 + 1 is 10.

$$\therefore \quad 1 + 1 = (10)_2 \qquad \ldots(2.1.1)$$

2.1.2 Sum and Carry :

- Thus the fourth case yields a binary two (10). When the binary numbers are added, the fourth case in Table 2.1.1 creates a sum of 0 in the given column and a carry of 1 over to the next column.
- The four basic rules of binary addition in terms of sum and carry are as follows.

Table 2.1.2 : Rules of binary addition

Rule	A		B		Sum	Carry
1	0	+	0	=	0	0
2	0	+	1	=	1	0
3	1	+	0	=	1	0
4	1	+	1	=	0	1

2.1.3 Addition of Large Binary Numbers :

The procedure for addition of larger binary numbers is as follows :

Steps to be followed :

Step 1 : Add the least significant bits.
Step 2 : Write the carry digit on top of the next column of bits.
Step 3 : Add carry with the digits in the second column to get the sum.
Step 4 : Write the carry on the top of next column and continue.

Refer the following example to understand binary addition.

Ex. 2.1.1 : Add the following binary numbers : 011 and 101.

Soln. :

(C-57)

Carry	1	1	1	
A		0	1	1
+ B		1	0	1
Ans :	1	0	0	0

$\therefore$ 011 + 101 = 1000
i.e. 3 + 5 = 8

2.1.4 Binary Subtraction :

Rules : In order to understand the binary subtraction, we should remember some of the important rules of decimal subtraction. They are as follows :

1. To carry out the subtraction (A – B) where A and B are the two single digit decimal numbers. We have to consider two cases,

Scanned by CamScanner

2. **Case I : Digit A > Digit B :**

$$\text{Let } A = (5)_{10} \text{ and } B = (3)_{10}$$

$$\text{Then } A - B = (\bullet\bullet\bullet\bullet\bullet) - (\bullet\bullet\bullet) = \bullet\bullet$$

$$\therefore (5)_{10} - (3)_{10} = (2)_{10}$$

3. **Case II : Digit A < Digit B :**

If $A = (3)_{10}$ and $B = (5)_{10}$ then we cannot perform (3 – 5) because we cannot take out 5 pebbles from 3. **Therefore we have to borrow 1.** After borrowing, the subtraction is changed to

$$\boxed{1}3 - 5 = 8$$

→ Borrow (C-4898)

We have to do the same thing for subtracting the binary numbers.

2.1.5 Subtraction and Borrow :

- These two words will be used very frequently for the binary subtraction. For binary subtraction we have to remember the following four cases given in Table 2.1.3.

Table 2.1.3 : Four basic rules for binary subtraction

Case	A		B	Subtraction	Borrow	Comment
1	0	–	0	0	0	same as decimal
2	1	–	0	1	0	same as decimal
3	1	–	1	0	0	same as decimal
4	0	–	1	1	1	Borrow needs to be taken

- Consider case 4 in Table 2.1.3. It is [0 – 1]. Hence a logic 1 is borrowed. This will change the subtraction from [0 – 1] to [10 – 1] that means [•• – •] = • = 1.

2.1.6 Subtraction of Larger Binary Numbers :

- Let $A = (1\ 1\ 0\ 1\ 1)_2$ and $B = (1\ 0\ 1\ 1\ 0)_2$ are the two binary numbers. Let us obtain the result of the subtraction (A – B).

Steps to be followed :

Step 1 : Subtract column by column.
Step 2 : Always start from the column of LSBs.
Step 3 : If necessary, borrow from the next higher column.

Ex. 2.1.2 : Subtract the decimal numbers $(38)_{10}$ and $(29)_{10}$ by converting them into binary.

Soln. :

Step 1 : Convert $(38)_{10}$ and $(29)_{10}$ into their binary equivalents :

Convert the given decimal numbers into binary numbers to get,

$$(38)_{10} = (1\ 0\ 0\ 1\ 1\ 0)_2, \quad (29)_{10} = (1\ 1\ 1\ 0\ 1)_2$$

Step 2 : Perform column by column subtraction from LSB to MSB :

Column of LSBs

Borrow						$\boxed{1}$		
A	1	0	0	1	1	0	Decimal 38	(C-63)
– B	0	1	1	1	0	1	Decimal 29	
						1		

Step 3 : Repeat the procedure for all the columns :

Column 5					Column 4		
Borrow		$\boxed{1}$	$\boxed{1}$			$\boxed{1}$	(C-64)
A	1	0	0	1	1	0	
– B	0	1	1	1	0	1	
		$\boxed{-1}$	$\boxed{-1}$			$\boxed{-1}$	
Ans :	0	0	1	0	0	1	

Column 4 : 10 – 1 = 1
Column 5 : 10 – 1 – 1 = •• – • – • = 0
Column 6 : 1 – 1 = 0

2.1.7 Binary Multiplication :

- The procedure used for binary multiplication is exactly same as that for the decimal multiplication.
- In fact binary multiplication is simpler than decimal multiplication because only 0s and 1s are involved.

Rules of binary multiplication :

- Rules of binary multiplication are as follows :

$$0 \times 0 = 0 \qquad 0 \times 1 = 0$$
$$1 \times 0 = 0 \qquad 1 \times 1 = 1$$

- The multiplication process of two binary numbers has been illustrated in the following example.

Ex. 2.1.3 : Perform the following binary multiplication 101.11 × 111.01.

Soln. :

	A					1	0	1.1	1		
	B				×	1	1	1.0	1		
						1	0	1	1	1	
+					0	0	0	0	0	-	← Shift left by 1 position
+				1	0	1	1	1	-	-	← Shift left by 2 positions
+			1	0	1	1	1	-	-	-	← Shift left by 3 positions
		1	0	1	1	1	-	-	-	-	← Shift left by 4 positions
	1	0	1	0	0	1	1	0	1	1	

Scanned by CamScanner

The binary point is placed after 4 positions from LSB.

$\therefore$ **Ans. : 101001.1011**

Cross check :

$$A = (101.11)_2 = (5.75)_{10} \text{ and } B = (111.01)_2 = (7.25)_{10}$$

$$\therefore A \times B = (41.6875)_{10}$$

Ans. = | 1 | 0 | 1 | 0 | 0 | 1 | · | 1 | 0 | 1 | 1 |

$$32 + 8 + \quad 1 + 0.5 + 0.125 + 0.0625 = (41.6875)_{10}$$ ...Ans.

Thus we have the correct answer.

Ex. 2.1.4 : Perform : $(11001)_2 \times (101)_2$ **May 12, 2 Marks**

Soln. :

$(11001)_2 \times (101)_2$:

```
                1 1 0 0 1
              ×        1 0 1
                1 1 0 0 1
          +     0 0 0 0 0 -   ← Shift left by 1 position
          +   1 1 0 0 1 - -   ← Shift left by 2 positions
Answer :    1 1 1 1 1 0 1
```

$\therefore (11001)_2 \times (101)_2 = (1111101)_2$ ...Ans.

2.1.8 Binary Division :

The division of binary numbers takes place in a similar way as that of decimal number. It is called as the long division procedure.

Ex. 2.1.5 : Perform directly without converting to any other base. $(11110)_2 \div (110)_2$ **May 10, 2 Marks**

Soln. :

$(11110)_2 \div (110)_2$: (C-2169)

```
          101  Quotient
110 ) 11110
      -110
        110
       -110
          0  Remainder
```

$(11110)_2 \div (110)_2 = (101)_2$ ...Ans.

Ex. 2.1.6 : Perform following operation without converting to any other base. $(10101011)_2 \div (101)_2$ **Dec. 10, 2 Marks**

Soln. :

$(10101011)_2 \div (101)_2$: (C-2172)

```
          100010
101 ) 10101011
      -101
        00101
      -   101
          001
```

Quotient = $(100010)_2$

Remainder = $(1)_2$ ...Ans.

Ex. 2.1.7 : Perform $(11010)_2 \div (101)_2$ **Dec. 10, 2 Marks**

Soln. :

$(11010)_2 \div (101)_2$: (C-2174)

```
          101
101 ) 11010
      -101
        110
      - 101
          1
```

Quotient = 101

Remainder = 1 ...Ans.

Ex. 2.1.8 : Perform $(11001)_2 \div (101)_2$ **May 12, 2 Marks**

Soln. :

$(11001)_2 \div (101)_2$: (C-3704)

```
          101
101 ) 11001
      -101
       00101
      -  101
        000
```

$\therefore (11001)_2 \div (101)_2 = (101)_2$ ...Ans.

2.2 Unsigned Binary Numbers :

- In some applications, all the data is either positive or negative. Then we can just forget about the (+) or (–) signs, and concentrate only on the magnitude (absolute value) of the data.
- For example, the smallest 8 bit binary number is 0000 0000 i.e. all zeros, and the largest 8 bit binary number is 1111 1111. Hence the complete range of unsigned 8 bit binary numbers extends from $(00)_H$ to $(FF)_H$ or from $(00)_{10}$ to $(255)_{10}$.

It is important to note that we have not included (+) or (–) signs with these numbers.

- Similarly for 16-bit numbers the complete range is given by,

  Smallest : 0000 0000 0000 0000 = $(0000)_H$

  Largest : 1111 1111 1111 1111 = $(FFFF)_H$

- This represents the magnitude of 0 to 65,535 decimal.

1	0	1	1	0	1	0	0	← Unsigned 8-bit binary number

All the bits are used for representing only the magnitude

Fig. 2.2.1 : Unsigned 8 bit binary number

- Data of this type is called as **unsigned binary** numbers because all of the bits in a binary number are used to represent the magnitude of the corresponding decimal number, as shown in Fig. 2.2.1.

2.2.1 Important Features of Unsigned Numbers :

1. We can add or subtract the unsigned binary numbers if certain conditions are satisfied.

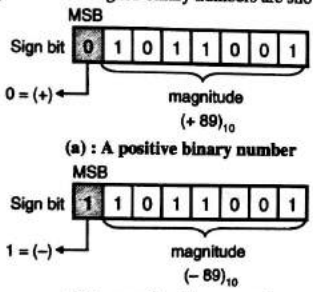
Scanned by CamScanner

2. The microcomputers of first generation were able to process only 8 bits at a time. Therefore with 8-bit unsigned arithmetic all the magnitudes were restricted between 0 and $(255)_{10}$ or $(00)_H$ to $(FF)_H$.
3. That means all the numbers being added or subtracted must be in the range 0 to 255. More important is that the answer also should be in the range 0 to 255.
4. For the magnitudes greater than 255 we have to use 16-bit arithmetic.

Overflow : If the addition or multiplication of two 8-bit numbers results in generation of a number, greater than $(255)_{10}$, then it is said that overflow has taken place.

2.3 Sign-Magnitude Numbers :

- If the data has positive as well as negative numbers then the signed binary numbers should be used, for their representation.
- For a sign-magnitude representation, the + or – signs are also represented in the binary form i.e. by using 0 or 1. So a 0 is used to represent the (+) sign and 1 is used to represent the (–) sign.
- The MSB of a binary number is used to represent the sign and the remaining bits are used for representing the magnitude. 8-bit signed binary numbers are shown in Fig. 2.3.1.



(a) : A positive binary number

(b) A negative binary number

Fig. 2.3.1 : 8-bit signed binary numbers

Definition of signed numbers :

The numbers shown in Fig. 2.3.1 contain a sign bit followed by magnitude bits. Numbers represented in this form are called as **sign-magnitude numbers** or only sign-numbers.

2.3.1 Range of Sign-Magnitude Numbers :

- The unsigned 8-bit numbers cover the decimal range of 0 to 255.
- But in the sign magnitude numbers, the MSB is utilized for representing the sign. Therefore the range gets modified. (as there are only 7 bits left to represent the magnitude).
- For an 8-bit sign-magnitude number, the largest negative number is (– 127) given by,

$= (-127)_{10}$

- And the largest positive number is (+ 127) given by,

$= (+127)_{10}$

In this way the range of sign-magnitude, 8-bit binary numbers is modified to decimal (– 127) to (+127) from 0 to 255.

- The largest magnitude is 127, which is approximately half of the largest magnitude obtained for unsigned binary numbers.
- With the sign-magnitude numbers, we can use the 8-bit arithmetic as long as the input data range falls in decimal – 127 to + 127. It is still necessary to check all the sums for overflow.
- If the magnitude of data is greater than 127 then 16 bit arithmetic should be used. With 16-bit numbers the range of sign-magnitude numbers extends from decimal (– 32,767) to (+ 32,767).

Advantages of sign-magnitude numbers :

1. The main advantage is their simplicity.
2. We can easily find the magnitude by deleting the sign bit.

Disadvantage :

The sign-magnitude numbers have a limited use because they require complicated circuits. These numbers are often used in analog-to-digital (A to D) converters.

2.4 Complements :

- Complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations.

2.4.1 Types of Complements :

- For each radix-r system there are two types of complements :
  1. The radix complement. 2. The diminished radix complement.
- The **radix complement** is referred to as the r's complement and the **diminished radix complement** is referred to as (r – 1)'s complement.
- Consider the binary system with a base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.
- Similarly for an octal system we have 8's complement and 7's complement. For a decimal system we have r = 10 and the types of complements are 10's complements and 9's complement.

2.4.2 1's Complement :

- The 1's and 2's complement of a binary number are important because we can use them for representation of negative numbers.
- The 1's complement of a number is found by inverting all the bits in that number. This is called as taking complement.
- This complemented value represents the negative of the original number. The 1's complement system is very easy to implement merely using inverters.

Ex. 2.4.1 : Obtain the 1's complement of the following numbers. $(1010)_2$, $(11010101)_2$.

Soln. :

Given number	1010	1101 0101
1's complement	0101	0010 1010

Ex. 2.4.2 : Convert following binary numbers to 1's complement : (a) $(1101)_2$ (b) $(1011)_2$

Soln. :

(a) **Given number :** 1 1 0 1 **1's complement :** 0 0 1 0 ...Ans.

(b) **Given number :** 1 0 1 1 **1's complement :** 0 1 0 0 ...Ans.

2.4.3 2's Complement :

- The 2's complement of a binary number is obtained by adding 1 to the LSB of 1's complement of that number.

$$\therefore \text{ 2's complement} = \text{1's complement} + 1$$

Scanned by CamScanner

- The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

Ex. 2.4.3 : Obtain the 2's complement of $(1011\ 0010)_2$.

Soln. :

1.	Given number :	1	0	1	1	0	0	1	0
2.	1's complement :	0	1	0	0	1	1	0	1
3.	Add 1 to + 1's complement								1
4.	2's complement :	0	1	0	0	1	1	1	0

Hence the 2's complement of $(1011\ 0010)_2$ is $(0100\ 1110)_2$.

Ex. 2.4.4 : Find the one's complement and two's complement of $(57)_{10}$. **May 15. 2 Marks**

Soln. :

Step 1 : Convert decimal to binary :

$$(57)_{10} \longrightarrow (111001)_2$$

Step 2 : 1's complement of $(57)_{10}$:

$$(111001)_2 \xrightarrow{\text{1's complement}} (000110)_2$$

Step 3 : 2's complement of $(57)_{10}$:

$$(111001)_2 \xrightarrow{\text{1's complement}} (000110)_2 \xrightarrow{\text{add 1}} (000111)_2$$

2.5 Binary Subtraction using 1's and 2's Complements :

- The direct binary subtraction becomes complicated as the number size increases.
- Therefore we can represent the subtraction of A – B in the form of addition as : A + (– B).
- We can represent number B (which is to be subtracted) in its 1's complement or 2's complement form and use addition instead of subtraction to get the result of A – B.

2.5.1 Subtraction using 1's Complement :

The steps to be followed for subtraction $(A)_2 - (B)_2$ using 1's complement are as follows :

Step 1 : Convert number to be subtracted $(B)_2$ to its 1's complement.

Step 2 : Add $(A)_2$ and 1's complement of $(B)_2$ using the rules of binary addition.

Step 3 : If final carry is 1, then add it to the result of addition obtained in step 2 to get the final result of $(A)_2 - (B)_2$. Note that if the final carry is 1 then the subtraction is positive and in its true form.

Step 4 : If the final carry produced in step 2 is 0, then the result obtained in step 2 is negative and in the 1's complement form. So convert it into the true form by complementing all the bits.

The following examples will make the concept of subtraction using 1's complement crystal clear.

Ex. 2.5.1 : Subtract $(32)_{10}$ from $(85)_{10}$ using 1's complement binary arithmetic.

Soln. :

$(85)_2 - (32)_2$: Convert both the numbers to binary :

$$(85)_{10} = 1010101\ (32)_{10} = 0100000$$

Step 1 : Obtain 1's complement of $(32)_{10}$:

1's complement of $(32)_{10}$ or $(0100000)_2$ is $(1011111)_2$.

Step 2 : Add $(85)_{10}$ and 1's complement of $(32)_{10}$:

$$\begin{array}{lr} (85)_{10} & 1\,0\,1\,0\,1\,0\,1 \\ \text{1's complement of } (32)_{10} & +\ 1\,0\,1\,1\,1\,1\,1 \\ \hline & \boxed{1}\,0\,1\,1\,0\,1\,0\,0 \end{array}$$

Result

Final carry is generated

(C-2948)

Step 3 : Add the final carry to the result obtained in step 2 :

$$\begin{array}{lr} & 0\ 1\ 1\ 0\ 1\ 0\ 0 \\ & +\qquad\qquad\ \ 1 \\ \hline \text{Answer in true form} & 0\ 1\ 1\ 0\ 1\ 0\ 1 \end{array} \leftarrow \text{Result} \quad = (53)_{10}$$

Thus the answer is positive and in the true form.

$$\therefore (85)_{10} - (32)_{10} = (53)_{10}$$

...Ans.

Ex. 2.5.2 : Perform the following operations :
$(1\,0\,1\,1\,0\,0\,1)_2 - (1\,1\,0\,1\,0\,1\,0)_2$ using 1's complement method.

Soln. :

$(1\,0\,1\,1\,0\,0\,1)_2 - (1\,1\,0\,1\,0\,1\,0)_2$ using 1's complement method.

Step 1 : Take complement of 1101010 :

1's complement of 1101010 : 0010101

Step 2 : Add $(1011001)_2$ and 1's complement of $(1101010)_2$:

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

Answer is negative and in 1's complement form

Final carry is 0

(C-2949)

$$\underbrace{1\,1\,0\,1\,1\,1\,0}_{\text{Invert all bits}} \downarrow \underbrace{0\,0\,1\,0\,0\,0\,1}_{\text{Answer}}$$

$$\therefore (1011001)_2 - (1101010)_2 = (10001)_2$$

...Ans.

Ex. 2.5.3 : Perform the given operation $(1111)_2 - (0110)_2$ using 1's complement.

Soln. :

$(1111)_2 - (0110)_2$

Step 1 : Obtain 1's complement of $(0110)_2$:

$$0110 \xrightarrow{\text{1's complement}} 1001$$

Scanned by CamScanner

Step 2 : Add $(1111)_2$ and 1's complement of $(0110)_2$:

(C-3092)

		1	1	1	1
1's Complement of $(0110)_2$:	+	1	0	0	1
Final carry		1	1	0	0
Add final carry	+				1
		1	0	0	1

$(1111)_2 - (0110)_2 = (1001)_2$ ...Ans.

2.5.2 Binary Subtraction using 2's Complement Method :

If the subtraction of two binary numbers A and B is to be performed using the 2's complement, then the following steps are to be followed.

Steps to be followed :

Step 1 : Add $(A)_2$ to the 2's complement of $(B)_2$.

Step 2 : If the carry is generated then the result is positive and in its true form.

Step 3 : If the carry is not produced, then the result is negative and in its 2's complement form.

Note : Carry is always to be discarded in the subtraction using 2's complement.

Ex. 2.5.4 : Perform $(9)_{10} - (5)_{10}$ using 2's complement method.

Soln. :

Step 1 : Obtain 2's complement of $(5)_{10}$:

Decimal	Binary	2's complement
$(5)_{10}$	$(0101)_2$	1011

Step 2 : Add $(9)_{10}$ to 2's complement of $(5)_{10}$:

$(9)_{10}$		1	0	0	1
2's complement of $(5)_{10}$	+	1	0	1	1
Carry		1	1		
	1	0	1	0	0 → $(4)_{10}$

Discard Carry → 1

Final carry indicates that the answer is positive and in its true form.

Answer

(C-70(a))

$\therefore \quad (9)_{10} - (5)_{10} = (4)_{10}$ ...Ans.

Note : The final carry bit acts as a sign bit for the answer. If it is 1 then the answer is positive, and if it is 0 then the answer is negative.

Ex. 2.5.5 : Perform subtraction using 2's complement for given numbers (– 48) – (+23) use 8 bit representation of number.

Soln. :

Step 1 : Convert both the numbers to their 2's complement :

Decimal	Binary	2's complement
$(48)_{10}$	$(00110000)_2$	(11010000)
$(23)_{10}$	$(00010111)_2$	(11101001)

Step 2 : Add 2's complement of $(48)_{10}$ and $(23)_{10}$:

2's complement of $(48)_{10}$	1	1	0	1	0	0	0	0
+								
2's complement of $(23)_{10}$	1	1	1	0	1	0	0	1
Final carry	1	1	0	1	1	1	0	0	1

Result is negative and in 2's complement form.

Step 3 : Convert answer to its true form :

Answer :	1	0	1	1	1	0	0	1	
Subtract 1 –								1	
:	1	0	1	1	1	0	0	0	
Invert all bits :	0	1	0	0	0	1	1	1	Answer in true form

$\therefore -(01000111)_2 = (-71)_{10}$

$\therefore (-48) - (+23) = -(01000111)_2 = (-71)_{10}$

Ex. 2.5.6 : Perform following subtraction using 2's complement method $(1101)_2 - (1001)_2$.

Soln. :

Given : $A = (1101)_2$; $B = (1001)_2$

Step 1 : 2's complement of B :

$(1001)_2 \xrightarrow[\text{All}]{\text{Invert}} (0110)_2 \xrightarrow[1]{\text{Add}} 0111$ → 2's complement of B

(C-2958)

Step 2 : Addition : (C-2951)

A		1	1	0	1
2's complement of B	+	0	1	1	1
		1	1	1	
	1	0	1	0	0

Final carry — Answer

As final carry is 1 the answer is positive and in true binary form.

$\therefore (1101)_2 - (1001)_2 = (0100)_2$ ...Ans.

Ex. 2.5.7 : Perform binary subtraction using 2's complement for $(62)_{10}$ and $(99)_{10}$.

May 15. 4 Marks

Soln. :

Step 1 : Convert both the numbers to binary :

$(62)_{10} = (0111110)_2$

$(99)_{10} = (1100011)_2$

Step 2 : Obtain 2's complement of $(99)_{10}$:

Decimal	Binary	1's complement	2's complement
$(99)_{10}$ →	$(1100011)_2$ →	$(0011100)_2$ →	$(0011101)_2$

Scanned by CamScanner

Step 3 : **Add $(62)_{10}$ and 2's complement of $(99)_{10}$:**

$(62)_{10}$: 0 1 1 1 1 1 0
+
2's complement of $(99)_{10}$: 0 0 1 1 1 0 1
Carry : 1 1 1 1

$\boxed{0}$ 1 0 1 1 0 1 1 (C-5193)

→ "0" carry indicates that the result is negative and in its 2's complement form

Step 4 : **Convert the answer into its true form :**

Answer : 1 0 1 1 0 1 1
−
Subtract 1 : 1

1 0 1 1 0 1 0 (C-5194)

↓ Invert all bits

0 1 0 0 1 0 1

$\therefore\ (62)_{10} - (99)_{10} = -(0100101)_2 = -(37)_{10}$ ...Ans.

Ex. 2.5.8 : Subtract using 1's and 2's complement method $(73)_{10} - (49)_{10}$ **Dec. 10, 2 Marks**

Soln. :

$(73)_{10} - (49)_{10}$: Using 1's complement method :

Step 1 : **Convert $(49)_{10}$ into 1's complement :**

$(49)_{10} = (0110001)_2$

and 1's complement of $(0110001)_2 = (1001110)_2$

Step 2 : **Add $(73)_{10}$ and 1's complement of $(49)_{10}$:** (C-2173)

$(73)_{10}$: 1 0 0 1 0 0 1
1's complement of $(49)_{10}$: 1 0 0 1 1 1 0

Final carry → $\boxed{1}$ 0 0 1 0 1 1 1

Step 3 : Add the final carry : → 1

Answer in the true form 1 1 0 0 0 = $(24)_{10}$

Thus the answer is positive and in the true form.

$\therefore\ (73)_{10} - (49)_{10} = (24)_{10}$ ...Ans.

Using 2's complement method :

Step 1 : Obtain 2's complement of $(49)_{10}$:

Decimal	Binary	2's complement
$(49)_{10}$	$(0110001)_2$	1001111

Step 2 : Add $(73)_{10}$ to 2's complement of $(49)_{10}$:

$(73)_{10}$: 1 0 0 1 0 0 1
2's complement of $(49)_{10}$: 1 0 0 1 1 1 1

Discard carry → $\boxed{1}$ 0 0 1 1 0 0 0 = $(24)_{10}$

Answer in the true form ←

Final carry 1 indicates that answer is positive and in true form.

$\therefore\ (73)_{10} - (49)_{10} = (24)_{10}$ ...Ans.

Ex. 2.5.9 : Subtract using 1's and 2's complement $(15)_{10} - (21)_{10}$. **May 12, 4 Marks**

Soln. :

Using 1's complement method :

Step 1 : Obtain 1's complement of $(21)_{10}$:

$(21)_{10} \xrightarrow{\text{Binary}} (10101)_2 \xrightarrow{\text{1's complement}} (01010)_2$

Step 2 : Add $(15)_{10}$ and 1's complement of $(21)_{10}$:

$(15)_{10}$: 0 1 1 1 1
$(21)_{10}$: 0 1 0 1 0
Carry 1 1 1

Final carry → $\boxed{0}$ 1 1 0 0 1 (C-3707)

As the final carry is 0, the answer is negative and in it's 1's complement form.

So convert the answer into its true form as follows :

$11001 \xrightarrow{\text{invert}} 00110$

$\therefore\ (15)_{10} - (21)_{10} = (-6)_{10} = -(110)_2$ ...Ans.

Using 2's complement form :

Step 1 : Obtain 2's complement of $(21)_{10}$:

$(21)_{10} \xrightarrow{\text{Binary}} (10101)_2 \xrightarrow{\text{1's complement}} (01010)_2 \xrightarrow{\text{add 1}} (01011)_2$

Step 2 : Add $(15)_{10}$ and 2's complement of $(21)_{10}$:

$(15)_{10}$: 0 1 1 1 1
$(21)_{10}$: 0 1 0 1 1
1 1 1 1

Final carry → $\boxed{0}$ 1 1 0 1 0 (C-3708)

As the final carry is 0, the answer is negative and in it's 2's complement form.

Scanned by CamScanner

Step 3 : Convert the answer into its true form :

```
Answer      : 1 1 0 1 0
Subtract 1  :         1
              -----------
              1 1 0 0 1
              |_____|
                 |
                 ↓ invert all bits
              00110
```

(C-3709)

$\therefore\ (15)_{10} - (21)_{10} = -(6)_{10} = (110)_2$ ...Ans.

Ex. 2.5.10 : Subtract the following using method given below :
1. $(11)_{10} - (22)_{10}$ using 2's complement.
2. $(33)_{10} - (44)_{10}$ using one's complement. **Dec. 13. 5 Marks**

Soln. :

1. $(11)_{10} - (22)_{10}$ using 2's complement :

Step 1 : Obtain 2's complement of $(22)_{10}$:

$$(22)_{10} \xrightarrow{\text{Binary}} (10110)_2 \xrightarrow[\text{complement}]{\text{1's}} (01001)_2 \xrightarrow{\text{add 1}} (01010)_2$$

Step 2 : Add $(11)_2$ and 2's complement of $(22)_{10}$:

```
                    (11)2  :  0 1 0 1 1
2's complement of (22)10   :  0 1 0 1 0
                              1   1
                           ------------
Final carry ──→            [0] 1 0 1 0 1
```

(C-4200)

As final carry is not generated the answer is negative and in 2's complement form.

Step 3 : Convert the answer into its true form :

$$10101 \xrightarrow[\text{complement}]{\text{2's}} 1011$$

$\therefore\ (11)_{10} - (22)_{10} = -(1011)_2 = -(11)_2$ ...Ans.

2. $(33)_{10} - (44)_{10}$ using one's complement :

Step 1 : Obtain 1's complement of $(44)_{10}$:

$$(44)_{10} \xrightarrow{\text{Binary}} (101100)_2 \xrightarrow{\text{1's complement}} (010011)_2$$

Step 2 : Add $(33)_{10}$ and 1's complement of $(44)_{10}$:

```
                    (33)10 :  1 0 0 0 0 1
1's complement of (44)10   :  0 1 0 0 1 1
                    Carry  :        1 1
                           ------------
Final carry ──→            [0] 1 1 0 1 0 0
```

(C-4201)

As final carry is not generated the answer is negative and in its 1's complement form.

Invert all bits $\therefore\ (110100)_2 \longrightarrow (001011)_2$

$\therefore\ (33)_{10} - (44)_{10} = -(1011)_2 = -(11)_2$ ...Ans.

Ex. 2.5.11 : Subtract using 1's and 2's complement method $(56)_{10} - (76)_{10}$. **May 16. 4 Marks**

Soln. :

Using 1's complement method :

Step 1 : Obtain 1's complement of $(76)_{10}$:

$$(76)_{10} \xrightarrow{\text{Binary}} (1001100)_2 \xrightarrow{\text{1's complement}} (0110011)_2$$

Step 2 : Add $(56)_{10}$ and 1's complement of $(76)_{10}$:

```
                    (56)10 :  0 1 1 1 0 0 0
1's complement of (76)10   :  0 1 1 0 0 1 1
                    Carry  :  1 1
                           ----------------
Final carry ──→            [0] 1 1 0 1 0 1 1
```

(C-5517)

As the final carry is 0, the answer is negative and in its 1's complement form. So convert answer into its true form as follows :

$$\text{Answer} : (1101011)_2 \xrightarrow{\text{Invert}} (0010100)_2 = (20)_{10}$$

$\therefore\ (56)_{10} - (76)_{10} = -(20)_{10}$ ...Ans.

Using 2's complement form :

Step 1 : Obtain 2's complement of $(76)_{10}$:

$$(76)_{10} \xrightarrow{\text{Binary}} (1001100)_2 \xrightarrow{\text{1's complement}} (0110011)_2 \xrightarrow{\text{add 1}} (0110100)_2$$

Step 2 : Add $(56)_{10}$ and 2's complement of $(76)_{10}$:

```
                    (56)10 :  0 1 1 1 0 0 0
2's complement of (76)10   :  0 1 1 0 1 0 0
                    Carry  :  1 1
                           ----------------
Final carry ──→            [0] 1 1 0 1 1 0 0
```

(C-5518)

As the final carry is 0, the answer is negative and in its 2's complement form.

Step 3 : Convert the answer into its true form :

```
Answer      : 1 1 0 1 1 0 0
Subtract 1  :             1
              ---------------
              1 1 0 1 0 1 1
              |_____________|
                    |
                    ↓ Invert
              0010100 = (20)10
```

(C-5519)

$\therefore\ (56)_{10} - (76)_{10} = -(20)_{10}$ ...Ans.

Scanned by CamScanner

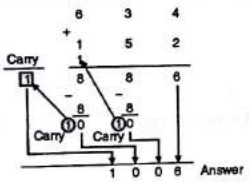
2.6 Octal Arithmetic :

2.6.1 Octal Addition :

- The sum of two octal numbers is same as sum of their decimal equivalents as long as the decimal sum is less than 8.
- But if the decimal sum is equal to or greater than 8 then subtract 8 from it (in order to obtain the octal digit, and generate a carry 1. The octal addition is illustrated in the following examples.

Ex. 2.6.1 : Add $(634)_8$ and $(152)_8$.

Soln. : The addition takes place as shown in Fig. P. 2.6.1.



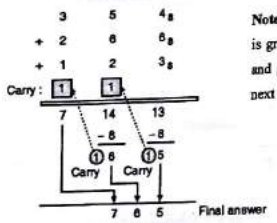
Note : Whenever column addition is greater than or equal to 8, subtract 8 and generate carry 1. Add this carry to next column.

Hence $(634)_8 + (152)_8 = (1006)_8$ ...Ans.

(C-1681) **Fig. P. 2.6.1**

Ex. 2.6.2 : Add the octal numbers. 354_8, 266_8 and 123_8.

Soln. : Refer Fig. P. 2.6.2 for solution.



Note : Whenever column addition is greater than or equal to 8, subtract 8 and generate carry 1. Add this carry to next column.

$(354)_8 + (266)_8 + (123)_8 = (765)_8$ ...Ans.

(C-1682) **Fig. P. 2.6.2**

2.6.2 Subtraction of Octal Numbers :

The following methods can be used for octal subtraction :

1. Direct subtraction.
2. Convert the numbers to binary, perform the subtraction and convert the result back to octal.
3. Use the 7's complement method.
4. Use the 8's complement method.
5. We will discuss the methods 1, 3 and 4.

2.6.3 Method 1 : Direct Subtraction of Octal Numbers :

- In the direct subtraction of octal numbers, we use the same rules as those used in decimal subtraction.
- That means, if the Minuend (the number from which second number is to be subtracted) is less than the subtrahend (the number to be subtracted) then we take borrow and return carry.
- The direct subtraction of octal numbers is illustrated in the following example.

Ex. 2.6.3 : Perform $(75)_8 - (68)_8$ without conversion.

Soln. :

$A = (75)_8$, $B = (68)_8$

(C-1992)

Borrow :		$\boxed{8}$ $(13)_{10}$
A :	7	5
B :	− 6	8
Carry :	$\boxed{1}$ $(7)_{10}$	
Result :	0	5

$\therefore (75)_8 - (68)_8 = (05)_8$ ....Ans.

Ex. 2.6.4 : Perform $(653)_8 - (177)_8$.

Soln. :

$A = (653)_8$, $B = (177)_8$

(C-1993)

Borrow :		$\boxed{8}$ $(13)_{10}$	$\boxed{8}$ $(11)_{10}$
A :	6	5	3
B :	− 1	7	7
Carry :	$\boxed{1}$ $(2)_{10}$	$\boxed{1}$ $(8)_{10}$	
Result :	4	5	4

$\therefore (653)_8 - (177)_8 = (454)_8$

2.6.4 Method 2 : Subtraction using 7's Complement :

The 7's complement is obtained by subtracting each digit from 7. This is illustrated in the following examples.

Ex. 2.6.5 : Find the seven's complement of $(512)_8$.

Soln. : Subtract $(512)_8$ from $(777)_8$ to get the seven's complement.

$$\begin{array}{r} 777 \\ -\,512 \\ \hline 265 \end{array}$$

Hence 7's complement of $(512)_8$ is $(265)_8$.

Procedure of subtraction using 7's complement :

Step 1 : Obtain 7's complement of the number to be subtracted.

Scanned by CamScanner

Step 2 : Add the first number (A) and the 7's complement of the number to be subtracted (B).
Step 3 : If carry is produced then add it to the sum obtained in step 2. The result will be in its true form.
Step 4 : If carry is not produced, then the sum obtained in step 2 is negative so take its 7's complement to obtain the answer.

Ex. 2.6.6 : Perform the subtraction $(536)_8 - (345)_8$ using the 7's complement method.

Soln. :

Step 1 : Obtain the 7's complement of $(345)_8$:

```
   777
 - 345
   432   7's complement
```

Step 2 : Add $(536)_8$ and 7's complement of 345 :

```
        536
        432
Carry     1
Final carry 1 170  simple sum
            1      Add final carry to the simple sum
Step 3 :
Answer :    171
```

Note : As the final carry is 1, the answer is positive and in its true form. Hence answer is $(171)_8$

$(536)_8 - (345)_8 = (171)_8$ ...Ans.

Ex. 2.6.7 : Perform the subtraction $(161)_8 - (243)_8$ using the 7's complement.

Soln. :

Step 1 : 7's complement of $(243)_8$:

```
   777
 - 243
   534   7's complement of (243)8
```

Step 2 : Add $(161)_8$ and 7's complement of $(243)_8$:

```
        161
        534
Carry     1
          0 715  Negative Result
            →    No final carry so the result is negative
```

Step 3 : No carry so take 7's complement of the result :

```
   777
 - 715   Result obtained in step 2.
   062   Answer
```

Hence $(161)_8 - (243)_8 = (62)_8$ and the answer is negative.

2.6.5 Method 3 : Subtraction using 8's Complement :

8's Complement of an octal number :

The 8's complement of an octal number can be obtained by adding 1 to the 7's complement of the given number.

$\therefore$ **8's complement = 7's complement + 1**

Ex. 2.6.8 : Obtain the 8's complement of the number $(324)_8$.

Soln. :

Step 1 : Obtain 7's complement :

```
   777
 - 324
   453   7's complement
Step 2 :
 +   1   Add 1 to 7's complement
   454   8's complement
```

Procedure for octal subtraction using 8's complement :

The procedure to perform $(A)_8 - (B)_8$ using the 8's complement is as follows ; where A and B are two octal numbers.

Step 1 : Obtain the 8's complement of number B.
Step 2 : Add (A)8 and 8's complement of B.
Step 3 : If carry is produced in step 2 then discard it. The addition is the answer in its true form.
Step 4 : If carry is not produced in step 2, then the answer is negative. Find the 8's complement of the sum produced in step 2 to get the answer in true form.

Ex. 2.6.9 : Perform the subtraction $(536)_8 - (345)_8$ using the 8's complement method.

Soln. :

Step 1 : Obtain the 8's complement of 345 :

```
   777
 - 345
   432   7's complement
 +   1
   433   8's complement of 345
```

Step 2 : Add $(536)_8$ and 8's complement of $(345)_8$:

```
        536
      + 433   8's complement of (345)8
Carry     1
Final carry 1 171
Discard it
                Answer is positive and in its true form.
```

Hence $(536)_8 - (345)_8 = (171)_8$ ...Ans.

2.6.6 Octal Multiplication :

Now we are going to learn multiplication in octal system. If you directly multiply octal number it comes bit complicated therefore normal procedure followed is as follows :

Scanned by CamScanner

Step 1 : Convert both octal number to binary. (both, multiplier and multiplicant.)
Step 2 : Perform simple binary multiplication.
Step 3 : After performing multiplication, whatever answer you get in binary, convert it to equivalent octal.

Ex. 2.6.10 : Perform $(12)_8 \times (7)_8$.

Soln. :

Step 1 : Convert both octal number to binary :

$$(12)_8 = (0\,0\,1\,0\,1\,0) = (1\,0\,1\,0)_2$$
$$(7)_8 = (1\,1\,1)_2 = (1\,1\,1)_2$$

Step 2 : Perform binary multiplication :

```
      1010
       111
    ------
      1010
     1010-
    1010--
    -------
    1000110     Binary
    1  0  6     (106)8
```

To cross check. $(12)_8 \times (7)_8 = (1 \times 8^1 + 2 \times 8^0) \times [7 \times 8^0] = (10)_{10} \times (7)_{10} = (70)_{10}$

Consider result i.e. $(106)_8$

$$(106)_8 = 1 \times 8^2 + 0 \times 8^1 + 6 \times 8^0 = (64 + 0 + 6)_{10} = (70)_{10}.$$

2.6.7 Octal Division :

For division in octal system, we follow the same steps i.e.

Step 1 : Convert both the given octal numbers to equivalent binary numbers.
Step 2 : Perform binary division.
Step 3 : Convert given binary quotient and remainder to octal.

Ex. 2.6.11 : Perform $(24)_8 \div (4)_8$. (C-2005)

Soln. :

Step 1 : Convert both numbers to binary :

```
2  4   octal   4
010 100 Binary 100   Divisor 100) 10100   101  Quotient
                                  100
                                  ----
                                   0100
                                    100
                                   ----
                                    000  Remainder
```

Step 2 : Convert the answer into octal :

$$(101)_2 = (5)_8$$

2.7 Hexadecimal Arithmetic :

In this section we are going to discuss the hexadecimal addition and subtraction.

2.7.1 Hex Addition :

1. The sum of two hex numbers is same as the sum of their decimal equivalents as long as the decimal sum is less than 16. (i.e. from 0 to 15)
2. But if the decimal sum is equal to or greater than 16 then subtract 16 from it (in order to obtain the hex digit) and generate a carry 1. The hex addition is illustrated in Ex. 2.7.1.

Ex. 2.7.1 : Perform the addition of $(8)_H$ and $(5)_H$.

Soln. : The subscript H indicates that both the numbers are hex numbers.

$$\therefore \quad 8\,H + 5\,H = (13)_{10} = D_H$$ ...Ans.

Note : 1. The addition takes place similar to that of decimal number.
2. As D_H is less than 16, no correction is necessary.

Ex. 2.7.2 : Add 9 H and 8 H.

Soln. :

Step 1 : Add the numbers by assuming them to be decimal numbers :

$$9 + 8 = (17)_{10}$$

Step 2 : Correct the result because it is greater than 16 :

Two corrections will have to be carried out as follows :

1. Subtract 16 from the result : $(17)_{10} - (16)_{10} = 1$
2. Generate a carry = 1.

```
        ∴ 9H
        + 8H
Carry 1   1
∴ 9H + 8H = 11H
```
...Ans.

Ex. 2.7.3 : Add $C2_H$ and $3E_H$.

Soln. : The addition takes place as shown in Fig. P. 2.7.3.

Step 1 : Add the digits by assuming them to be digital :

```
  C  2 ---> (12)10  (2)10
+ 3  E ---> (3)10   (14)10
Step 2 : Correct the result :
            (15)10  (16)10 ] Correction necessary
          + (1)     -(16)10 ] subtract 16 and generate carry.
Correction [ (16)10   (1) 0
carried out [ -(16)10  Carry
            (1) 0
Final answer : (1 0 0)16
```

(C-1683)

Hence $C2_H + 3E_H = (100)_H$

Fig. P. 2.7.3 : Addition of two hex numbers ...Ans.

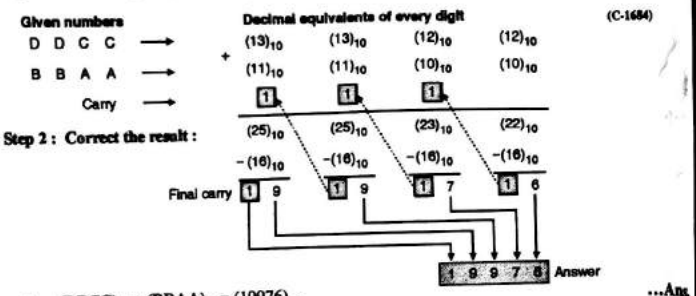
Scanned by CamScanner

Ex. 2.7.4 : Add $(DDCC)_{16}$ and $(BBAA)_{16}$. Dec. 14, 2 Marks

Soln. :

The addition takes place as shown in Fig. P. 2.7.4.

Step 1 : Add the digits by assuming them to be digital :



$\therefore$ $(DDCC)_{16} + (BBAA)_{16} = (19976)_{16}$ ...Ans.

Fig. P. 2.7.4 : Addition of $(DDCC)_{16}$ and $(BBAA)_{16}$

2.7.2 Hex Subtraction :

There are various methods used to perform the hex subtraction. They are as follows :

1. Direct subtraction.
2. Subtraction by converting the given numbers into binary.
3. Subtraction using 15's complement.
4. Subtraction using 16's complement.

Out of them, we will concentrate only on the direct method and the methods using 15's and 16's complement.

2.7.3 Method 1 : Direct Subtraction :

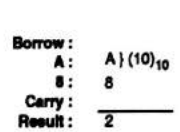
- This method is similar to the direct octal subtraction.
- If the Minuend is less than the subtrahend then we have to take borrow and return carry.
- The concept of direct hex subtraction is illustrated in the following examples.

Ex. 2.7.5 : Perform the following hex subtraction without converting the numbers :

(a) $(A)_{16} - (8)_{16}$ (b) $(73)_H - (1C)_H$ (c) $(A65)_H - (777)_H$.

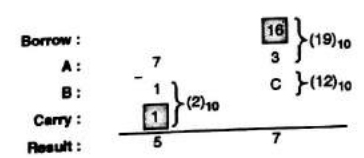
Soln. :

(a) $(A)_{16} - (8)_{16}$: (C-2007)

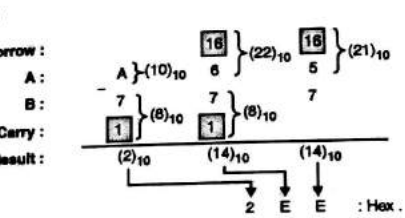


$\therefore (A)_{16} - (8)_{16} = (2)_{16}$ ...Ans

(b) $(73)_{16} - (1C)_{16}$: (C-2008)



(c) $(A65)_H - (777)_H$: (C-2009)



$\therefore$ $(A65)_{16} - (777)_{16} = (2EE)_{16}$ ...Ans.

2.7.4 Hex Subtraction using 15's Complement :

The 15's complement of a hex number is obtained by subtracting each hex digit from 15. This is illustrated in the following example.

Ex. 2.7.6 : Convert the hex number $(C\ 8\ 4)_H$ into its 15's complement.

Soln. :

Subtract each digit of the given hex number from 15.

	15	15	15	
−	C	8	4	Given number
	3	7	B	15's complement

$\therefore$ 15's complement of $(C\ 8\ 4)_{16}$ is $(3\ 7\ B)_{16}$

Procedure for subtraction using 15's complement :

To perform the subtraction $(A)_{16} - (B)_{16}$, the following procedure should be followed :

Step 1 : Obtain the 15's complement of the number to be subtracted (Number B).

Step 2 : Add A16 and 15's complement of B.

Step 3 : If the carry is produced, then add it to the sum obtained in step 2 to get the final result. The result is positive and in its true form.

Step 4 : If the carry is not produced, then the addition obtained in step 2 is negative. Hence take its 15's complement to get the answer in its true form.

Scanned by CamScanner

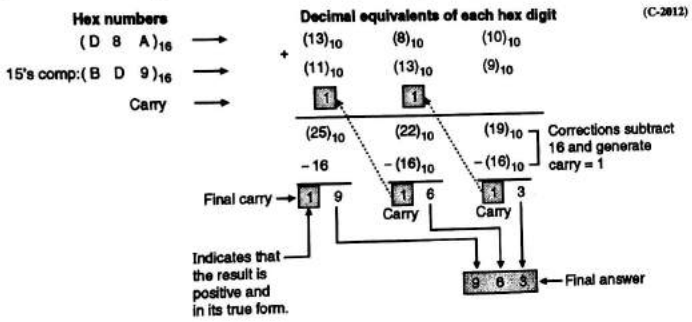
Ex. 2.7.7 : Perform the subtraction $(D8A)_{16} - (426)_{16}$ using the 15's complement.

Soln. :

Step 1 : Obtain the 15's complement of $(426)_{16}$:

	15	15	15	
−	4	2	6	Given number
	B	D	9	15's complement

Step 2 : Add $(D\ 8\ A)_{16}$ and 15's complement of $(4\ 2\ 6)_{16}$:



Result of addition = 963 with carry 1.

Step 3 : As carry = 1, add carry to the result of addition :

9 6 3	Result of addition
+ 1	Add final carry
$(9\ 6\ 4)_{16}$	Answer

The answer is positive and in its true form.

$$(D8A)_{16} - (426)_{16} = (964)_{16}$$ ...Ans.

Ex. 2.7.8 : Perform the subtraction $(426)_{16} - (D8A)_{16}$ using the 15's complement method.

Soln. :

Step 1 : Obtain the 15's complement of $(D8A)_{16}$:

	$(15)_{10}$	$(15)_{10}$	$(15)_{10}$	
−	D	8	A	Given number
	2	7	5	15's complement

Step 2 : Add $(426)_{16}$ and 15's complement of $(D8A)_{16}$:

4 2 6	
+ 2 7 5	
1 6 9 B	Result

No final carry. So answer is negative.

Step 3 : Take 15's complement of the result obtained in step 2 :

	15	15	15	
−	6	9	B	Result in step 2
	9	6	4	Final answer

$$\therefore \quad (426)_H - (D8A)_H = -(964)_H$$ ...Ans.

2.7.5 Hex Subtraction using 16's Complement :

The 16's complement of a hex number is obtained by adding 1 to the 15's complement of the given hex number. This is illustrated in the following example.

Ex. 2.7.9 : Obtain the 16's complement of $(C84)_H$.

Soln. :

Step 1 : Obtain 15's complement of $(C84)_H$:

	15	15	15	
−	C	8	4	Given number
	3	7	B	15's complement

Step 2 : Add 1 :

+			1	
	3	7	C	16's complement

Procedure for the subtraction using 16's complement :

The procedure for subtracting $(A)_{16} - (B)_{16}$ is as follows :

Step 1 : Obtain the 16's complement of number B.

Step 2 : Add $(A)_{16}$ and 16's complement of $(B)_{16}$.

Step 3 : If carry is produced, then the sum obtained in step 2 is positive and represents the answer. Discard carry.

Step 4 : If carry is not produced, then the answer is negative. Take the 16's complement of the sum obtained in step 2 to get the final result.

Ex. 2.7.10 : Perform the subtraction $(D8A)_{16} - (426)_{16}$ using 16's complement.

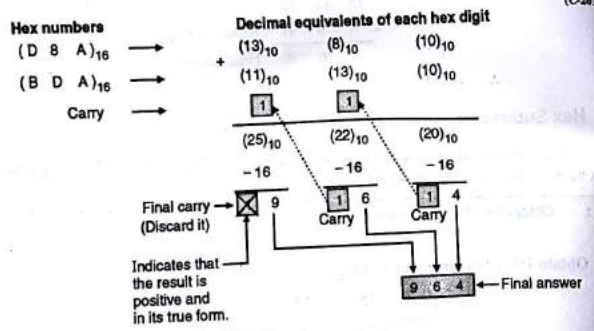
Soln. :

Step 1 : Obtain the 16's complement of $(426)_{16}$:

	15	15	15	
−	4	2	6	
	B	D	9	15's complement
+			1	Add 1
	B	D	A	16's complement

Scanned by CamScanner

Step 2 : Add $(D8A)_{16}$ and 16's complement of $(426)_{16}$: (C-2017)



$\therefore$ $(D8A)_{16} - (426)_{16} = (964)_{16}$ ...Ans.

Ex. 2.7.11 : Perform the subtraction $(426)_{16} - (D8A)_{16}$ using the 16's complement method.

Soln. :

Step 1 : Obtain the 16's complement of $(D8A)_{16}$:

	15	15	15	
−	D	8	A	
	2	7	5	15's complement
+			1	Add 1
	2	7	6	16's complement

Hence the 16's complement of $(D8A)_{16}$ is $(276)_{16}$.

Step 2 : Add $(426)_{16}$ and 16's complement of $(D8A)_{16}$:

4 2 6		$(4)_{10}$	$(2)_{10}$	$(6)_{10}$
2 7 6	+	$(2)_{10}$	$(7)_{10}$	$(6)_{10}$
Carry				
No carry produced 0		6	9	C

Result is negative.

Step 3 : Take 16's complement of $(69C)_{16}$:

	15	15	15	
−	6	9	C	
	9	6	3	15's complement
+			1	Add 1
	9	6	4	16's complement (Final Answer)

$\therefore$ $(426)_{16} - (D8A)_{16} = -(964)_{16}$ ...Ans.

2.7.6 Hexadecimal Multiplication :

If you directly multiply hex numbers the thing becomes bit complicated. Therefore it will be better if one follows following steps :

Step 1 : Convert hex to binary. (Multiplier and Multiplicand both).
Step 2 : Perform Simple binary multiplication.
Step 3 : After performing multiplication, whatever answer you get in binary, convert it to equivalent hex. This you can achieve by grouping 4 binary bits. Add extra zeros where required.

Ex. 2.7.12 : Multiply $(72)_{16}$ and $(39)_{16}$.

Soln. :

Step 1 : Convert hex to binary :

$(72)_{16} = (0\,1\,1\,1\;0\,0\,1\,0)_2 = (1\,1\,1\,0\,0\,1\,0)_2$

$(39)_{16} = (0\,0\,1\,1\;1\,0\,0\,1)_2 = (1\,1\,1\,0\,0\,1)_2$

Step 2 : Perform binary multiplication :

								1	1	1	0	0	1	0
							×		1	1	1	0	0	1
								1	1	1	0	0	1	0
							0	0	0	0	0	0	0	–
						0	0	0	0	0	0	0	–	–
					1	1	1	0	0	1	0	–	–	–
				1	1	1	0	0	1	0	–	–	–	–
			1	1	1	0	0	1	0	–	–	–	–	–
		10	10	1	1	1	1	1						
1	1	0	0	1	0	1	1	0	0	0	1	0		binary

Step 3 : Convert binary to hex :

extra zeros

$\underbrace{0001}_{1}$ $\underbrace{1001}_{9}$ $\underbrace{0110}_{6}$ $\underbrace{0010}_{2}$

$\therefore$ Ans. $= (1962)_{16}$

Scanned by CamScanner

2.7.7 Hex Division :

To perform division in hex number you have to follow steps given below :

Step 1 : Convert hex number to equivalent binary.
Step 2 : Perform binary division.
Step 3 : Convert final answer in binary i.e. Quotient and Remainder to equivalent hex.

Ex. 2.7.13 : Perform $(24)_H \div (8)_H$.

Soln. :

Step 1 : Convert both numbers into binary :

2 4 Hex 8

0010 0100 Binary 1000 (C-1710(a))

Step 2 : Perform division :

```
        100      Quotient
1000 ) 100100
       1000
        100 Remainder
```

Step 3 : Convert the answer into hex :

Quotient : 100 = $(0100)_2$ = $(4)_H$

Remainder : 100 = $(0100)_2$ = $(4)_H$

2.8 Solved University Examples :

Ex. 2.8.1 : Perform directly without converting to any other base :

1. $(BC\ 5)_H - (A2B)_H$
2. $(210.2)_4 + (312.2)_4$
3. $(56)_8 \times (45)_8$

May 11. 6 Marks

Soln. :

1. $(BC\ 5)_H - (A2B)_H$:

```
                    16 }(21)10
  B (11)10   C }(12)10   5
                2 }(3)10
- A (10)10    1          B
  ----------------------------
  (1)10      (9)10      (10)10
    ↓          ↓          ↓
    1          9          A
```

(C-2175)

$\therefore\ (BC5)_H - (A2B)_H = (19A)_H$

2. $(210.2)_4 + (312.2)_4$:

```
  2 1 0 . 2
- 3 1 2 . 2
      1
  ---------
  5 2 3 . 4
          -4
          10
  5 2 3 . 0  Final answer
```

(C-2176)

$\therefore\ (210.2)_4 + (312.2)_4 = (523.0)_4$ ...Ans.

3. $(56)_8 \times (45)_8$:

Step 1 : Convert the octal numbers to binary :

$(56)_8 = (101110)_2$

and $(45)_8 = (100101)_2$

Step 2 : Carry out the binary multiplication :

```
        101110
       ×100101
       -------
        101110
       000000×
      101110××
     000000×××
    000000××××
   101110×××××
   111111
   ----------------
   11010100110
```

Step 3 : Convert the answer into octal :

$\underbrace{011}_{3}\ \underbrace{010}_{2}\ \underbrace{100}_{4}\ \underbrace{110}_{6}$

$\therefore\ (56)_8 \times (45)_8 = (3246)_8$ ...Ans.

Ex. 2.8.2 : Perform the following directly without converting to any other base :

1. $(63)_8 \times (21)_8$
2. $(D9)_H - (80)_H$

Dec. 13. 5 Marks

Soln. :

1. $(63)_8 \times (21)_8$:

Step 1 : Convert octal number to binary :

$(63)_8 = (110011)_2$

$(21)_8 = (010001)_2$

Scanned by CamScanner

Step 2 : Perform binary multiplication :

(C-4202)

```
          1 1 0 0 1 1
        × 0 1 0 0 0 1
        -------------
          1 1 0 0 1 1
        0 0 0 0 0 0 -
      0 0 0 0 0 0 - -
    0 0 0 0 0 0 - - -
  1 1 0 0 1 1 - - - -
0 0 0 0 0 0 - - - - -
Carry     1 1
  0 0 1 1 0 1 1 0 0 0 1 1   Binary
  1   5   4   3           Octal
```

...Ans.

$\therefore\ (63)_8 \times (21)_8 = (1543)_8$

2. $(D9)_H - (80)_H$:

(C-4203)

```
Borrow  16
  A : 6   3
-
  B : 8   0
     14   3
```

Ex. 2.8.3 : Perform following without converting into other bases :

1. $(57)_8 \times (24)_8$ 2. $(312.0)_4 + (213.2)_4$

May 14, 5 Marks

Soln. :

1. $(57)_8 \times (24)_8$:

Step 1 : Convert both octal numbers into binary :

$(57)_8 = (101\ 111)_2$

$(24)_8 = (010\ 100)_2$

Step 2 : Carry out the binary multiplication :

(C-4960)

```
            1 0 1 1 1 1
          × 0 1 0 1 0 0
          -------------
            0 0 0 0 0 0
          + 0 0 0 0 0 0 X
        + 1 0 1 1 1 1 X X
      + 0 0 0 0 0 0 X X X
    + 1 0 1 1 1 1 X X X X
  + 0 0 0 0 0 0 X X X X X
Carry   1 1 1 1
  0 1 1 1 0 1 0 1 1 0 0
```

Step 3 : Convert the answer into octal :

(C-4961)

```
001 110 101 100
1   6   5   4
```

$\therefore\ (57)_8 \times (24)_8 = (1654)_8$

2. $(312.0)_4 + (213.2)_4$:

(C-4962)

```
  3 1 2 . 0
  2 1 3 . 2
  [1]
  5 3 5 . 2
  -4  -4
  (1) (1)
  1 1 3 1 . 2
```

$\therefore\ (312.0)_4 + (213.2)_4 = (1131.2)_4$ ...Ans.

Ex. 2.8.4 : Perform hexadecimal arithmetic operation : DADA + BABA.

Dec. 15, 2 Marks

Soln. :

Step 1 : Add the digits by assuming them to be decimal :

```
                Given numbers          Decimal equivalents
  D A D A       (13)10  (10)10  (13)10  (10)10
+ B A B A       (11)10  (10)10  (11)10  (10)10
      Carry      [1]     [1]     [1]
  1 9 5 9 4      (25)10  (21)10  (25)10  (20)10
                -(16)10 -(16)10 -(16)10 -(16)10
                 [1] 9   [1] 5   [1] 9   [1] 4
Final carry
                [1 9 5 9 4]
```

(C-5261)

$\therefore\ (DADA)_{16} + (BABA)_{16} = (19594)_{16}$ ...Ans.

Ex. 2.8.5 : Perform the following operations without changing the base :

1. $(314)_8 + (737)_8$ 2. $(312.40)_5 + (214.33)_5$

May 16, 4 Marks

Soln. :

1. $(314)_8 + (737)_8$:

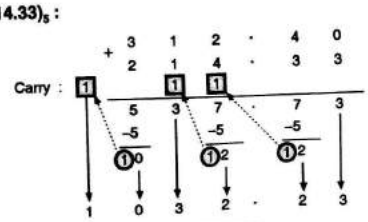
```
        3 1 4
      + 7 3 7
Carry [1]
        10 5 11
        -8  -8
        (1)2 (1)3
        1  2  5  3
```

(C-5515)

$\therefore\ (314)_8 + (737)_8 = (1253)_8$ ...Ans.

Scanned by CamScanner

2. $(312.40)_5 + (214.33)_5$:



(C-5516)

$\therefore \quad (312.40)_5 + (214.33)_5 = (1032.23)_5$ ...Ans.

Review Questions

Q. 1 Explain with example the binary addition and subtraction.

Q. 2 Give and explain the advantages of one's complement representation.

Q. 3 Explain with example 2's complement method used for subtraction.

Q. 4 What is two's complement number ?

Q. 5 Explain the binary multiplication and division with example.

Q. 6 What is two's complement number ?

Q. 7 Find out 2's complement number of following decimal number :

1. 17 2. 20 3. 107

Q. 8 Using 1's complement add + 9 and – 8 (using 8 bit format)

Q. 9 Subtract using 2's complement $(11011011)_2$ from $(0101010)_2$.

Q. 10 Subtract using 2's complement 11001 from 01101.

Q. 11 Subtract 1101101 – 11010 using 2's complement.

□□□

CHAPTER 3

Codes

Module 1

Syllabus :

Codes : Gray code, BCD code, Excess – 3 code, ASCII code, Error detection and correction : Hamming codes.

3.1 Concept of Coding :

- When numbers, letters or text characters are represented by a specific group of symbols, it is said that the number, letter or word is being **encoded.**
- And the group of symbols is called as the **code.**
- The digital data is represented, stored and transmitted as group of binary bits. Such a group of binary bits is also called as **binary code.**
- The binary codes can be used for representing the numbers as well as alphanumeric letters.

3.1.1 Advantages of Binary Codes :

1. Binary codes are suitable for the computer application.
2. The analysis and designing of digital circuits becomes easy if we use the binary codes.
3. As only 0 and 1 are being used, implementation of binary codes becomes easy.

3.1.2 Applications of Binary Codes :

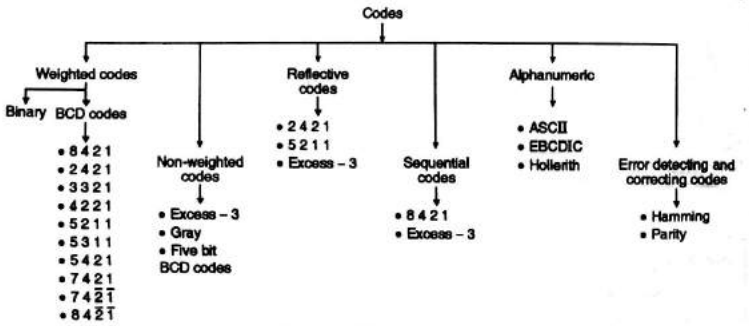
1. In digital communication.
2. In digital computers.

3.2 Classification of Codes :

Fig. 3.2.1 shows the classification of codes. The codes are broadly categorized into following six categories :

1. Weighted codes.
2. Non-weighted codes.
3. Reflective codes.
4. Sequential codes.
5. Alphanumeric codes.
6. Error detecting and correcting codes.

Scanned by CamScanner



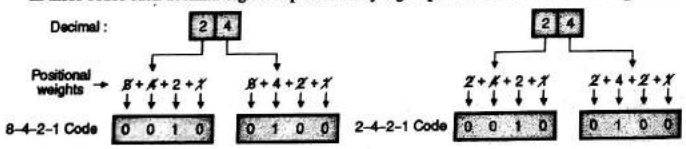
(C-72) **Fig. 3.2.1 : Classification of codes**

3.2.1 Weighted Binary Codes : MU : Dec. 15

University Questions

Q. 1 Explain in brief weighted and non-weighted codes with one example each. **(Dec. 15, 2 Marks)**

- Weighted binary codes are those codes which are based on the principle of positional weight.
- Each position of a number represents a specific weight.
- Several systems of codes are used to express the decimal digits 0 through 9. These codes have been listed in Fig. 3.2.2. The codes 8421, 2421, 3321 .... all are the weighted codes.
- In these codes each decimal digit is represented by a group of four bits as shown in Fig. 3.2.2.



(C-73) **Fig. 3.2.2**

3.2.2 Non Weighted Codes : MU : Dec. 15

University Questions

Q. 1 Explain in brief weighted and non-weighted codes with one example each. **(Dec. 15, 2 Marks)**

- The codes in which the positional weights are not assigned, are known as non weighted codes.
- The examples of non-weighted codes are excess-3 and gray codes.

3.2.3 Reflective Codes :

- A code is said to be reflective, when the code for 9 is the complement of code for 0, code for 8 is complement of code for 1, 7 for 2, 6 for 3 and 5 for 4 etc.
- Reflectivity is desirable in a code when the 9's complement is required to be found. e.g. in 9's complement subtraction which is used for BCD subtraction.
- 2421, 5211 and XS-3 are the examples of reflective codes.

3.2.4 Sequential Codes :

- A code is called as a sequential when each succeeding code word is one binary number greater than the preceding code word. For example in the BCD code the codewords are 0000, 0001, 0010 .....
- This greatly simplifies the mathematical operations to be carried out on that code.
- The 8421 and XS-3 are sequential codes.

3.2.5 Alphanumeric Codes :

- The special code designed to represent numbers as well as alphabetic characters are called as the alphanumeric codes.
- Some of these codes are capable to representing some symbols and instructions as well, in addition to the numbers and alphabetic characters.
- Examples of alphanumeric codes are : ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code) and Hollerith code.

3.2.6 Error Detecting and Correcting Codes :

- When digital data or information is transmitted from one system (such as a computer) to the other, an unwanted electrical disturbance called "Noise" gets added to it.
- The noise can force an "error" in the digital information. That means a 0 may change to 1 or a 1 can change to 0.
- To detect and correct such errors we can use some special codes which possess the capacity to detect and correct the error.
- Such codes are called as error detecting and error correcting codes. Parity can be used for error detection whereas Hamming codes can be used for error correction.

3.3 Binary Coded Decimal (BCD) Code :

- BCD is the short form of "Binary Coded Decimal." In this code each decimal digit is represented by a 4-bit binary number.
- Thus BCD is a way to express each of the decimal digits with a binary code.
- The positional weights associated to the binary bits in BCD code are 8-4-2-1 with 1 corresponding to LSB and 8 corresponding to MSB. These weights are actually 2^3, 2^2, 2^1 and 2^0 which are same as those used in the normal binary system.

Conversion from decimal to BCD :

- The decimal digits 0 to 9 are converted into a BCD, exactly in the same way as binary .
- For example decimal 4 corresponds to 0100 BCD which is same as binary. Similarly the BCD numbers corresponding to the decimal numbers 0 to 9 are exactly same as the corresponding binary numbers. This is illustrated in Table 3.3.1.

Table 3.3.1

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

← BCD numbers are same as 4 bit binary numbers →

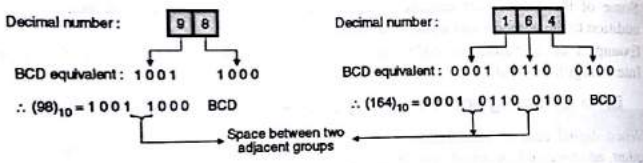
Scanned by CamScanner

Invalid BCD codes :

- With four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only the first ten of these are used (0000 to 1001).
- The remaining six code combinations ie 1010 to 1111 are invalid in BCD.

Conversion of big decimal numbers to BCD :

- To express any decimal number in BCD, simply express each decimal digit with its equivalent 4-bit BCD code as illustrated below.
- Fig. 3.3.1 shows the BCD codes for various decimal numbers.



(C-74) **Fig. 3.3.1 : Decimal to 8-4-2-1 BCD conversion**

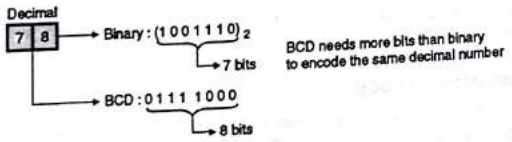
Smallest and largest number in BCD :

The smallest digit in BCD is (0000) i.e. 0 and the largest one is (1001) i.e. 9. The next number to 9 will be $(10)_{10}$ which is expressed as (0001 0000) in BCD.

3.3.1 Comparison with Binary :

1. **BCD is less efficient than binary**

Conversion of a decimal number 78 into BCD and binary is illustrated in Fig. 3.3.2.



(C-75) **Fig. 3.3.2 : BCD is less efficient than binary**

Fig. 3.3.2 illustrates that in order to encode the same decimal number, BCD needs more number of bits than binary. Hence BCD is less efficient as compared to binary.

2. BCD arithmetic is more complicated than Binary arithmetic.
3. The advantage of a BCD code is that the conversion from Decimal to BCD or vice versa is simpler.
4. Table 3.3.2 shows the comparison of binary and BCD numbers from 0 to 15 decimal.

Table 3.3.2

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

BCD and Binary numbers are same

BCD and Binary numbers are different

Packed BCD :

The BCD numbers corresponding to decimal numbers beyond 9 are called as packed BCD. Some examples of packed BCD are presented in Table 3.3.3.

Table 3.3.3 : Examples of packed BCD

Decimal	Packed BCD		
25		0010	0101
169	0001	0110	1001
523	0101	0010	0011

3.3.2 Advantages of BCD Codes :

1. It is very similar to decimal system.
2. We need to remember binary equivalents of decimal numbers 0 to 9 only.

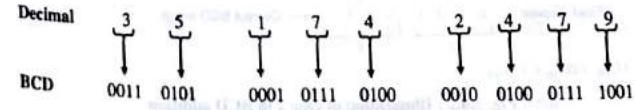
3.3.3 Disadvantages :

1. The addition and subtraction of BCD have different rules.
2. The BCD arithmetic is little more complicated.
3. BCD needs more number of bits than binary to represent the same decimal number. So BCD is less efficient than binary.

Ex. 3.3.1 : Convert the following decimal numbers to BCD :

(a) 35 (b) 174 (c) 2479

Soln. :



Scanned by CamScanner

BCD Arithmetic :

In this section we are going to understand the rules for BCD addition and subtraction.

BCD Addition :

In BCD addition we have to deal with three different situations. Assume that two 4 bit BCD numbers A and B are being added. Then the three cases to be considered are :

Case 1 : Sum is equal to or less than 9 and carry is 0.

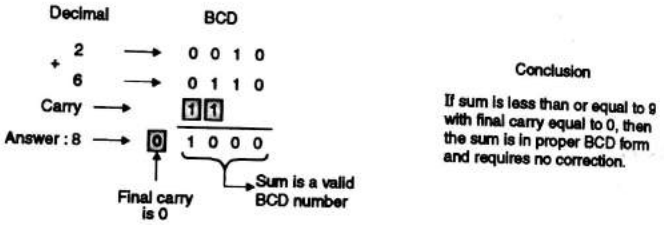
Case 2 : Sum is greater than 9 and carry is 0.

Case 3 : Sum is less than or equal to 9 but carry is 1.

The BCD addition is to be carried out in an identical manner as normal binary addition.

1 : Sum equal to or less than 9 with carry 0

The addition of $(2)_{10}$ and $(6)_{10}$ in BCD is shown in Fig. 3.4.1.



$(2)_{10} + (6)_{10} = (8)_{10}$

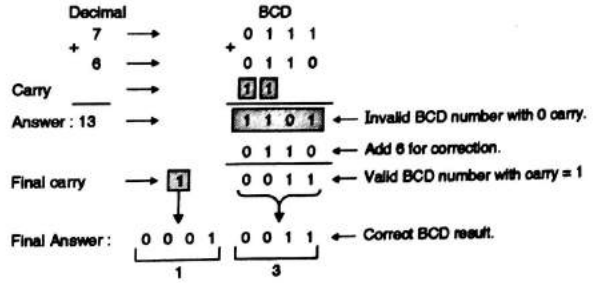
(C-76) **Fig. 3.4.1 : Illustration of case 1 in BCD addition**

2 : Sum greater than 9 but carry = 0

In this case the sum of the two BCD numbers is greater than 9 that means it is an invalid BCD number. But the final carry is 0.

So we have to correct the sum by adding decimal 6 or BCD 0110 to it. After doing this we get the correct BCD sum.

Case 2 of BCD addition is illustrated in Fig. 3.4.2.

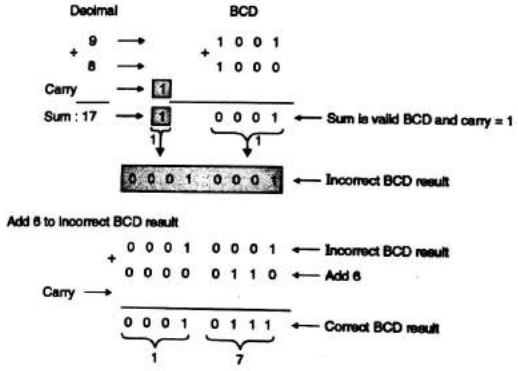


$(7)_{10} + (6)_{10} = (13)_{10}$

(C-77) **Fig. 3.4.2 : Illustration of case 2 in BCD addition**

Case 3 : Sum less than or equal to 9 but carry = 1

- Here the sum of two BCD numbers is less than or equal to 9 i.e. it is a valid BCD number, and final carry = 1.
- A nonzero carry indicates that the answer is wrong and needs correction.
- Add $(6)_{10}$ or (0110) BCD to the sum to correct the answer.
- Case 3 of BCD addition is illustrated in Fig. 3.4.3.

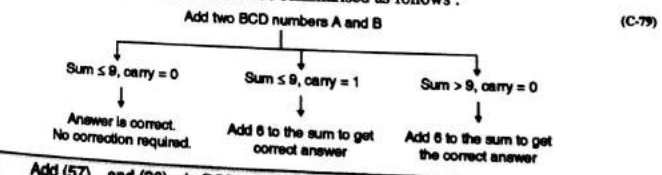


$(9)_{10} + (8)_{10} = (17)_{10}$

(C-78) **Fig. 3.4.3 : Illustration of case 3 in BCD addition**

Summary of BCD addition :

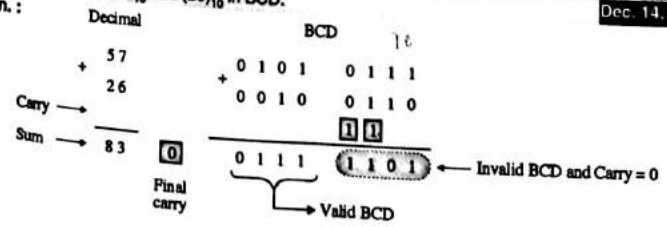
The BCD addition discussed so far can be summarised as follows :



(C-79)

Ex. 3.4.1 : Add $(57)_{10}$ and $(26)_{10}$ in BCD. **Dec. 14, 2 Marks**

Soln. :



(C-80)

Scanned by CamScanner

We have to add 6 to the sum to correct it.

```
          0 1 1 1   1 1 0 1   Incorrect sum
        + 0 0 0 0   0 1 1 0   Add 6 for correction
Carry     1 1 1 1   1
          1 0 0 0   0 0 1 1   Correct result
          8         3
```

$\therefore\ (57)_{10} + (26)_{10} = (83)_{10}$ ...Ans.

Ex. 3.4.2 : Add $(83)_{10}$ and $(34)_{10}$ in BCD.

Soln. :

```
Decimal                BCD
(83)10                 1 0 0 0   0 0 1 1
+ (34)10             + 0 0 1 1   0 0 1 0
                 Carry
(117)10          Sum :  1 0 1 1   0 1 1 1   Sum > 9, Carry = 0
                        Invalid  Valid     ∴ Correction required
                        BCD      BCD
```

Add 6 to the invalid BCD, for correction.

```
                1 0 1 1   0 1 1 1
          +     0 1 0 1   0 0 0 0   6 is added only to the invalid BCD.
Carry           1 1
                0 0 0 1   0 1 1 1
                0 0 0 1   0 0 0 1   0 1 1 1   Correct BCD result.
                1         1         7
```

$\therefore\ (83)_{10} + (34)_{10} = (117)_{10}$ ...Ans.

Ex. 3.4.3 : Add $(569)_{10}$ and $(687)_{10}$ in BCD.

Soln. :

```
(569)10          0 1 0 1   0 1 1 0   1 0 0 1
(687)10        + 0 1 1 0   1 0 0 0   0 1 1 1
                 1                 1 1 1
                 1 0 1 1   1 1 1 1   1 0 0 0 0
                 Invalid   Invalid   Valid
                 BCD       BCD       BCD with
                                     carry 1
```

(C-3060)

Add $(6)_{10}$ to each one.

```
                 1 0 1 1   1 1 1 1   0 0 0 0
               + 0 1 1 0   0 1 1 0   0 1 1 0
                 1 1 1 1   1 1
                 0 0 1 0   0 1 0 1   0 1 1 0
                 1         2         5         6   Final answer
```

(C-3061)

...Ans.

$\therefore\ (569)_{10} + (687)_{10} = (1256)_{10}$

Ex. 3.4.4 : Perform BCD addition for numbers 56 and 65. Dec. 10, 2 Marks

Soln. :

```
Decimal          BCD
  5 6            0 1 0 1   0 1 1 0
+ 6 5            0 1 1 0   0 1 0 1
                 1         1         : Carry
Sum → 1 2 1      1 0 1 1   1 0 1 1   : Sum
                 Invalid   Invalid
                 BCD       BCD
```

(C-3692)

Add 6 to the invalid BCD for correction :

```
                 1 0 1 1   1 0 1 1   Incorrect sum
                 0 1 1 0   0 1 1 0   Add 6 for correction
Carry :          1 1 1 1   1 1
                 1         0 0 1 0   0 0 0 1
                 0 0 0 1   0 0 1 0   0 0 0 1   Correct BCD result
                 1         2         1
```

(C-3693)

...Ans.

Ex. 3.4.5 : Perform BCD addition of the decimal numbers 45 and 27 May 12, 4 Marks

Soln. :

```
                 Decimal   BCD
(45)10 :                   0 1 0 0   0 1 0 1
(27)10 :                   0 0 1 0   0 1 1 1
Carry :                              1 1 1
                           0 1 1 0   1 1 0 0
                           Valid     Invalid
                           BCD       BCD
```

(C-3694)

Add 6 to invalid BCD for correction.

```
                 0 1 1 0   1 1 0 0
               + 0 0 0 0   0 1 1 0
Carry                        1 1
                 0 0 1 1 1   0 0 1 0
                 7           2
Final carry
```

(C-3695)

$\therefore\ (45)_{10} + (27)_{10} = (72)_{10}$ ...Ans.

3.4.2 BCD Subtraction :

The subtraction of two BCD numbers A and B can be performed using one of the following methods :

1. Nine's complement method.
2. Ten's complement method.

Scanned by CamScanner

1. BCD Subtraction using 9's Complement :

Nine's complement :

- The 9's complement of a BCD number can be obtained by subtracting it from 9.
- For example 9's complement of 1 is 8. The 9's complement of various digits are given in Table 3.4.1.

Table 3.4.1 : 9's complement of various decimal digits

Decimal digit	0	1	2	3	4	5	6	7	8	9
9's complement	9	8	7	6	5	4	3	2	1	0

This is similar to the subtraction using 1's complement. It is performed as follows :

Step 1 : Obtain the 9's complement of number B.
Step 2 : Add A and 9's complement of B.
Step 3 : If a carry is generated in step 2 then add it to the sum to obtain the final result. The carry is called as end around carry.
Step 4 : If carry is not produced then the result is negative hence take the 9's complement of the result.

Ex. 3.4.6 : Subtract $(3)_{10}$ from $(7)_{10}$ in BCD.

Soln. :

Step 1 : Obtain 9's complement of $(3)_{10}$:

9's complement of $(3)_{10}$ is $9 - 3 = 6$.

Step 2 : Add 7 and nine's complement of 3 :

```
                 7
              + 6     Nine's complement of 3
End around carry 1  3  Sum
              +     1  Add carry to sum.
                    4  Final result is positive and in its true form.
```

$(7)_{10} - (3)_{10} = (4)_{10}$ ...Ans.

Ex. 3.4.7 : Perform the subtraction $(4)_{10} - (7)_{10}$ using the 9's complement.

Soln. :

Step 1 : Obtain 9's complement of $(7)_{10}$:

9's complement of 7 is $(9 - 7) = (2)_{10}$.

Step 2 : Add $(4)_{10}$ and 9's complement of $(7)_{10}$:

```
                 (4)10          0100   BCD of 4
9's complement of (7)10       + 0010
                               0 0110  Result
```

Final carry is 0. Hence result is negative, hence take 9's complement of the result.

Step 3 : Take 9's complement of the result :

```
   9          1001
 - 6        - 0110   Result obtained in step 2.
   3          0011   Final answer
```

$\therefore (4)_{10} - (7)_{10} = (-3)_{10}$ ...Ans.

Ex. 3.4.8 : Perform $(83)_{10} - (21)_{10}$ using the 9's complement method.

Soln. :

Step 1 : Obtain 9's complement of $(21)_{10}$:

Subtract each digit of 21 from 9.

```
∴      99
     - 21   Given number
      78    9's complement of 21
```

Step 2 : Add $(83)_{10}$ and 9's complement of $(21)_{10}$:

```
(83)10          1000   0011
+ (78)10        0111   1000   9's complement of 21
  1     Carry
 161    Sum     1111   1011   Result is in invalid BCD form
                Invalid Invalid
                BCD     BCD
```

Step 3 : Add $(6)_{10}$ i.e. $(0110)_2$ to each invalid BCD number :

```
        1111   1011   Result obtained in step 2
      + 0110   0110   Add (6)10 for correction
        1111   11
      0 0110 1 0001
```

Carry is shifted to next higher digit

Final carry is 1 which indicates that the result is positive and in its true form.

Step 4 : Add carry to the sum obtained in step 3 :

```
Final carry   1 0110   0001   Result obtained in step 3
            +         1      End around carry.
Carry                   1
              0110     0010   Final answer positive and in its true form.
               6        2
```

$\therefore (83)_{10} - (21)_{10} = (62)_{10}$ ...Ans.

Ex. 3.4.9 : Perform $(52)_{10} - (89)_{10}$ using 9's complement.

Soln. :

Step 1 : Obtain 9's complement of $(89)_{10}$:

9's complement of $(89)_{10} = 99 - 89 = 10$

Scanned by CamScanner

Step 2 : Add $(52)_{10}$ and 9's complement of $(89)_{10}$:

	0 1 0 1	0 0 1 0	BCD of $(52)_{10}$
+	0 0 0 1	0 0 0 0	9's complement of $(89)_{10}$
Carry :	1		
Sum :	0 1 1 0	0 0 1 0	Final carry is 0, so the sum is negative.

Step 3 : Take 9's complement of the sum :

	1 0 0 1	1 0 0 1	BCD of 99
−	0 1 1 0	0 0 1 0	Sum
	1 1	1 1	
	0 0 1 1	0 1 1 1	Final answer
	3	7	

$\therefore (52)_{10} - (89)_{10} = -(37)_{10}$ ...Ans.

2. Subtraction using 10's Complement :

The 10's complement is obtained by adding 1 to the 9's complement. The 10's complement can be used to perform the BCD subtraction as follows :

Step 1 : Obtain the 10's complement of subtrahend (number to be subtracted) i.e. number B.
Step 2 : Add the minuend i.e. number A to the 10's complement of subtrahend (i.e. of B)
Step 3 : Discard carry. If carry is 1 then the answer is positive and in its true form.
Step 4 : If carry is not produced then the answer is negative. So take 10's complement to get the answer.

Ex. 3.4.10 : Perform the subtraction $(9)_{10} - (4)_{10}$ in BCD using the 10's complement.

Soln. :

Step 1 : Obtain the 10's complement of $(4)_{10}$:

9's complement of $4 = 9 - 4 = (5)_{10}$

Add 1 + 1

10's complement of 4 $(6)_{10}$

Step 2 : Add $(9)_{10}$ and 10's complement of $(4)_{10}$:

	$(9)_{10}$	1 0 0 1
+	$(6)_{10}$	0 1 0 1
	Carry	
		1 1 1 1 Invalid BCD and carry = 0
Step 3 : Add $(6)_{10}$:	+	0 1 1 0 Add $(0110)_2$ for correction.
		1 1
Discard final carry		0 1 0 1 Answer is positive and in true BCD form
		5

$\therefore (9)_{10} - (4)_{10} = (5)_{10}$ ...Ans.

Ex. 3.4.11 : Perform $(3)_{10} - (8)_{10}$ in BCD using 10's complement method.

Soln. :

Step 1 : 10's complement of $(8)_{10}$:

10's complement of $(8)_{10}$ is $(9 - 8) + 1 = 2$.

Step 2 : Add 3 and 10's complement of 8 :

	$(3)_{10}$	0 0 1 1
10's complement of 8	+	0 0 1 0
		1
		0 1 0 1 Sum is negative

Final carry is 0. Hence sum is negative and not in its true form.

Step 3 : Obtain 10's complement of the sum :

Subtract the sum from $(9)_{10}$ 1 0 0 1

− 0 1 0 1 Sum

1

0 1 0 0

Add 1 + 1

Final Result : 0 1 0 1 BCD 5

The result is negative.

$\therefore (3)_{10} - (8)_{10} = (-5)_{10}$ ...Ans.

Ex. 3.4.12 : Perform $(54)_{10} - (22)_{10}$ in BCD using 10's complement.

Soln. :

Step 1 : 10's complement of 22 :

9's complement of 22	99 − 22 =	77
Add 1		+ 1
10's complement of 22		78

Step 2 : Add $(54)_{10}$ and 10's complement of $(22)_{10}$:

	$(54)_{10}$	0 1 0 1	0 1 0 0
	10's complement of $(22)_{10}$	+ 0 1 1 1	1 0 0 0
	Carry	1 1 1	
	Invalid BCD numbers	1 1 0 0	1 1 0 0
Step 3 : Add $(0110)_2$:		0 1 1 0	0 1 1 0
	Carry	1 1 1	
	Discard final carry	0 0 1 1	0 0 1 0 Answer is positive and in true BCD form
		3	2

$\therefore (54)_{10} - (22)_{10} = (32)_{10}$ ...Ans.

Scanned by CamScanner

Ex. 3.4.13 : Perform $(22)_{10} - (54)_{10}$ in BCD using 10's complement.

Soln. :

Step 1 : 10's complement of $(54)_{10}$:

$$99 - 54 = 45 + 1 = 46$$

$\therefore$ 10's complement of 54 is 46.

Step 2 : Add $(22)_{10}$ and 10's complement of $(54)_{10}$:

$(22)_{10}$	0010	0010	
10's complement of $(54)_{10}$	+ 0100	0110	$(46)_{10}$
		11	
	0110	1000	BCD for $(68)_{10}$

Carry = 0 and answer is in valid BCD form. So answer is negative.
Take 10's complement of the answer.

Step 3 : Take 10's complement of the answer :

	1001	1001	BCD for $(99)_{10}$
−	0110	1000	Answer
	11		
	0011	0001	9's complement of answer
+		1	Add 1
		1	
	0011	0010	Final answer
	3	2	

$\therefore$ $(22)_{10} - (54)_{10} = -(32)_{10}$ ...Ans.

3.5 Non – weighted Codes :

These codes do not work of the principle of positional weights. We will consider two such codes :

1. Excess – 3 code 2. Gray code

3.6 Excess – 3 Code :

MU : May 15

University Questions

Q. 1 Write short note on Excess-3 code. **(May 15, 6 Marks)**

- Excess – 3 is also called as XS - 3 code. It is a nonweighted code used to express decimal numbers as shown in Table 3.6.1.
- The Excess-3 code words are derived from the 8421 BCD code words by adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess - 3 codes are obtained as follows :

$$\text{Decimal Number} \rightarrow \text{8421 BCD} \xrightarrow[0011]{\text{Add}} \text{Excess – 3 code}$$

- Excess – 3 codes for the single digit decimal numbers are listed in Table 3.6.1.

Table 3.6.1 : Excess – 3 codes

Decimal	BCD 8	4	2	1	Excess - 3 BCD + 0011			
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Note : Excess – 3 is a sequential code, because each succeeding code is one binary number greater than its preceeding code.

- Excess – 3 is a **sequential code** because we get any code word by adding binary 1 to its previous code word as illustrated in Table 3.6.1.
- Excess – 3 is a **self complementing code.** This is because in Excess – 3 we get the 9's complement of a number by just complementing each bit that means by replacing a 0 by 1 and 1 by 0.

Ex. 3.6.1 : Obtain the XS - 3 code for $(428)_{10}$.

Soln. :

Given number :	4	2	8	Decimal
	↓	↓	↓	
	0100	0010	1000	BCD
+ 3 →	0011	0011	0011	
	0111	0101	1011	

$\therefore$ XS - 3 equivalent : 0111 0101 1011 ...Ans.

Ex. 3.6.2 : Convert the following decimal numbers into excess-3 code :
1. $(5)_{10}$ 2. $(37)_{10}$ 3. $(247.6)_{10}$

Soln. :

Decimal to excess - 3 conversion

1. **$N = (5)_{10}$:**

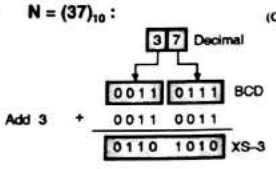
Step 1 : Write the BCD equivalent : $(5)_{10} = 0101$

Step 2 : Convert to excess-3 :

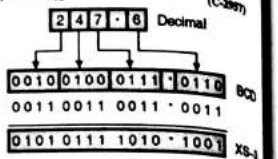
$$\text{Excess-3 of } (5)_{10} = 0101 + 0011 = 1000$$ ...Ans.

Scanned by CamScanner

2. $N = (37)_{10}$: (C-2956)



3. $N = (247.6)_{10}$: (C-2957)



3.7 Gray Code :

MU : Dec. 02, Dec. 03, May 07, May 15

University Questions

Q. 1 Write note on : Gray code. **(Dec. 02, 4 Marks)**

Q. 2 What are gray codes ? **(Dec. 03, May 07, 2 Marks)**

Q. 3 Write short note on gray code. **(May 15, 3 Marks)**

- Gray code is another non-weighted code. It is not an arithmetic code.
- It has a very special feature that only one bit in the gray code will change, each time the decimal number is incremented as shown in Table 3.7.1.
- As only one bit changes at a time, the Gray code is called as a **unit distance code.** The Gray code is a **Cyclic code.**

Table 3.7.1 (C-91)

Decimal	Binary	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0
16	1 0 0 0 0	0 0 0 0

Note :
1. In gray code only the encircled bit changes when the decimal number is incremented by 1.
2. In binary one, two, three or sometimes all the 4 bits change when the decimal number is incremented by 1.

3.7.1 Reflective Property :

The Gray code also exhibits the **reflective property.**

It is explained as follows :

- Refer Table 3.7.2. It shows that the two least significant bits for decimal 4 to 7 are the mirror images of those for decimal 3 to 0.

Table 3.7.2 (C-2378)

Decimal	Gray
0	0 0 0 0
1	0 0 0 1
2	0 0 1 1
3	0 0 1 0
4	0 1 1 0
5	0 1 1 1
6	0 1 0 1
7	0 1 0 0

The two least significant bits for decimal 4 to 7 are mirror images of those for decimal 3 to 0

- In addition to this the three least significant bits for decimal 8 through 15 are mirror images of those for decimal 7 through 0 as shown in Table 3.7.3.

Table 3.7.3 (C-2379)

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

The three LSBs for $(8)_{10}$ to $(15)_{10}$ are mirror images of those for $(7)_{10}$ through $(0)_{10}$

Scanned by CamScanner

- The gray code possesses another important property. It states that the gray code for a decimal number $(2^n - 1)$ where n = 2, 3, 4 ..., will differ from gray 0 (0000) in only 1 bit position. This is illustrated in Fig. 3.7.1.

n	Decimal number $2^n - 1$	Gray code
2	$2^2 - 1 = (3)_{10}$	0 0 1 0
3	$2^3 - 1 = (7)_{10}$	0 1 0 0

This gray code differs from gray 0(0000) only in this position

(C-92) **Fig. 3.7.1**

3.7.2 Application of Gray Code :

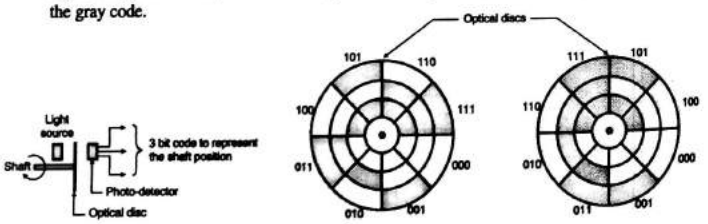
MU : May 07, Dec. 14

University Questions

Q. 1 Give application of Gray code. **(May 07, 2 Marks)**

Q. 2 Explain uses of Gray code. **(Dec. 14, 4 Marks)**

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.
- The shaft encoder consists of a light source, an optical disc and a light detector as shown in Fig. 3.7.2(a).
- Patterns of opaque and transparent segments is etched out on the optical disc. So corresponding to the black portion, the photodetector produces a "1" and corresponding to the transparent portion a "0" is produced.
- The patterns on the disc are according to the code required to be produced at the detector output.
- Fig. 3.7.2(b) shows the pattern for binary code and Fig. 3.7.2(c) shows the pattern for producing the gray code.



(a) Shaft position encoder **(b) Pattern for binary code** **(c) Pattern for gray code**

(C-93) **Fig. 3.7.2**

3.7.3 Advantages of Gray Code :

- Consider the disc which produces the binary codes. Imagine a situation in which the existing position is 111 and the position is about to change to 000.
- If one light source out of the three is slightly ahead of the others, then the detector output would be "011" instead of 111 or 000.
- Hence a 180° error in the disc position would result and the user would not even notice it, because in binary codes, any number of digits can change their values at a given instant of time.

- This problem can be eliminated by using the Gray code instead of binary. In Gray code, only one bit changes at a time. So in a 3-bit code the probability of error introduction will be reduced to 33% whereas in a 4-bit code it reduces to 25%. This is the advantage of using gray code.

3.7.4 Gray-to-Binary Conversion :

For gray to binary conversion, follow the steps given below :

Step 1 : The MSB of Gray and binary are same. So write it directly.
Step 2 : Add (mod-2 addition) binary MSB to the next bit of Gray code. Note down the result and ignore the carries.
Step 3 : Continue this process until the LSB is reached.

- Note that the addition mentioned in step 2 is a modulo 2 addition (MOD - 2). It is equivalent to an Ex-OR operations hence denoted by ⊕ sign instead of simple (+) sign.
- The rules of MOD-2 addition are as follows :

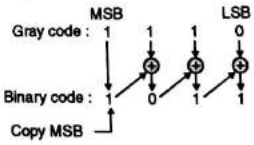
Table 3.7.4 : Rules of MOD-2 addition

0	⊕	0	=	0
0	⊕	1	=	1
1	⊕	0	=	1
1	⊕	1	=	0

- The gray to binary conversion is illustrated in the following example.

Ex. 3.7.1 : Convert 1110 gray to binary. (C-94)

Soln. :



$\therefore (1\,1\,1\,0)_{gray} = (1\,0\,1\,1)_2$

In general we can say that the conversion of a 4-bit gray number $G_3\,G_2\,G_1\,G_0$ into a 4-bit binary number $B_3\,B_2\,B_1\,B_0$ takes place as follows :

$$B_3(\text{MSB}) = G_3(\text{MSB}), \qquad B_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1, \qquad B_0 = B_1 \oplus G_0$$

3.7.5 Binary to Gray Conversion :

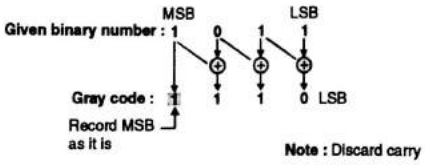
A straight binary number can be converted in gray by following the steps given below :

Step 1 : Record the MSB as it is, because the MSB of Gray is same as that of binary.
Step 2 : Add this bit to the next position, note down the sum and neglect any carry.
Step 3 : Repeat step 2.

Scanned by CamScanner

Ex. 3.7.2 : Convert binary 1011 to gray.

Soln. :



In general the conversion of binary to gray takes place as follows :

$$G_3\,(MSB) = B_3\,(MSB), \qquad G_2 = B_3 \oplus B_2$$

$$G_1 = B_2 \oplus B_1, \qquad (LSB)\ G_0 = B_1 \oplus B_0$$

- The $\oplus$ sign in the above expressions, also indicates an EX-OR (exclusive OR) function.

Ex. 3.7.3 : Represent $(29)_{10}$ into Excess-3 code and Gray code.

Dec. 10. 4 Marks. Dec. 14. 2 Marks

Soln. :

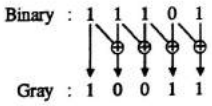
Gray code :

Step 1 : Convert $(29)_{10}$ into binary :

$$(29)_{10} = (11101)_2$$

(C-2177)

Step 2 : Binary to gray :



$$\therefore\ (29)_{10} = (10011)_{gray}$$

...Ans.

Excess code :

Given number : 2 9

Excess-3 equivalent 0101 1100

Ex. 3.7.4 : Represent $(25)_{10}$ in excess 3 and gray code.

May 12. 4 Marks

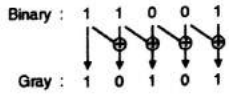
Soln. :

Step 1 : Convert $(25)_{10}$ into binary :

$$(25)_{10} = (11001)_2$$

Step 2 : Convert binary to gray :

(C-3696)



$$\therefore (25)_{10} = (10101)_{gray}$$

...Ans.

Step 3 : Convert decimal to excess – 3 :

Given number : 2 5

(C-3697)

XS-3 eqivalent : 0101 1000

$$\therefore\ (25)_{10} = (0101\ 1000)_{XS-3}$$

...Ans.

Ex. 3.7.5 : Represent $(52)_{10}$ into Excess - 3 code and Gray code.

May 15. 2 Marks

Soln. :

Step 1 : Write the BCD equivalent of $(52)_{10}$:

$$(52)_{10} = (01010010)_{BCD}$$

Step 2 : Convert decimal to excess-3 code :

$(52)_{10}$: 0101 0010

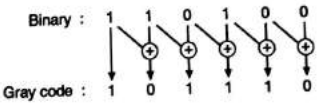
Excess 3 : 1000 0101

$$\therefore\ (52)_{10} = (1000\ 0101)_{Excess-3}$$

...Ans.

Step 3 : Convert $(52)_{10}$ to gray code :

$$(52)_{10} = (110100)_2$$



(C-5187)

$$\therefore\ (52)_{10} = (101110)_{Gray}$$

...Ans.

3.8 Alphanumeric Codes :

- A binary digit or bit can represent only two symbols as it has only two states "0" or "1". But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent :
  - 26 alphabets with capital and small letters.
  - Numbers from 0 to 9.
  - Punctuation marks and other symbols.
- **Alphanumeric codes** are the codes that represent numbers and alphabetic characters (letters). Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information.
- An alphanumeric code should at least represent 10 decimal digits and 26 letters of alphabet i.e. total 36 items.
- Therefore instead of using only single binary bits, a group of bits is used as a code to represent a symbol. The number of bits used per code word will be dependent on the total number of symbols to be represented e.g. if 5 bits are used per code word then $2^5 = 32$ combination would be possible. If the word length is increased to 8 then the number of combinations would be $2^8 = 256$, hence an 8-bit code can represent 256 symbols.
- The following three alphanumeric codes are very commonly used for the data representation :
  1. American Standard Code for Information Interchange (ASCII)
  2. Extended Binary-Coded-Decimal Interchange Code (EBCDIC)
  3. Five bit Baudot code.

Scanned by CamScanner

- ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

3.8.1 ASCII - (American Standard Code for Information Interchange) :

- ASCII is the abbreviation of American Standard Code for Information Exchange. It is a universally accepted alphanumeric (character) code.
- It is used in most computers and other electronic equipment. Most computer keyboards are standardized with ASCII. When we press a key, the corresponding ASCII code is generated which goes into the computer.
- ASCII has 128 characters and symbols. We need 7 bits to represent 128 characters. So ASCII is a 7 bit code.
- It can be considered as an 8 bit code with MSB = 0 always.
- This 8-bit code is 00 through 7F in Hexadecimal.
- The first thirty-two ASCII characters are nongraphic commands which are never printed or displayed. They are used only for the control purpose.
- Examples of control characters are "line feed" or "null" or "escape" etc.
- The remaining characters are graphic symbols which can be displayed or printed. This includes the letters of alphabets, ten decimal digits, punctuation signs and other commonly used symbols.
- The ASCII code set consists of 94 printable characters, SPACE and DELETE characters, and 32 control symbols. This is as shown in Table 3.8.1.

Table 3.8.1 : ASCII code set

Bit numbers	7	0	0	0	0	1	1	1	1
	6	0	0	1	1	0	0	1	1
	5	0	1	0	1	0	1	0	1
4321		0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SPACE	0	@	P		p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	–	=	M	]	m	}
1110	E	SO	RS		>	N	^	n	~
1111	F	SI	US	/	?	O	—	o	DEL

Ex. 3.8.1 : Using the Table 3.8.1 obtain the ASCII code word for the character "K".

Soln. : Referring to Table 3.8.1, the co-ordinates of the character "K" are (4, B). Hence the seven bit ASCII code for K is as follows :

	7	6	5	4	3	2	1
ASCII Code :	1	0	0	1	0	1	1
	4			B			

(C-97)

3.8.2 Extended ASCII Characters :

- The normal ASCII has 128 characters called as the standard characters. With 8-bit ASCII code it is possible to have a total 256 characters.
- The additional 128 characters other than the standard ones were adopted by IBM for use in their PCs. These are called as extended ASCII characters and they are used in applications other than PCs.
- The extended ASCII characters are represented by the 8 bit codes from 80 H to FFH.
- Table 3.8.2 lists the extended ASCII characters.

Table 3.8.2 : Extended ASCII characters

Symbol	Dec	Hex.	Symbol	Dec.	Hex	Symbol	Dec	Hex.	Symbol	Dec.	Hex
Ç	128	80	á	160	A0	└	192	C0	α	224	E0
ü	129	81	í	161	A1	┴	193	C1	β	225	E1
é	130	82	ó	162	A2	┬	194	C2	γ	226	E2
â	131	83	ú	163	A3	├	195	C3	Γ	227	E3
ä	132	84	ñ	164	A4	─	196	C4	π	228	E4
à	133	85	Ñ	165	A5	┼	197	C5	Σ	229	E5
å	134	86	ª	166	A6	╞	198	C6	σ	230	E6
ç	135	87	º	167	A7	╟	199	C7	µ	231	E7
ê	136	88	¿	168	A8	╚	200	C8	τ	232	E8
ë	137	89	⌐	169	A9	╔	201	C9	Φ	233	E9
è	138	8A	¬	170	AA	╩	202	CA	Θ	234	EA
ï	139	8B	½	171	AB	╦	203	CB	Ω	235	EB
î	140	8C	¼	172	AC	╠	204	CC	δ	236	EC
ì	141	8D	¡	173	AD	═	205	CD	∞	237	ED
Ä	142	8E	«	174	AE	╬	206	CE	ψ	238	EE
Å	143	8F	»	175	AF	╧	207	CF	∈	239	EF
É	144	90	□	176	B0	╨	208	D0	∩	240	F0
æ	145	91		177	B1	╤	209	D1	≡	241	F1

Scanned by CamScanner

Symbol	Dec	Hex.	Symbol	Dec.	Hex	Symbol	Dec	Hex.	Symbol	Dec.	Hex
Æ	146	92	■	178	B2	╥	210	D2	±	242	F2
ô	147	93	│	179	B3	╙	211	D3	≥	243	F3
ö	148	94	┤	180	B4	╘	212	D4	≤	244	F4
ò	149	95	╡	181	B5	╒	213	D5	⌠	245	F5
û	150	96	╢	182	B6	╓	214	D6	÷	246	F6
ù	151	97	╖	183	B7	╫	215	D7	≈	247	F7
ÿ	152	98	╕	184	B8	╪	216	D8	°	248	F8
Ö	153	99	╣	185	B9	┘	217	D9	•	249	F9
Ü	154	9A	║	186	BA	┌	218	DA	·	250	FA
¢	155	9B	╗	187	BB	│	219	DB	√	251	FB
£	156	9C	╝	188	BC	▄	220	DC	η	252	FC
¥	157	9D	╜	189	BD	▌	221	DD	²	253	FD
P_T	158	9E	╛	190	BE	▐	222	DE	■	254	FE
ƒ	159	9F	┐	191	BF	▀	223	DF		255	FF

- The extended ASCII contain characters in the following general categories :
  1. Non-English alphabetic characters.
  2. Foreign currency symbols.
  3. Greek letters.
  4. Mathematical symbols.
  5. Drawing characters.
  6. Bar graphic characters.
  7. Shading characters.

Ex. 3.8.2 : Represent the following characters in ASCII code. Express each as a bit pattern as well as in hex notation.

1. M 2. y 3. + 4. $

Soln. : All these are the standard ASCII characters.

Character	ASCII code
M	1001101 or 4D H
y	1111001 or 79 H
+	0101011 or 2B H
$	0100100 or 24 H

3.9 Code Conversions :

In this section we are going to learn the following code conversions :

1. Binary to BCD.
2. BCD to Binary.
3. BCD to Excess – 3.
4. Excess – 3 to BCD.

3.9.1 Binary to BCD Conversion :

For the binary to BCD conversion the steps to be followed are as given below :

Step 1 : Convert the Binary number to decimal.
Step 2 : Convert decimal number into BCD.

Ex. 3.9.1 : Convert the binary number $(110101)_2$ into BCD.

Soln. :

Step 1 : Conversion from binary to decimal :

Binary 1 1 0 1 0 1

$2^5 + 2^4 + 0 + 2^2 + 0 + 1$
$= 32 + 16 + 4 + 1 = (53)_{10}$

(C-2255)

Step 2 : Decimal to BCD :

5 3

BCD : (0 1 0 1 0 0 1 1)

(C-2256)

$\therefore \quad (110101)_2 = (0101\ 0011)_{BCD}$ ...Ans.

3.9.2 BCD to Binary Conversion :

Steps to be followed :

Step 1 : Convert the BCD number into decimal.
Step 2 : Convert decimal number to binary.

Ex. 3.9.2 : Convert $(0101\ 0011)_{BCD}$ into binary.

Soln. :

Step 1 : Convert BCD to decimal :

BCD → 0 1 0 1 0 0 1 1

5 3 ← Decimal

(C-2257)

$\therefore$ Equivalent decimal number is 53.

Step 2 : Convert decimal to binary :

Use the long division method for decimal to binary conversion to get,

$$\therefore \quad (53)_{10} = (110101)_2$$

$$\therefore \quad (0101\ 0011)_{BCD} = (110\ 101)_2$$

3.9.3 BCD to Excess – 3 :

For BCD to Excess – 3 conversion, follow the steps given below :

Step 1 : Convert BCD to decimal.
Step 2 : Add $(3)_{10}$ to this decimal number.
Step 3 : Convert the decimal number of step 2 into binary, to get the excess – 3 code.

Ex. 3.9.3 : Convert $(1001)_{BCD}$ to excess – 3.

Soln. :

BCD 1 0 0 1

Decimal 9
Add 3 + 3
$(12)_{10}$ $\xrightarrow[\text{to binary}]{\text{Convert}}$ $(1\ 1\ 0\ 0)_2$

(C-2258)

$\therefore (1\ 0\ 0\ 1)_{BCD} = (1\ 1\ 0\ 0)_{xs-3}$

Scanned by CamScanner

Ex. 3.9.4 : Convert $(0101\ 0011)_{BCD}$ into excess – 3.

Soln. :

BCD number → 0101 0011

Decimal →	5	3
Add 3 → to each digit	3	3
	8	6

Convert each digit to 4 bit binary

86 = 1000 0110 (C-2259)

$\therefore (0101\ 0011)_{BCD} = (1000\ 0110)_{xs-3}$

Alternative method for BCD to XS – 3 conversion :

Add $(0011)_2$ to each 4-bit BCD number to obtain the corresponding Excess – 3 number.

Ex. 3.9.5 : Convert BCD 0101 0011 to excess – 3.

Soln. :

Given BCD number	0101	0011
Add $(0011)_2$	– 0011	0011
Excess – 3 number	1000	0110

$\therefore (01010011)_{BCD} = (10000110)_{XS-3}$ ...Ans.

3.9.4 Excess – 3 to BCD Conversion :

Subtract $(0011)_2$ from each 4 bit excess – 3 digit to obtain the corresponding BCD code.

Ex. 3.9.6 : Obtain the BCD code equivalent of (1001 1010).

Soln. :

Given XS - 3 number	1001	1010
Subtract $(0011)_2$	– 0011	0011
	11	111
BCD	0110	0111

$\therefore (10011010)_{XS-3} = (01100111)_{BCD}$

Ex. 3.9.7 : Convert $(47.3)_7$ into BCD, excess-3 and gray code. **May 16, 3 Marks**

Soln. :

Step 1 : Convert base 7 to decimal :

Given number = 4 7 · 3 (C-5512)

Decimal number $= (4 \times 7^1) + (7 \times 7^0) \cdot (3 \times 7^{-1})$

$= 28 + 7 + 0.4285 = 35.4285$

$\therefore (47.3)_7 = (35.4285)_{10}$

Step 2 : Conversion to BCD :

Decimal :	3	5	·	4	2	8	5
BCD :	0011	0101	·	0100	0010	1000	0101

$\therefore (47.3)_7 = (0011\ 0101 \cdot 0100\ 0010\ 1000\ 0101)_{BCD}$ ...Ans.

Step 3 : Conversion to Excess – 3 :

Decimal :	3	5	·	4	2	8	5
Excess-3 equivalent :	6	8	·	7	5	11	8
	0110	1000	·	0111	0101	1011	1000

$\therefore (47.3)_7 = (0110\ 1000 \cdot 0111\ 0101\ 1011\ 1000)_{Excess-3}$ ...Ans.

Step 4 : Conversion to binary :

Convert the integer :

2	35	1 (LSB)
2	17	1
2	8	0
2	4	0
2	2	0
2	1	1
	0	(MSB)

$\therefore (35)_{10} = (100011)_2$ (C-5513)

Convert the fractional part :

				Base	Product	Carry
0.4285	×	2	=		0.857	0 (MSB)
0.857	×	2	=		1.714	1
0.714	×	2	=		1.428	1
0.428	×	2	=		0.856	0
0.856	×	2	=		1.712	1 (LSB)

$\therefore (0.4285)_{10} = (0.01101)_2$

$\therefore (35.4285)_{10} = (100011.01101)_2$

Step 5 : Conversion to gray :

Binary :	1	0	0	0	1	1	·	0	1	1	0	1	
Gray :	1	1	0	0	1	0	·	1	1	0	1	1	

(C-5514)

$\therefore (47.3)_7 = (100011 \cdot 01101)_2 = (110010 \cdot 11011)_{gray}$ ...Ans.

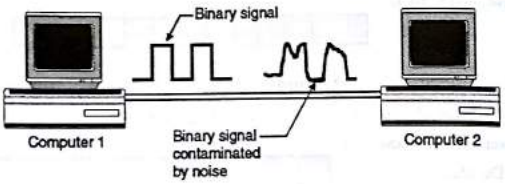
3.10 Error Detection and Correction Codes : MU : May 12

University Questions

Q. 1 Write short note on : Error detecting and correcting codes. **(May 12, 5 Marks)**

- When transmission of digital signals takes place between two systems (such as computers) as shown in Fig. 3.10.1, the signal get contaminated due to the addition of "Noise" to it.

Scanned by CamScanner



(C-1792(a)) **Fig. 3.10.1 : Noise contaminates the binary signal**

- The noise can introduce an error in the binary bits travelling from one system to the other. This means a 0 may change to 1 or a 1 may change to 0.
- These error can become a serious threat to the accuracy of the digital system. Therefore it is necessary to detect and correct the errors.

How to detect and correct errors ?

- For the detection, and / or correction of these errors, one or more than one extra bits are added to the data bits at the time transmitting.
- These extra bits are called as parity bits. They allow the detection or sometimes correction of the errors.
- The data bits alongwith the parity bits form a code word.

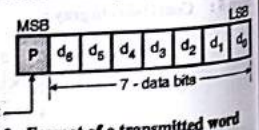
Error detection methods :

Some of the popular error detection methods are :

1. Parity checking
2. Checksum error detection
3. Cyclic redundancy check

3.10.1 Parity :

- The simplest technique for detecting errors is to add an extra bit known as parity bit to each word being transmitted.
- As shown in Fig. 3.10.2, generally the MSB of an 8-bit word is used as the parity bit and the remaining 7 bits are used as data or message bits.



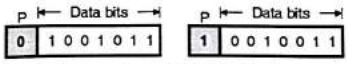
(C-1793) **Fig. 3.10.2 : Format of a transmitted word with parity bit**

- The parity of the 8-bit transmitted word can be either even parity or odd parity.
- **Even parity** means the numbers of 1's in the given word including the parity bit should be even (2, 4, 6.....).
- **Odd parity** means the number of 1's in the given word including the parity bit should be odd (1, 3, 5.....).

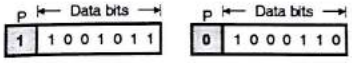
3.10.2 Use of Parity Bit to Decide Parity :

- The parity bit can be set to 0 or 1 depending on the type of parity required.
- For odd parity this bit is set to 1 or 0 at the transmitter such that the number of "1 bits" in the entire word is odd.

- For even parity this bit is set to 1 or 0 such that the number of "1 bits" in the entire word is even. This is illustrated in Fig. 3.10.3.



(C-1794) **(a) : Inclusion of a parity bit to obtain an even parity**

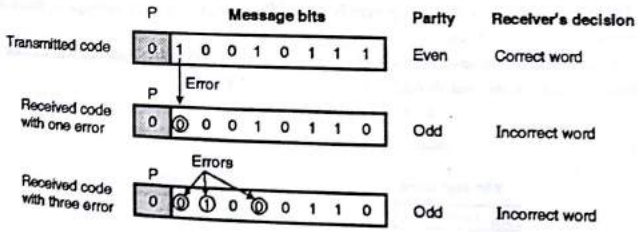


(C-1794(a)) **(b) : Inclusion of a parity bit to obtain an odd parity**

Fig. 3.10.3

How does error detection take place ?

- The parity checking at the receiver can detect the presence of an error if the parity of the received signal is different from the expected parity.
- That means if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity then the receiver can conclude that the received signal is not correct. This is as shown in Fig. 3.10.4.
- When a single error or an odd number of errors occur during transmission the parity of the code word will change. Parity of the received code word is checked at the receiver and if there is change in parity then it is understood that error is present in the received word. This is as shown in Fig. 3.10.4.
- If presence of error is detected then the receiver will ignore the received byte and request for the retransmission of the same byte to the transmitter.

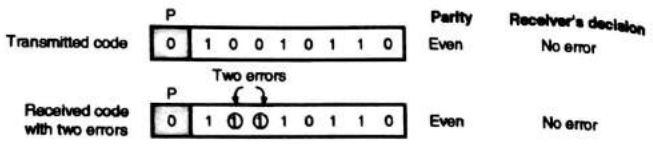


(C-99) **Fig. 3.10.4 : The receiver detects the presence of error if the number of errors is odd i.e. 1, 3, 5 ....**

When does parity checking fail to detect errors ?

If the number of errors introduced in the transmitted code is two or any even number, then the parity of the received code word will not change. It will still remain even as shown in Fig. 3.10.5 and the receiver will fail to detect the presence of errors.

Scanned by CamScanner



(C-100) **Fig. 3.10.5 : The receiver cannot detect the presence of error if the number of errors is even i.e. 2, 4, 6 ....**

Conclusions :

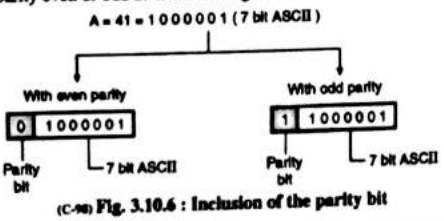
- Double or any even number of errors in the received word will not change the parity. Therefore even number of errors will be unnoticed.
- If one or odd number of errors occur then the parity of the received word will be different from the parity of transmitted signal. Thus error is noticed. However this error can neither be located nor be corrected.

Limitations of parity checking :

1. Thus the simple parity checking method has its limitations. It is not suitable for detection of multiple errors (two, four, six etc).
2. The other limitation of parity checking method is that it cannot reveal the location of erroneous bit. It cannot correct the error either.

3.10.3 Error Detection using ASCII Code :

- Basically ASCII is a 7-bit code. But in order to detect errors in data communication and processing, an eight bit is sometimes added to the ASCII character to indicate its **parity.**
- The **parity bit** is an extra bit (generally the MSB) included with a message to make the total number of 1's either even or odd.
- Consider letter A, the seven bit ASCII code for it is 41 i.e. 1000001. Now we will include a parity bit to make its parity even or odd as shown in Fig. 3.10.6.



(C-98) **Fig. 3.10.6 : Inclusion of the parity bit**

- For making the parity **even**, we should select the parity bit in such a way that the number of 1's be even. In Fig. 3.10.6 a parity bit of 0 is added for even parity and P = 1 for ensuring an odd parity.

Ex. 3.10.1 : Write the ASCII code of word "HOLE" using even parity.

Soln. : Table P. 3.10.1 shows the ASCII code of word HOLE using even parity.

Table P. 3.10.1

	P	7	6	5	4	3	2	1
H	0	1	0	0	1	0	0	0
O	1	1	0	0	1	1	1	1
L	1	1	0	0	1	1	0	0
E	1	1	0	0	0	1	0	1

Note that the parity bits are selected in order to obtain an even parity for each row (i.e. for each letter).

3.11 Hamming Code (Error Correcting Code) : MU : May 11, Dec. 12, May 14

University Questions

Q. 1 Prove that hamming code is an error detecting and correcting code.
(May 11, 8 Marks, Dec. 12, 10 Marks, May 14, 5 Marks)

- Hamming code is basically a linear block code named after its inventor. It is an error correcting code. The parity bits are inserted in between the data bits as shown in Fig. 3.11.1.
- The 7-bit Hamming code is used commonly, but the concept can be extended to any number of bits.

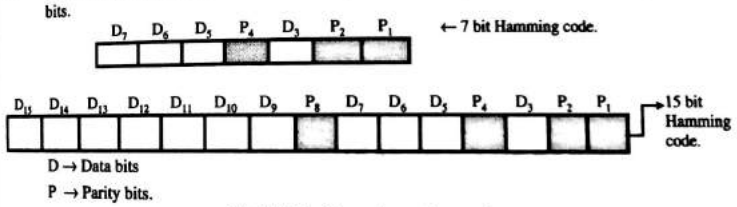


Fig. 3.11.1 : Hamming code words

- Note that the parity bits are inserted at each 2^n bit where n = 0, 1, 2, 3 ..... Thus P_1 is at $2^0 = 1$, i.e. at first bit, P_2 is at $2^1 = 2$, P_4 is at 2^2 and P_8 is at 2^3 as shown in Fig. 3.11.1.

3.11.1 7-Bit Hamming Code :

- A scientist named R.W. Hamming developed a coding system which was easy to implement. Assuming that four data bits are to be transmitted, he suggested a code word pattern shown in Fig. 3.11.2.

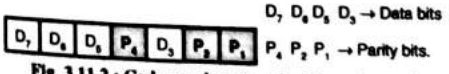


Fig. 3.11.2 : Code word pattern for Hamming code

- The D bits in Fig. 3.11.2 are data bits, whereas P bits are parity bits. The parity bits P_1, P_2, P_4 are adjusted in a particular way as explained below.

3.11.2 Minimum Number of Parity Bits :

- Table 3.11.1(a) gives a listing of minimum number of parity bits needed for various ranges of "m" information bits.

Scanned by CamScanner

Table 3.11.1(a) : Number of parity bits to be used

Number of information bits	Number of parity bits
2 to 4	3
5 to 11	4
12 to 26	5
27 to 57	6
58 to 120	7

3.11.3 Deciding the Values of Parity Bits :

Table 3.11.1 indicates which bit positions are associated with each parity bit in order to establish required parity (even or odd) over the selected bits positions.

Table 3.11.1

Parity Bit	Bits to be checked
P_1	1,3,5,7,9,11,13,15,.....
P_2	2,3,6,7,10,11,14,15, ...
P_4	4,5,6,7,12,13,14,15, ...
P_8	8,9,10,11,12,13,14,15, ...

3.11.4 Deciding the Parity Bits for a 7 Bit Code :

Selection of P_1 :

P_1 is adjusted to 0 or 1 so as to establish even parity over bits 1,3,5 and 7 i.e. P_1, D_3, D_5 and D_7.

7	6	5	4	3	2	1	→ Consider bits 1,3,5,7 for P_1
7	6	5	4	3	2	1	→ Consider bits 2,3,6,7 for P_2
7	6	5	4	3	2	1	→ Consider bits 4,5,6,7 for P_4

Selection of P_2 :

P_2 is adjusted to 0 or 1 so as to set even parity over bits 2,3,6 and 7 (P_2, D_3, D_6 and D_7).

Selection of P_4 :

P_4 is adjusted to 0 or 1 so as to set even parity over bits 4, 5, 6 and 7 (P_4, D_5, D_6 and D_7).

The selection of parity bits will be clear after solving the following example.

Ex. 3.11.1 : A bit word 1 0 1 1 is to be transmitted. Construct the even parity seven-bit Hamming code for this data. **May 15, 2 Marks**

Soln. :

Step 1 : The code word format :

The seven bit Hamming code format is shown in Fig. P. 3.11.1. Given bit word = 1 0 1 1

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		1		

To be decided

Fig. P. 3.11.1

Step 2 : Decide P_1 :

P_1 sets the parity of bits P_1, D_3, D_5 and D_7. As $D_7 D_5 D_3$ = 1 1 1 we have to set P_1 = 1 in order to have the even parity.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		1		1

Set P_1 = 1 to have the even parity of P_1 D_3 D_5 D_7

Step 3 : Decide P_2 :

P_2 is set to have the even parity of P_2, D_3, D_6 and D_7. But D_3, D_6, D_7 = 1 0 1 hence set P_2 = 0.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		1	0	1

Set P_2 = 0 to have even parity of P_2 D_3 D_6 and D_7

Step 4 : Decide P_4 :

P_4 is set to have the even parity of P_4, D_5, D_6 and D_7. But D_5 D_6 D_7 = 1 0 1, hence set P_4 = 0.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	1	0	1

← Complete code word

P_4 = 0 to have even parity of P_4 D_5 D_6 D_7

Ex. 3.11.2 : Encode the data bits 0 1 0 1 into a seven bit even parity Hamming code.

Soln. :

Step 1 :

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	0		1		

Step 2 : Select P_1 for P_1 D_3 D_5 D_7 :

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	0		1		1

→ P_1 = 1 for P_1 D_3 D_5 D_7 = 1 1 0 0

Scanned by CamScanner

Step 3 : Select P_2 for $P_2 D_3 D_6 D_7$:

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	0		1	0	1

→ $P_2 = 0$ for $P_2 D_3 D_6 D_7 = 0110$

Step 4 : Select P_4 :

D_7	D_6	D_5	P_4	D_3	D_2	D_1
0	1	0	1	1	0	1

Set $P_4 = 1$ to have $P_4 D_5 D_6 D_7 = 1010$

Hence the complete 7-bit Hamming code word is as shown below.

0	1	0	1	1	0	1

← Complete code word

- The concepts developed till now can be applied to larger hamming code words as well.

Ex. 3.11.3 : Write the hamming code for 1010.

May 10. 3 Marks. May 11. 4 Marks. May 14. 5 Marks. Dec. 15. 2 Marks

Soln. :

Step 1 : The code word format :

The seven bit Hamming code format is as follows :

Given bit word = 1010

D_7	D_6	D_5	D_4	D_3	D_2	D_1
1	0	1		0		

(C-3698)

To be decided

Step 2 : Decide P_1 :

P_1 is set to have even parity of $P_1 D_3 D_5 D_7$

But $D_7 D_5 D_3 = 110$ hence set $P_1 = 0$.

D_7	D_6	D_5	D_4	D_3	P_2	P_1
1	0	1		0		0

Set $P_1 = 0$ to have even parity of $P_1 D_3 D_5 D_7$ (C-3699)

Step 3 : Decide P_2 :

P_2 is set to have even parity of $P_2 D_3 D_6$ and D_7. But $D_3 D_6 D_7 = 001$ hence set $P_2 = 1$

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		0	1	0

(C-3700)

Set $P_2 = 1$ to have even parity of $P_2 D_3 D_6$ and D_7

Step 4 : Decide P_4 :

P_4 is set to have even parity of $P_4 D_5 D_6 D_7$. But $D_5 D_6 D_7 = 101$. Hence set $P_4 = 0$.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	0	1	0

← Complete code word

Set $P_4 = 0$ to have even parity of $P_4 D_5 D_6 D_7$

(C-3701)

Ex. 3.11.4 : Write Hamming code for number 0111.

Dec. 10. 2 Marks

Soln. : Assume even parity hamming code.

Step 1 :

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	1		1		

Step 2 : Select P_1 for $P_1 D_3 D_5 D_7$:

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	1		1		0

→ $P_1 = 0$ for $P_1 D_3 D_5 D_7 = 0110$

Step 3 : Select P_2 for $P_2 D_3 D_6 D_7$:

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	1		1	0	0

$P_2 = 0$ for $P_2 D_3 D_6 D_7 = 0110$

Step 4 : Select P_4 for $P_4 D_5 D_6 D_7$:

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	1	0	1	0	0

$P_4 = 0$ for $P_4 D_5 D_6 D_7 = 0110$

Hence the complete 7-bit hamming code word is as shown below

0	1	1	0	1	0	0

← Complete code word

Ex. 3.11.5 : Obtain odd parity Hamming code for 1011.

May 16. 2 Marks

Soln. :

Data word : 1011

Format of 7-bit hamming code is as follows :

	7	6	5	4	3	2	1
Codeword :	1	0	1	P_4	1	P_2	P_1

Step 1 : Decide P_1 :

Consider bits 1, 3, 5 and 7. They are 111 P_1

So for odd parity set $P_1 = 0$

$\therefore P_1 = 0$

Scanned by CamScanner

Step 2 : Decide P_2 :

Consider bits 2, 3, 6 and 7. They are 101 P_2

So for odd parity set $P_2 = 1$ $\therefore$ $P_2 = 1$

Step 3 : Decide P_4 :

Consider bits 4, 5, 6 and 7. They are 101 P_4

So for odd parity set $P_4 = 1$ $\therefore$ $P_4 = 1$

Obtained codeword is as follows :

	7	6	5	4	3	2	1
Codeword =	1	0	1	1	1	1	0

3.11.5 Detection and Correction of Errors :

- The Hamming coded data is now transmitted. At the receiver it is decoded to get the data back.
- The bits (1, 3, 5, 7) , (2, 3, 6, 7) and (4, 5, 6, 7) are checked for even parity.
- If all the 4-bit groups mentioned above possess the even parity then the received code word is correct i.e. it does not contain errors.
- But if the parity is not even then error exists. Such an error can be located by forming a three bit number out of the three parity checks. This process becomes clear by solving the example given below.

Ex. 3.11.6 : If the 7-bit Hamming code word received by a receiver is 1 0 1 1 0 1 1. Assuming the even parity state whether the received code word is correct or wrong. If wrong, locate the bit in error.

Soln. :

	7	6	5	4	3	2	1
Received code word :	1	0	1	1	0	1	1
				P_4		P_2	P_1

Step 1 : Check the bits 4, 5, 6 and 7 :

$P_4\ D_5\ D_6\ D_7 = 1\ 1\ 0\ 1 \rightarrow$ Odd parity.

$\therefore$ Error exists here.

$\therefore$ Put $P_4 = 1$ in the 4's position of the error word.

Step 2 : Analyze bits 2, 3, 6 and 7 :

$\therefore P_2\ D_3\ D_6\ D_7 = 1\ 0\ 0\ 1 \rightarrow$ Even parity so no error.

Hence put $P_2 = 0$ in the 2's position of the error word.

Step 3 : Check the bits 1, 3, 5, 7 :

$\therefore P_1\ D_3\ D_5\ D_7 = 1\ 0\ 1\ 1 \rightarrow$ Odd parity so error exists.

Hence put $P_1 = 1$ in the 1's position of the error word.

Step 4 : Write the error word :

Error word E = | P_4 | P_2 | P_1 |

1's position

2's position

4's position

Substituting the values of P_4 , P_2 and P_1 obtained in steps 1, 2 and 3 we get

E = | 1 | 0 | 1 |

$= (5)_{10}$

Hence bit 5 of the transmitted code word is in error.

7	6	5	4	3	2	1
1	0	1	1	0	1	1

Incorrect bit.

Step 5 : Correct the error :

Invert the incorrect bit to obtain the correct code word as follows :

Correct code word = [1 0 0 1 0 1 1]

Ex. 3.11.7 : A seven bit Hamming code is received as 1 1 1 0 1 0 1. What is the correct code ? Assume the parity to be even.

Soln. :

7	6	5	4	3	2	1	
1	1	1	0	1	0	1	← Received code word.

Step 1 : Check bits 4, 5, 6,7 : Odd parity hence error $\therefore P_4 = 1$

Step 2 : Check bits 2, 3, 6, 7 : Odd parity, hence error $\therefore P_2 = 1$

Step 3 : Check bits 1, 3, 5, 7 : Even parity, hence no error $\therefore P_1 = 0$

$\therefore$ Error word E =

P_4	P_2	P_1
1	1	0

Step 4 : Decimal equivalent of $E = 1\ 1\ 0 = (6)_{10}$:

$\therefore$ 6th bit in the received code word is incorrect. So invert it.

$\therefore$ Correct code word =

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	1	0	1

Inverted bit.

- This concept of error detection and correction can be extended to the longer hamming code words as well.
- The following example demonstrates this.

Ex. 3.11.8 : A receiver received the following Hamming code 0011100101101 with odd parity. Find the error in the received code and give the corrected data.

Soln. :

Received codeword : 0011100101101, Type of parity : odd

13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	1	P_8	0	1	0	P_1	1	P_0	1

(C-1862)

Scanned by CamScanner

Step 1 : Check bits 8, 9, 10, 11, 12, 13 :

001110 → odd parity so no error

$\therefore P_8 = 0$

Step 2 : Check bits 4, 5, 6, 7, 12, 13 :

The bits are 0 0 0 1 0 1 → even parity. ∴ error exists.

∴ Make $P_4 = 1$

Step 3 : Check bits 2, 3, 6, 7, 10, 11 :

The bits are 1 1 0 1 1 0 → even parity. ∴ error exists.

∴ Make $P_2 = 1$

Step 4 : Check bits 1, 3, 5, 7, 9, 11, 13 :

The bits are 0110011 → even parity. ∴ error exists.

∴ Make $P_1 = 1$

Step 5 : Write the error word : (C-1803)

Error word = E = | 0 | 1 | 1 | 1 |
P_8 P_4 P_2 P_2

Step 6 : Obtain correct codeword :

$E = 0111 = (7)_{10}$

So the seventh bit in the received codeword is incorrect so invert it from 0 to 1.

Hence the correct codeword is,

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

↑ Inverted seventh bit

Ex. 3.11.9 : A 7 bit even parity hamming code is received as 1000010. Correct it for any errors and extract 4 bit data. **Dec. 16. 4 Marks**

Soln.

Received codeword : 7 6 5 4 3 2 1

| 1 | 0 | 0 | 0 | 0 | 1 | 0 |

Step 1 : Check bits 4, 5, 6, 7 = 0001, odd parity hence error ∴ $P_4 = 1$

Step 2 : Check bits 2, 3, 6, 7 = 1001, even parity hence no error ∴ $P_2 = 0$

Step 3 : Check bits 1, 3, 5, 7 = 0001, odd parity hence error ∴ $P_1 = 1$

∴ Error word E = | 1 | 0 | 1 |
P_4 P_2 P_1

Decimal equivalent of error word is $(5)_{10}$. Hence the fifth bit in the received code word is incorrect. Hence invert the fifth bit to get the correct codeword as follows :

Correct codeword : | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

↑ Fifth bit inverted

3.11.6 Limitation of Hamming Code :

The main limitation of hamming code is that it can detect and correct only one error.

Ex. 3.11.10 : What is the limitation of Hamming codes ? Generate the Hamming Code for the data 111011001 with even parity.

Soln. :

For limitations of Hamming codes refer section 3.11.6.

Data number : 1 1 1 0 1 1 0 0 1

Step 1 : Number of message bits is 9. So we need to add 4 parity bits in the codeword.

The parity bits will be at the positions 1, 2, 4, and 8 as shown below : (C-2059)

D_{13}	D_{12}	D_{11}	D_{10}	D_9	P_8	D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	1	1	0	1		1	0	0		1		

Step 2 : Select P_1 for P_1 D_3 D_5 D_7 D_9 D_{11} D_{13} :

Parity needs to be even parity (C-2060)

D_{13}	D_{11}	D_9	D_7	D_5	D_3	P_1
1	1	1	1	0	1	

For even parity P_1 should be 1. $\therefore P_1 = 1$

Step 3 : Select P_2 :

To select P_2 we have to consider the bits in positions 2, 3, 6, 7, 10 and 11

$\therefore 10101P_2 \rightarrow P_2 = 1$ $\therefore P_2 = 1$

Step 4 : Select P_4 :

For P_4, we have to consider the bits in the following positions 4, 5, 6, 7, 12, 13 and select the value of P_4 for even parity.

$\therefore 11100P_4$ $\therefore P_4 = 1$

Step 5 : Select P_8 :

To select P_8, consider the bit in following positions 8, 9, 10, 11, 12, 13 and select P_8 for even parity

$11101P_8$ $\therefore P_8 = 0$

So the codeword is as follows :

(C-1806) Codeword

1	1	1	0	1	P_8 0	1	0	0	P_4 1	1	P_2 1	P_1 1

Review Questions

Q. 1 What is BCD code ?

Q. 2 What is excess-3 code ?

Q. 3 Write short note on ASCII code.

Scanned by CamScanner

Q. 4 What are the different types of codes used in digital systems ? Explain them.

Q. 5 Give four comparison between BCD code and Gray code.

Q. 6 Differentiate between Binary and Gray code.

Q. 7 Distinguish between Excess-3 code and Gray code.

Q. 8 Explain the rules of BCD addition.

Q. 9 What is error detection and correction ? Why is it necessary ?

Q. 10 Explain the use of parity for error detection.

Q. 11 When does parity technique fail to detect errors in the received code words ?

Q. 12 Explain Hamming codes. What are the limitations of Hamming code ?

CHAPTER 4

Module 2

Logic Gates and Boolean Algebra

Syllabus :

Theorems and properties of Boolean algebra, Boolean functions, Boolean function reduction using Boolean laws, Canonical forms, standard SOP and POS form, Basic digital gates : NOT, AND, OR, NAND, NOR, EX-OR, EX-NOR, Positive and negative logic.

4.1 Introduction :

- In the decimal system which is used in our day-to-day life, the arithmetic operations such as addition, subtraction, multiplication, division, square, square root etc. are used to solve arithmetic equations.
- In 1850s, the Irish mathematician George Boole developed a mathematical system for formulating logic statements with symbols so that problems can be written and solved in a manner similar to ordinary algebra.
- This is called Boolean algebra and it is extremely useful in the design and analysis of digital systems. Boolean algebra is used to write or simplify the logical expressions.

4.1.1 Basic Logical Operations (Logic Variables) :

- To solve or simplify the logical expressions, used in digital circuits we need to use "logical operators". The three main logic operators are :

1. AND operator. 2. OR operator. 3. NOT operator (Inverter).

4.1.2 Logic Gates :

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

Truth table :

- The operation of a logic gate or a logic circuit can be best understood with the help of a table called "Truth Table". The truth table consists of all the possible combinations of the inputs and the corresponding state of output produced by that logic gate or logic circuit.

Scanned by CamScanner

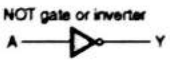
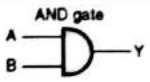
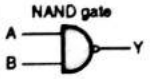
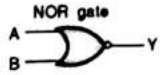
oolean expression :

The relation between the inputs and the outputs of a gate can be expressed mathematically by means of the **Boolean Expression.** Let us now discuss the operation of various logic gates.

1.3 Gates, Symbols and Boolean Expression :

In order to understand Boolean algebra , we need to use the gates. So the symbols and Boolean pressions should be known to us. Table 4.1.1 gives information.

(C-196) **Table 4.1.1 : Various logic gates**

Sr. No.	Name of gate	Boolean Expression	Logical Operation
1.	NOT gate or inverter 	$Y = \bar{A}$	Inversion
2.	AND gate 	$Y = AB$	Logical multiplication
3.	OR gate	$Y = A + B$	Logical addition
4.	NAND gate 	$Y = \overline{AB}$	NOT AND
5.	NOR gate 	$Y = \overline{A + B}$	NOT OR
6.	Exclusive OR	$Y = A \oplus B$	Addition / Subtraction
7.	Exclusive NOR	$Y = \overline{A \oplus B}$	NOT EXOR

Note : A and B are the inputs whereas Y denotes the output.

4.2 Basic Definitions :

MU : Dec. 14

University Questions

Q. 1 State the Boolean algebra laws used in K-map simplification. **(Dec. 14, 5 Marks)**

- In order to analyze and design digital circuits it is necessary to express them mathematically. In order to do so we need to use the Boolean algebra.
- Boolean algebra can be defined with the help of the following :
  1. A set of elements 2. A set of operators
  3. A number of axioms or postulates which have not been proved yet they are very useful in practice.
- A set of elements is defined as the collection of objects which have some properties common among them. If S is a set and A and B are objects then $A \in S$ denotes that A is the member of set S. On the other hand $B \notin S$ is an indication that B is not the member of set S.
- A set is specified as follows :

$$S = \{ A, B, C, D \}$$

This shows that A, B, C and D are the members of set S.

- A **binary operator** is a rule which is assigned to a pair of elements from S. For example, if $A \cdot B = C$ then $\cdot$ is the logic operator which operates on the pair A, B to produce C.

4.2.1 Postulates :

- The postulates of a mathematical system such as the Boolean system are the basic assumptions from which we can derive or formulate the rules, theorems and properties of the system.
- The most important postulates used in Boolean system are as follows :
  1. Closure 2. Associative law
  3. Commutative law 4. Identity element
  5. Inverse 6. Distributive law.

1. Closure :

A set S is said to be closed for a particular binary operator if for every pair of elements of S, that binary operator specifies a rule to obtain a unique element of S.

2. Associative law :

For a given set "S" a binary operator * can be said to be associative if the following equation is satisfied.

$$(A * B) * C = A * (B * C) \quad \text{for all } (A, B, C) \in S \qquad \ldots(4.2.1)$$

Note that * can be an operator such that · (product) or + (addition) etc.

3. Commutative law :

For a given set "S" a binary operator * is said to be commutative, if the following equation is satisfied.

$$A * B = B * A \quad \text{for all } A, B \in S \qquad \ldots(4.2.2)$$

4. Identity element :

- A set S is said to be having an identity element with respect to a binary operation * on S, if there is an element x in set S that satisfies the following property.

Scanned by CamScanner

$$x * A = A * x = A \quad \text{for every } A \in S \qquad \ldots(4.2.3)$$

- The example of identity element is 0 for the + operation on the set of integers (I).

$$I = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$$

$$\therefore \quad A + 0 = 0 + A = A \quad \text{for every } A \in I.$$

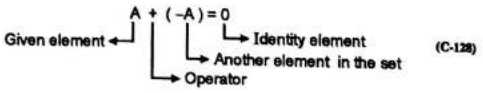
Thus for the set of integers (I) the element "0" acts as an identity element.

5. Inverse :

- A set S having an identity element x with respect to a binary operator * is said to have an inverse if for every A which belongs to S, there exists another element B which also belongs to S such that,

$$A * B = x$$

- Take the example of set of integers (I) for which 0 is the identity element with respect to + operation that means x = 0 and * = +. Hence the inverse of an element A would be – A because only then the condition A * B = x is satisfied as follows :



(C-128)

6. Distributive law :

If * and · are two binary operators working on a set S, then * is said to be distributive over · if the following condition is satisfied.

$$A * (B \cdot C) = (A * B) \cdot (A * C) \qquad \ldots(4.2.4)$$

Conclusions :

- **Field** is an example of an algebraic structure. It is a set of elements along with binary operators. Each of these possess properties 1 to 5 mentioned above. The logic operators jointly possess the 6th property mentioned above.
- The **field of real numbers** is obtained by combining the set of real numbers and the two binary operators + and ·.
- This field of real numbers is the basis for arithmetic and algebra.
- The operators and postulates of such a field have the following meanings :
  1. We define the binary operator + as addition.
  2. The identity element for + operator is 0.
  3. We define the additive inverse as subtraction.
  4. The binary operator · is defined as multiplication.
  5. The identity element for · operator is 1.
  6. We define the multiplicative inverse of A as 1/A and it is called as division.
  7. The only distributive law that is applicable is that of · over +. That means,

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

4.3 Axiomatic Definition of Boolean Algebra :

Boolean algebra is used to analyze and simplify the digital (logic) circuits.

Since it uses only the binary numbers i.e. 0 and 1 it is also called as "Binary Algebra", or "Logical Algebra".

Rules in boolean algebra :

There are some rules to be followed while using a Boolean algebra, these are :

1. Variables used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
2. Complement of a variable is represented by a overbar (–). Thus complement of variable B is represented as $\bar{B}$. Thus if B = 0 then $\bar{B} = 1$ and if B = 1 then $\bar{B} = 0$.
3. ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
4. Logical ANDing of the two or more variables is represented by writing a dot between them such as A · B · C · D · E. Sometimes the dot may be omitted like ABCDE.

4.3.1 Boolean Postulates and Laws :

- Boolean algebra is an algebraic structure, which is defined by a set of elements and two binary operators namely (+) and (·) if and only if it satisfies the postulates given below :

1(a) Closure with respect to operator (+) :

When operator (+) i.e. OR is used over two binary elements in a given set, it produces a unique binary element e.g. 1 + 0 = 1, 0 + 0 = 0.

1(b) Closure with respect to operator (·) :

When operator (·) i.e. AND is used over two binary elements in a given set, it produces a unique binary element e.g. 1 · 0 = 0, 1 · 1 = 1 etc.

2(a) An identity element with respect to (+), designated by 0 :

This is because for any A, the following expression is always true.

$$A + 0 = 0 + A = A$$

2(b) An identity element with respect to (·), designated by 1 :

This is because for any A the following expression is always true.

$$A \cdot 1 = 1 \cdot A = A.$$

3(a) Commutative law with respect to (+) :

$$A + B = B + A$$

3(b) Commutative law with respect to (·) :

$$A \cdot B = B \cdot A$$

4(a) (·) is distributed over (+) :

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

4(b) (+) is distributed over (·) :

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

5. For every binary element A, there exists an element called complement of A (it is denoted by $\bar{A}$) such that,

$$A + \bar{A} = 1 \text{ and } A \cdot \bar{A} = 0$$

6. Note that if A = 0 then $\bar{A} = 1$ and if A = 1 then $\bar{A} = 0$.
In a set of binary elements, there always exist atleast two elements A and B such that A ≠ B. If A = 0 then B = 1 and vice versa.

Scanned by CamScanner

3.2 Differences between Boolean and Ordinary Algebra :

If we compare Boolean algebra with arithmetic and conventional algebra in the field of real numbers, then the following differences are observed :

1. The Huntington's postulates do not include the associative law. But this law is applicable to Boolean algebra and it can be derived for both the operators (+ and ·) from the other postulates.
2. The following distributive law of + over · is valid for the Boolean algebra. But it is not valid for ordinary algebra.

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

3. In Boolean algebra, there are no subtraction or division operations in Boolean algebra. These are present for ordinary algebra.
4. The operator called **Complement** is present for the Boolean algebra. But it is not available for the ordinary algebra.
5. The ordinary algebra operates on the real numbers, which consists of infinite elements (integers, negative numbers, fractions etc). But Boolean algebra works upon only two elements 0 and 1.

4.4 Two Valued Boolean Algebra : MU : May 05, Dec. 06, Dec. 11

University Questions

Q. 1 State the Boolean algebra laws used in K-map simplification. (May 05, 4 Marks)

Q. 2 State distribution laws for simplification of Boolean equation and prove it. (Dec. 06, 3 Marks)

Q. 3 State and explain associative and distributive law for Boolean equation. (Dec. 11, 2 Marks)

- It is possible to formulate many Boolean algebras on the basis of elements we choose and the rules of operation.
- In this book we are going to deal with only the two valued Boolean algebra. It is called two valued because it has only two elements 0 and 1 with binary operators AND (·), OR (+) and NOT (inversion).
- The following tables show the truth tables for these operators.

Table 4.4.1(a) : AND operator

Inputs		Output
A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

Table 4.4.1(b) : OR operator

Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Table 4.4.1(c) : NOT operator

Input	Output
A	$\bar{A}$
0	1
1	0

- We are now going to demonstrate that Huntington postulates are valid for the two valued Boolean Algebra. Let S = {0, 1} and the operators be +, · and NOT.

1. **Closure :**

The closure is obvious from Tables 4.4.1(a), (b) and (c) which shows that the result of any operation (AND, OR, NOT) is either 0 or 1, and we know that 0, 1 ∈ S.

2. **Identity Elements :**

From Tables 4.4.1(a) and (b), we can write that,

$$0 + 0 = 0 \qquad 0 + 1 = 1 + 0 = 1$$
$$1 \cdot 1 = 1 \qquad 1 \cdot 0 = 0 \cdot 1 = 0$$

From these expressions, two identity elements are established which are 0 for + and 1 for · as defined by postulate 2.

3. **The commutative laws :**

The commutative laws are

$$0 + 1 = 1 + 0 \quad \text{and} \quad 0 \cdot 1 = 1 \cdot 0$$

These laws are satisfied due to the symmetry of Tables 4.4.1(a) and (b). This is illustrated by using the following truth table.

Table 4.4.2 : Verification of the commutative law

Inputs		A · B	B · A	A + B	B + A
A	B				
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	1	1	1

A · B = B · A A + B = B + A

4. **The distributive law :**

(a) The distributive law of $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ can be proved by using the truth tables as shown below.

Table 4.4.3 : Verification of the distributive law

Inputs			B + C	A · (B + C)	AB	AC	AB + AC
A	B	C					
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

A · (B + C) = AB + AC

(b) The distributive law of + over · can also be proved using the truth tables in the similar way.

Scanned by CamScanner

5. **Inverse :**

From Table 4.4.1(b) it is easily seen that,

$A + \bar{A} = 1$ i.e. $0 + 1 = 1$ and $1 + 0 = 1$

And from Table 4.4.1(a) it can be shown that,

$A \cdot \bar{A} = 0$ i.e. $0 \cdot 1 = 0$ and $1 \cdot 0 = 0$

This verifies postulate 5.

6. Postulate 6 says that there exists at least two elements A and B such that $A \neq B$. As Boolean algebra has two distinct elements 0 and 1 (A and B), this postulate is getting satisfied because $0 \neq 1$ $(A \neq B)$,

7. **Associative Law :**

- This law states that the order in which the logic operations are performed is not important because for any order the effect is the same.

$$\text{i.e. } (A \cdot B) \cdot C = A \cdot (B \cdot C)$$
$$\text{and } (A + B) + C = A + (B + C)$$

- The associative law can be verified by referring to the following truth table.

Table 4.4.4 : Verification of the associative law

Inputs			$B \cdot C$	$A \cdot (B \cdot C)$	$A \cdot B$	$(A \cdot B) \cdot C$
A	B	C				
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

Similarly we can verify the other statement i.e.

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

4.5 Basic Theorems and Properties of Boolean Algebra : MU : May 05

University Questions

Q. 1 State the Boolean algebra laws used in K-map simplification. (May 05, 4 Marks)

4.5.1 Duality :

- According to the duality theorem the following conversions are possible in a given Boolean expression :
  1. Change each AND operation to an OR operation.
  2. Change each OR operation to an AND operation.
  3. Complement any 1 or 0 appearing in the expression.
- Duality theorem is sometimes useful in creating new expressions from the given Boolean expressions.

- For example if the given expression is $A + 1 = 1$ then replace the OR (+) operation by AND (·) operation and take complement of 1 to write the dual of the given relation as,

$$A \cdot 0 = 0$$

- The dual of $A(B + C) = AB + AC$ is given by,

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

- It is possible to verify this theorem by constructing truth tables for both the sides of equations.

4.5.2 Basic Theorems :

1. **AND Laws :**

These laws are related to the AND operation therefore they are called as "AND" laws. The AND laws are as follows :

1. $A \cdot 0 = 0$
   - That means if one input of an AND operation is permanently kept at the LOW (0) level, then the output is zero irrespective of the other variable.
2. $A \cdot 1 = A$
   - That means if one input of an AND operation is HIGH (1) permanently then the output is always equal to the other variable.
   - So if $A = 0$, then $Y = 0 \cdot 1 = 0$ i.e. $Y = A$ and if $A = 1$, then $Y = 1 \cdot 1 = 1$ so $Y = A$.
3. $A \cdot A = A$
   - That means if both the inputs in an AND operation have the same value either "0" or "1" then the output will also have the same value as that of the input.
4. $A \cdot \bar{A} = 0$
   - This law states that the result of an "AND" operation on a variable (A) and its complement $(\bar{A})$ is always LOW (0).
   - If $A = 0$ then $\bar{A} = 1$ and $Y = 0 \cdot 1 = 0$ whereas if $A = 1$ then $\bar{A} = 0$ and $Y = 1 \cdot 0 = 0$.

Summary of AND Laws :

1. $A \cdot 0 = 0$ 2. $A \cdot 1 = A$ 3. $A \cdot A = A$ 4. $A \cdot \bar{A} = 0$

2. **OR Laws :**

These laws use the OR operation. Therefore they are called as OR laws. The OR laws are as follows :

1. $A + 0 = A$
   - That means if one variable of an "OR" operation is LOW (0) permanently, then the output is always equal to the other variable.
   - $B = 0$ permanently therefore for $A = 0$, $Y = 0 + 0 = 0$ i.e. $Y = A$ and for $A = 1$, $Y = 1 + 0 = 1$ i.e. $Y = A$.
2. $A + 1 = 1$
   - That means if one variable of an "OR" operation is HIGH (1) permanently, then the output is HIGH (1) permanently irrespective of the value of the other variable.
   - Here $B = 1$ permanently, so if $A = 0$ then $Y = 0 + 1 = 1$ and if $A = 1$ then $Y = 1 + 1 = 1$.
   - Thus output remains HIGH (1) always, irrespective of the value of A.

Scanned by CamScanner

3. $A + A = A$

- This law states that if both the variables of an OR operation have the same value either "0" or "1" then the output also will be equal to the input i.e. 0 or 1 respectively.
- For $A = 0$, $Y = 0 + 0 = 0$ i.e. $Y = A$ and for $A = 1$, $Y = 1 + 1 = 1$ i.e. $Y = A$.

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

- This law states that the result of an "OR" operation on a variable and its complement is always 1 (HIGH).
- If $A = 0$ then $\bar{A} = 1$ and $Y = 0 + 1 = 1$ whereas if $A = 1$ then $\bar{A} = 0$ and $Y = 1 + 0 = 1$.

Summary of OR Laws :

1. $A + 0 = A$ 2. $A + 1 = 1$ 3. $A + A = A$ 4. $A + \bar{A} = 1$

3. INVERSION Law :

- This law uses the "NOT" operation. The inversion law states that if a variable is subjected to a double inversion then it will result in the original variable itself i.e.

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

- Inversion law is being illustrated in Fig. 4.5.1 which shows that if $A = 0$ then $\bar{A} = 1$ and $(\bar{\bar{A}}) = 0 \therefore Y = A$, whereas if $A = 1$ then $\bar{A} = 0$ and $(\bar{\bar{A}}) = 1 \therefore Y = A$.

(B-481) **Fig. 4.5.1 : Illustration of inversion law : $\bar{\bar{A}} = A$**

Table 4.5.1 : Collection of Boolean laws

Sr. No.	Name	Statement of the law
1.	Commutative Law	$A \cdot B = B \cdot A$ $A + B = B + A$
2.	Associative Law	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$ $(A + B) + C = A + (B + C)$
3.	Distributive Law	$A \cdot (B + C) = AB + AC$
4.	AND Laws	$A \cdot 0 = 0$ $A \cdot 1 = A$ $A \cdot A = A$ $A \cdot \bar{A} = 0$
5.	OR Laws	$A + 0 = A$ $A + 1 = 1$ $A + A = A$ $A + \bar{A} = 1$
6.	Inversion Law	$\bar{\bar{A}} = A$

Sr. No.	Name	Statement of the law
7.	Other Important Laws	$A + BC = (A + B)(A + C)$ $\bar{A} + AB = \bar{A} + B$ $\bar{A} + A\bar{B} = \bar{A} + \bar{B}$ $A + AB = A$ $A + \bar{A}B = A + B$

4.5.3 De-Morgan's Theorems : MU : May 03, Dec. 03, Dec. 05, May 06, May 07, Dec. 08, May 09, Dec. 09, May 10, Dec. 11, May 12, Dec. 12, Dec. 13, May 15, May 16

University Questions

Q. 1 State and prove De-Morgan's theorems. **(May 03, Dec. 03, Dec. 05, May 06, May 07, Dec. 08, May 09, Dec. 09, May 10, Dec. 11, May 12, Dec. 12, May 16, 4 Marks)**

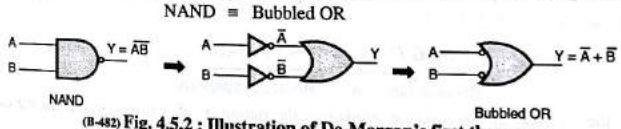
Q. 2 State De-Morgan's theorems. **(Dec. 13, 5 Marks)**

Q. 3 State De-Morgan's law. **(May 15, 2 Marks)**

The two theorems suggested by De-Morgan and which are extremely useful in Boolean algebra are as follows :

Theorem 1 : $\overline{AB} = \bar{A} + \bar{B}$: NAND = Bubbled OR :

- This theorem states that the, complement of a product is equal to addition of the complements.
- This rule is illustrated in Fig. 4.5.2. The Left Hand Side (LHS) of this theorem represents a NAND gate with inputs A and B whereas the Right Hand Side (RHS) of the theorem represents an OR gate with inverted inputs. Such an OR gate is called as "Bubbled OR". Thus we can state De-Morgan's first theorem as a NAND operation is equivalent to a bubbled OR operation.



(B-482) **Fig. 4.5.2 : Illustration of De-Morgan's first theorem**

This theorem can be verified by writing a truth table as shown in Fig. 4.5.3.

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

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

Fig. 4.5.3 : Verification of the theorem $\overline{AB} = \bar{A} + \bar{B}$

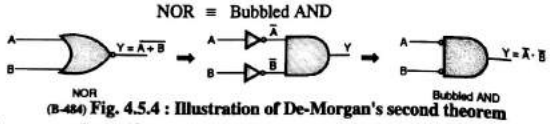
Theorem 2 : $\overline{A + B} = \bar{A} \cdot \bar{B}$: NOR = Bubbled AND :

- This theorem is illustrated in Fig. 4.5.4. The LHS of this theorem represents a NOR gate with inputs A and B whereas the RHS represents an AND gate with inverted inputs.

Scanned by CamScanner

- Such an AND gate is called as "Bubbled AND". Thus we can state De-Morgan's second theorem as a NOR function is equivalent to a bubbled AND function.

NOR ≡ Bubbled AND



(B-484) **Fig. 4.5.4 : Illustration of De-Morgan's second theorem**

- This theorem can be verified by writing a truth table for both the sides of the theorem statement. This truth table is shown in Fig. 4.5.5, which shows that LHS = RHS.

A	B	$\overline{A+B}$	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	**1**	1	1	**1**
0	1	**0**	1	0	**0**
1	0	**0**	0	1	**0**
1	1	**0**	0	0	**0**

LHS $\overline{A+B} = \bar{A} \cdot \bar{B}$ RHS

Fig. 4.5.5 : Truth table to verify De-Morgan's theorem

4.6 Boolean Expression and Boolean Function :

- Boolean algebra deals with binary variables and logic operations. A Boolean function is described by an algebraic expression called Boolean expression which consists of binary variables, the constants 0 and 1 and the logic operation symbols.
- Consider the following example :

$$\underbrace{F(A, B, C, D)}_{\text{Boolean function}} = \underbrace{A + B\bar{C} + ADC}_{\text{Boolean expression}} \quad \ldots(4.6.1)$$

- Here the left side of the equation represents the output Y of a logic circuit. So we can state Equation (4.6.1) as,

$$Y = A + B\bar{C} + ADC \quad \ldots(4.6.2)$$

- Another example of Boolean function is as follows :

$$F(A, B, C) = A + BC \quad \ldots(4.6.3)$$

$$\text{or simply} \quad Y = A + BC \quad \ldots(4.6.4)$$

4.6.1 Truth Table Formation :

- It is possible to convert the switching equation into a truth table. For example consider the following switching equation.

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

- It shows that there are three inputs A, B, and C and one output f (A, B, C) or simply F.
- The output will be high (1) if A = 1 or BC = 1 or both are 1 due to the (+) i.e. OR function present between A and BC.

- The truth table for this equation is shown by Table 4.6.1. The number of rows in the truth table is 2^n where n is the number of input variables (n = 3 for the given equation). Hence there are $2^3 = 8$ possible input combinations of inputs from ABC = 000 to ABC = 111.

Table 4.6.1 : Truth table for f (A, B, C) = A + BC

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Eight possible combinations of inputs

Value of output F corresponding to each combination of inputs

- If there are two inputs (n = 2), then there are $2^2 = 4$ combinations of inputs. For four inputs (n = 4), there are $2^4 = 16$ combinations.

4.6.2 Examples on Reducing the Boolean Expression :

Ex. 4.6.1 : Prove the following Boolean expressions :

1. $A + AB = A$ 2. $A + \bar{A}B = A + B$ 3. $(A + B)(A + C) = A + BC$

Soln. :

1. **To prove that A + AB = A :**

$$\text{LHS} = A + AB = A(1 + B)$$

But $(1 + B) = 1$ $\quad \therefore$ LHS = A · 1 $\quad$ But A · 1 = A

$\therefore$ LHS = A = RHS

$\therefore A + AB = A$

...Proved.

2. **To prove that $A + \bar{A}B = A + B$:**

$$\text{LHS} = A + \bar{A}B = A + AB + \bar{A}B \quad \ldots \text{since } A = A + AB$$

$$\therefore \text{LHS} = A + B(A + \bar{A})$$

But $(A + \bar{A}) = 1$

$$\therefore \text{LHS} = A + (B \cdot 1) = A + B$$

$$\therefore \text{LHS} = \text{RHS}$$

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

... Proved.

3. **To prove that (A + B) (A + C) = A + BC :**

$$\text{LHS} = (A + B)(A + C)$$

$$\therefore \text{LHS} = AA + AC + BA + BC \quad \ldots\ldots \text{According to the distributive law.}$$

But $AA = A$, $\therefore$ LHS = A + AC + BA + BC

$$\therefore \text{LHS} = A(1 + C) + AB + BC$$

Scanned by CamScanner

But $(1+C) = 1$

$\therefore$ LHS $= A + AB + BC = A(1+B) + BC$

But $(1+B) = 1$

$\therefore$ LHS $= A + BC$

Thus $(A+B)(A+C) = A + BC$ ... Proved

Ex. 4.6.2 : Prove that $(A + \bar{B} + AB)(A + \bar{B})(\bar{A}B) = 0$

Soln. :

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

But $A + AB = A$ ... Refer to Ex. 4.6.1

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

But $A \cdot A = A$ and $\bar{B} \cdot \bar{B} = \bar{B}$ and $A\bar{B} + A\bar{B} = \bar{B}(A + A) = A\bar{B}$

$\therefore$ LHS $= (A + A\bar{B} + \bar{B})(\bar{A}B) = [A(1 + \bar{B}) + \bar{B}](\bar{A}B)$

But $(1 + \bar{B}) = 1$ $\therefore$ LHS $= [(A \cdot 1) + \bar{B}](\bar{A}B)$

$\therefore$ LHS $= (A + \bar{B})(\bar{A}B)$ ... since $A \cdot 1 = A$

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

But $A\bar{A} = 0$ and $B\bar{B} = 0$

$\therefore$ LHS $= 0 + 0 = 0$ ... Proved

Ex. 4.6.3 : Simplify the following expression : $Y = \overline{\overline{(\overline{AB} + \bar{A} + AB)}}$

Soln. :

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

But $\overline{AB} = \bar{A} + \bar{B}$ ...De-Morgan's first theorem

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

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

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

Now use De-Morgan's second theorem which states that,

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

$\therefore Y = \bar{\bar{A}} \cdot \bar{\bar{B}} \cdot \overline{\overline{AB}}$

But $\bar{\bar{A}} = A$ and $\bar{\bar{B}} = B$

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

But $\overline{\overline{AB}} = (\bar{A} + \bar{B})$ .... De-Morgan's second theorem.

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

But $A\bar{A} = 0$ and $B\bar{B} = 0$

$\therefore Y = 0 \cdot B + A \cdot 0$

$= 0 + 0$ ...since $0 \cdot B = 0$ and $A \cdot 0 = 0$

$\therefore Y = 0$

Ex. 4.6.4 : Simplify the following Boolean expressions :

1. $A\bar{B} + \bar{A}B + AB + \bar{A}\bar{B}$ 2. $A\bar{B}C + \bar{A}BC + ABC$

Soln. :

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

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

$\therefore Y = 1$ ....since $\bar{B} + B = 1$

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

$= AC + \bar{A}BC$ ...since $\bar{B} + B = 1$

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

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

Ex. 4.6.5 : Prove the following Boolean identities :

1. $A + \bar{A}B + AB = A + B$ 2. $A\bar{B} + \bar{A}B + AB + \bar{A}\bar{B} = 1$.

Soln. :

1. LHS $= A + \bar{A}B + AB = A + B(\bar{A} + A)$

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

$\therefore$ LHS $= A + B =$ RHS.

2. LHS $= A\bar{B} + \bar{A}B + AB + \bar{A}\bar{B} = A\bar{B} + \bar{A}\bar{B} + \bar{A}B + AB$

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

$= 1$ ...since $\bar{B} + B = 1$

$\therefore$ LHS $= 1$ **...Proved.**

Ex. 4.6.6 : Simplify $y = AB(\overline{CD}) + \bar{B}CD + (\bar{A} + \bar{C})(B + D)$

Soln. : May 04. 5 Marks. Dec. 14. 5 Marks

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

$= AB(\bar{C} + \bar{D}) + \bar{B}CD + \bar{A}B + \bar{A}D + B\bar{C} + \bar{C}D$ [$\because \overline{AB} = \bar{A} + \bar{B}$]

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

$= B\bar{C}(A + 1) + AB\bar{D} + D(\bar{B}C + \bar{C}) + \bar{A}B + \bar{A}D$

$= B\bar{C} + AB\bar{D} + D(\bar{C} + \bar{B}) + \bar{A}B + \bar{A}D$ [$\because 1 + A = 1$ and $A + \bar{A}B = A + B$]

Scanned by CamScanner

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

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

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

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

$= B\,\overline{ACD} + D\,\overline{ABC}$ [$\because \bar{A} + \bar{B} = \overline{A \cdot B}$]

Ex. 4.6.7 : A misguided mathematician would like to subtract term $A\bar{C}$ form both sides of equality.

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

Would they still be equal if he did so. Justify.

Simplify the expression : $F = (X + \bar{Z})(\overline{Z + WY}) + (VZ + W\bar{X})(\overline{Y + Z})$

Dec. 04. 10 Marks, Dec. 14. 10 Marks

Soln. :

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

Subtracting $A\bar{C}$ from both sides

$BC + ABD + A\bar{C} - A\bar{C} = BC + A\bar{C} - A\bar{C}$

$BC + ABD = BC$

$LHS \neq RHS$

Given :

$F = (X + \bar{Z})(\overline{Z + WY}) + (VZ + W\bar{X})(\overline{Y + Z})$

$= (X + \bar{Z})(\bar{Z} \cdot \overline{WY}) + (VZ + W\bar{X})(\bar{Y}\ \bar{Z})$ [$\because \overline{A + B} = \bar{A} \cdot \bar{B}$]

$= (X + \bar{Z})[\bar{Z} \cdot (\bar{W} + \bar{Y})] + VZ\bar{Y}\bar{Z} + W\bar{X}\bar{Y}\bar{Z}$

$= (X + \bar{Z})[\bar{W}\bar{Z} + \bar{Y}\bar{Z}] + 0 + W\bar{X}\bar{Y}\bar{Z} = \bar{Z}[X\bar{W} + \bar{W} + X\bar{Y} + \bar{Y} + W\bar{X}\bar{Y}]$

$= \bar{Z}[\bar{W}(X + 1) + \bar{Y}(X + 1) + W\bar{X}\bar{Y}]$

$= \bar{Z}[\bar{W} + \bar{Y} + W\bar{X}\bar{Y}]$ [$\because 1 + A = 1$]

$= \bar{Z}[\bar{W} + \bar{Y}(1 + W\bar{X})] = \bar{Z}[\bar{W} + \bar{Y}]$

Ex. 4.6.8 : Prove the following :

1. $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC = AB + BC + AC$
2. $\overline{A \oplus B} = A \oplus \bar{B} = \bar{A} \oplus B$

May 05. Dec. 13. 5 Marks

Soln. :

1. $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC = AB + BC + AC$:

L.H.S. = $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

Rearranging the terms,

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

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

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

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

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

$= AB + AC + BC$ **...Hence proved**

2. $\overline{A \oplus B} = A \oplus \bar{B} = \bar{A} \oplus B$:

L. H. S. $= \overline{A \oplus B} = \overline{\bar{A}B + A\bar{B}}$

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

$= (A + \bar{B}) + (\bar{A} + B)$ [$\because \overline{AB} = \bar{A} + \bar{B}$]

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

Ex. 4.6.9 : Simplify using Boolean laws : $\overline{\overline{AB} + \bar{A} + AB}$.

Dec. 08. 4 Marks

Soln. : $\overline{\overline{AB} + \bar{A} + AB}$ ...Given

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

$= AB \cdot A \cdot (\bar{A} + \bar{B})$ [$\because \bar{\bar{A}} = A$]

$= AB \cdot (\bar{A} + \bar{B})$ [$\because A \cdot A = A$]

$= A\bar{A}B + AB\bar{B}$ [$\because A\bar{A} = 0$]

$= 0 + 0 = 0$

Ex. 4.6.10 : Simplify and implement using gates :

$Y = \overline{AB}(B + C) + AB\,\overline{(B + C)}$

Dec. 09. 4 Marks

Soln. :

$y = \overline{AB}(B + C) + AB\,(\overline{B + C})$

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

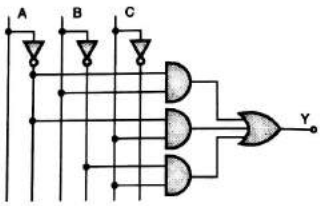
[$\because \overline{AB} = \bar{A} + \bar{B}$ and $\overline{B + C} = \bar{B} \cdot \bar{C}$ De-Morgan's theorem]

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

$= \bar{A}B + \bar{A}C + \bar{B}C$ [$\because B\bar{B} = 0$]

Scanned by CamScanner

Implementation using gates :



(C-2178) **Fig. P. 4.6.10**

Ex. 4.6.11 : Simplify using Boolean theorems and draw logic diagram for the following :

1. $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$ 2. $A[B + C(\overline{AC + AB})]$

3. $\overline{AB}(B + C) + AB(\overline{B + C})$

Dec. 10. 12 Marks

Soln. :

1. $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$:

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

$$= \bar{A}BC + A\bar{B}C + AB \qquad [\because C + \bar{C} = 1]$$

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

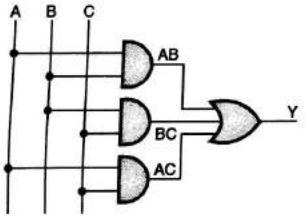
$$= \bar{A}BC + A(B + C) \qquad [\because A + \bar{A}B = A + B]$$

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

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

$$= B(A + C) + AC \qquad [\because A + \bar{A}B = A + B]$$

$$= AB + BC + AC \qquad \text{... Final expression}$$



(C-2179) **Fig. P. 4.6.11(a)**

Implementation using logic gates :

The implementation of final expression is as shown in Fig. P. 4.6.11(a).

2. $A[B + C(\overline{AC + AB})]$:

$$= A[B + C(\overline{AC} \cdot \overline{AB})] \qquad [\because \overline{A + B} = \bar{A} \cdot \bar{B}]$$

$$= A[B + C((\bar{A} + \bar{C}) \cdot (\bar{A} + \bar{B}))] \qquad [\because \overline{AB} = \bar{A} + \bar{B}]$$

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

$$= A[B + C(\bar{A} + \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{B}\bar{C})] \qquad [\because \bar{A} \cdot \bar{A} = \bar{A}]$$

$$= A[B + C(\bar{A}(1 + \bar{B} + \bar{C}) + \bar{B}\bar{C})]$$

$$= A[B + C(\bar{A} + \bar{B}\bar{C})] \qquad [\because 1 + \bar{B} + \bar{C} = 1]$$

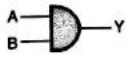
$$= A[B + \bar{A}C + \bar{B}C\bar{C}]$$

$$= A[B + \bar{A}C] \qquad [\because C\bar{C} = 0]$$

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

$$= AB \qquad [\because A\bar{A} = 0]$$

Implementation using gates :



(C-2180) **Fig. P. 4.6.11(b)**

3. $\overline{AB}(B + C) + AB(\overline{B + C})$:

$$= \overline{AB}(B + C) + AB(\bar{B} \cdot \bar{C}) \qquad [\because \overline{A + B} = \bar{A} \cdot \bar{B}]$$

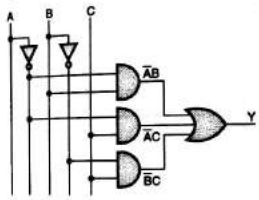
$$= \overline{AB}(B + C) + AB\bar{B}\bar{C}$$

$$= \overline{AB}(B + C) \qquad [\because B\bar{B} = 0]$$

$$= (\bar{A} + \bar{B})(B + C) \qquad [\because \overline{AB} = \bar{A} + \bar{B}]$$

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

$$= \bar{A}B + \bar{A}C + \bar{B}C \qquad [\because B\bar{B} = 0]$$



(C-2181) **Fig. P. 4.6.11(c)**

Implementation using gates : The implementation is as shown in Fig. P. 4.6.11(c).

Ex. 4.6.12 : Simplify $Y = \overline{(A + \bar{A}B)(C + \bar{D})}$.

May 15. 5 Marks

Soln. :

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

$$= \overline{(A + \bar{A}B)} + \overline{(C + \bar{D})} \qquad \text{...(Using De-morgan's theorem)}$$

$$= \left[\bar{A} \cdot \overline{\bar{A}B}\right] + \left[\bar{C} \cdot \bar{\bar{D}}\right] \qquad \text{...(Using De-morgan's theorem)}$$

$$= \bar{A} \cdot [\bar{\bar{A}} \cdot \bar{B}] + [\bar{C} \cdot D] \qquad \text{...(Using De-morgan's theorem)}$$

$$= \bar{A} \cdot [A\bar{B}] + \bar{C}D \qquad \text{... } (\because \bar{\bar{A}} = A)$$

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

$$Y = \bar{C}D \qquad \text{... } (\because A\bar{A} = 0)$$

Scanned by CamScanner

4.6.3 Complement of a Function :

- The complement of a function F is denoted by $\overline{F}$. We can obtain $\overline{F}$ by replacing 0's by 1's and 1's by 0's while calculating the value of F.
- It is possible to derive the complement of a function algebraically using De Morgan's theorems. We can extend De Morgan's theorems to three or more variables as well.
- The three variable form of De Morgan's first theorem is derived as follows :

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

Let $B+C = D$ $\quad\therefore \text{LHS} = \overline{(A+D)} = \overline{A}\cdot\overline{D}$ .... As per De Morgan's theorem

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

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

- On the same lines we can derive the De Morgan's theorems for any number of variables. These theorems in the generalized form can be stated as follows :

1. $\overline{(A+B+C+D \ldots\ldots\ldots\ldots +G)} = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\ldots\ldots\cdot\overline{G}$
2. $\overline{(A\cdot B\cdot C\cdot D\cdot\ldots\ldots\ldots\ldots G)} = \overline{A}+\overline{B}+\overline{C}+\overline{D}+\ldots\ldots+\overline{G}$

Ex. 4.6.13 : Find the complements of the following functions :

$$F_1 = \overline{A}B\overline{C} + \overline{A}\overline{B}C \quad\text{and}\quad F_2 = A(\overline{B}\overline{C}+BC)$$

Soln. :

1. $F_1 = \overline{A}B\overline{C} + \overline{A}\overline{B}C$:

$$\therefore \overline{F_1} = \overline{(\overline{A}B\overline{C}+\overline{A}\overline{B}C)} = \overline{(\overline{A}B\overline{C})}\cdot\overline{(\overline{A}\overline{B}C)}$$

$$\therefore \overline{F_1} = (A+\overline{B}+C)\cdot(A+B+\overline{C})$$ ...Ans.

2. $F_2 = A(\overline{B}\overline{C}+BC)$:

$$\therefore \overline{F_2} = \overline{[A(\overline{B}\overline{C}+BC)]} = [\overline{A}+\overline{(\overline{B}\overline{C}+BC)}] = \overline{A}+\overline{(\overline{B}\overline{C})}\cdot\overline{(BC)}$$

$$\therefore \overline{F_2} = \overline{A}+(B+C)(\overline{B}+\overline{C})$$ ...Ans.

4.7 Logic Gates :

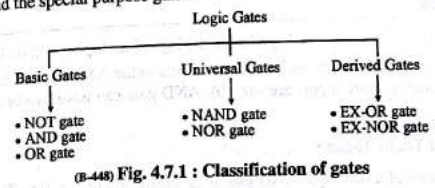
- Logic gates are the logic circuits which act as the basic building blocks of any digital system. It is an electronic circuit having one or more than one inputs and only one output.
- The relationship between the input and the output is based on a "certain logic". Depending on this logic, the gates are named as NOT gate, AND gate, OR, NAND, NOR etc.
- We have to use different laws, rules and theorems to analyze the digital circuits.

- By connecting the gates mentioned earlier, in different ways, we can build circuits that can perform arithmetic and other functions associated with the human brain.

Because they simulate mental processes, gates are often called as logic circuits.

4.7.1 Classification of Logic Gates :

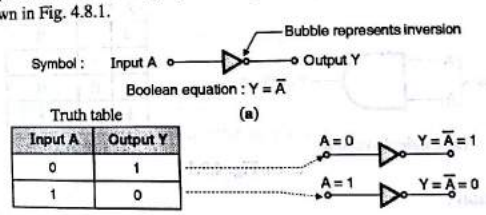
As shown in Fig. 4.7.1, the logic gates are classified into three categories namely, the basic gates, the universal gates and the special purpose gates.



(B-448) **Fig. 4.7.1 : Classification of gates**

4.8 NOT Gate or Inverter :

The NOT gate or Inverter is a logic gate having one input (A) and one output (Y). Its symbol and truth table are shown in Fig. 4.8.1.



(b) (c) Equivalent circuits

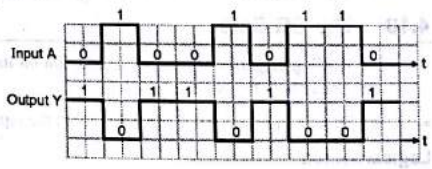
(B-450) **Fig. 4.8.1 : Symbol, truth table and equivalent circuit of a NOT gate**

Important points about the NOT gate :

1. The NOT gate is also known as an "Inverter" because its output is the inverted version or "complement" of its input. This is shown in the truth table of NOT gate.
2. The bubble (o) in the symbol of a NOT gate indicates the inversion operation.
3. The operation of a NOT gate for various input combinations is as shown in Fig. 4.8.1(c).

4.8.1 Operation for Pulsed Input :

- The input to a gate may not always be in the form of a dc voltage. In certain applications the inputs can be in the form of a train of rectangular pulses with their high and low levels within the prescribed limits as shown in Fig. 4.8.2.



(B-451) **Fig. 4.8.2 : Timing diagram for the pulsed operation of a NOT gate**

Scanned by CamScanner

- Fig. 4.8.2 shows the timing diagrams for the pulsed operation of a NOT gate.
- In the pulsed operation, pulses of finite duration are applied at the gate input. These waveforms show that for a "0" input, the gate produces a "1" output. Thus the inversion is observed.

4.8.2 Standard Package (IC Gate) :

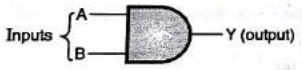
- The NOT gate is available in the monolithic Integrated Circuit (IC) form.
- IC 7404 is a 14 pin TTL integrated circuit with six inverters inside.

4.9 AND Gate :

- AND is one of the logic operators. It performs the logical multiplication on its inputs.
- The output is high (Y = 1) if and only if all the inputs to the AND gate are high (1). The output is low (0), if any one or more inputs are low (0). AND gate can have two or more inputs and only one output.

Logical symbol and Truth Table :

The logical symbol of a two input AND gate is as shown in Fig. 4.9.1(a). The truth table of a two input AND gate is shown in Fig. 4.9.1(b). Note that the output is HIGH (1) only when both the inputs are HIGH (1).



(a) Logical symbol

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

(B-453) Fig. 4.9.1

Boolean expression :

The expression relating the inputs (A, B) and output (Y) of a gate is called as the Boolean expression. The Boolean expression for an AND gate is,

$$Y = A \cdot B$$

where the "dot" between A and B represents multiplication.

4.9.1 Standard Package (IC Gate) :

- The AND gate is available in the monolithic Integrated Circuit (IC) form. IC 7408 is a 14 pin TTL integrated circuit with four AND gates inside. It is therefore called as Quad AND gate IC.

4.10 The OR Gate :

- An "OR" gate performs the logical addition on its inputs therefore its output will be high (1) if any one or both the inputs are high (1).
- Its output will be low (0) if and only if both the inputs are simultaneously low (0).

Logical symbol :

- Fig. 4.10.1(a) shows the logical symbol of a two input OR gate.

Truth table :

- Fig. 4.10.1(b) shows the truth table for a two input OR gate. Note that the output Y is LOW (0) if and only if both the inputs are LOW (0).
- If any one or both the inputs are HIGH (1), then the output Y is HIGH (1).

(a) Logical symbol

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

(B-460) Fig. 4.10.1 : Two input OR gate

Boolean equation :

The Boolean expression for a two input OR gate is,

$$Y = A + B$$

4.10.1 Standard Package (IC Gate) :

The OR gate is available in the monolithic integrated circuit form. IC 7432 is a 14 pin TTL integrated circuit with four, two input OR gates inside.

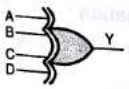
4.11 Special Type of Gates or Derived Gates :

- EX-OR and EX-NOR gates are special type of gates. They can be used for applications such as half adder, full adder and subtractors.
- These gates are also called as the derived gates.

4.11.1 The EX-OR Gate :

- The exclusive-OR gate is abbreviated as EX-OR gate or sometimes as X-OR gate.
- An EX-OR gate can have two or more than two input terminals and one output terminal as shown in Figs. 4.11.1(a) and (b).

(a) Symbol of a two input EX-OR gate



(b) Symbol of a four input EX-OR gate

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(c) Truth table of a two input EX-OR gate

(B-475) Fig. 4.11.1

Scanned by CamScanner

- The truth table of a two input EX-OR gate is shown in Fig. 4.11.1(c), which shows that, when both the inputs are at identical logic levels (A = B), the output is low (0) i.e. Y = 0 for A = B = 0 or A = B = 1, and the output is high (1) when A ≠ B.
- The Boolean expression for the two input EX-OR gate is, $Y = A \oplus B$.

Applications of EX-OR :

Some of the applications of an EX-OR gate are as follows :

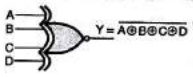
1. As a magnitude comparator.
2. In the binary to gray code converter.
3. In the adder and subtractor circuits.
4. In the parity generator.
5. As a modulo-2 adder.

4.11.2 The EX-NOR Gate :

- The word EX-NOR is a short form of exclusive-NOR. Exclusive-NOR means NOT-exclusive OR, so EX-NOR gate is equivalent to an EX-OR gate followed by a NOT gate.
- The symbol for a two input EX-NOR gate is as shown in Fig. 4.11.2(a) and its truth table is given in Fig. 4.11.2(c), which shows that the output of an EX-NOR gate is high (1) if both the inputs are identical (A = B) and the output is low (0) if the inputs are not identical (A ≠ B).
- The Boolean expression for an EX-NOR gate is given by,

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

(a) Symbol of a two input EX-NOR gate



(b) A four input EX-NOR gate

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

(c) Truth table of a two input EX-NOR gate

(B-478) **Fig. 4.11.2**

Applications of EX-NOR :

1. As even parity generator. 2. As a comparator. 3. As even parity checker.

4.11.3 Realization of Switching Functions :

- The Boolean expression is also called as a switching function. A switching function can be realized or described in three different ways :
  1. Switching equations 2. Truth tables 3. Logic diagrams

4.11.4 To Draw a Logic Circuit from Boolean Equation :

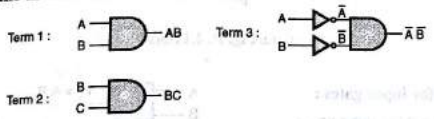
- First we develop the switching equations necessary for realization of a combinational logic function.
- Then each term in this equation is represented by a logic gate.
- All these logic gates are then connected to each other to form the logic diagram.
- This is called realization of switching functions using logic diagram. Following examples will illustrate the concept.

Ex. 4.11.1 : Draw the combinational circuit using the basic gates to obtain the following output.

$$Y = AB + BC + \overline{AB}$$

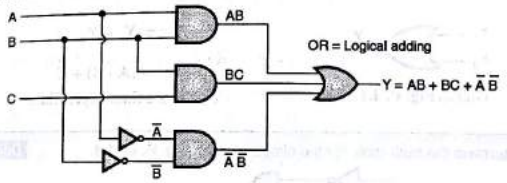
Soln. : This circuit has three inputs A, B, C and one output.

Step 1 : Implement the three individual terms :



(C-158) **Fig. P. 4.11.1(a)**

Step 2 : Logically add the outputs of the three circuits :

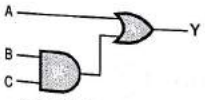


(C-159) **Fig. P. 4.11.1(b)**

Ex. 4.11.2 : Sketch the following equation. Use one AND gate and one OR gate only.

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

Soln. :



(C-160) **Fig. P. 4.11.2**

$$Y = (A + B)(A + C) = AA + AB + AC + BC$$
$$= A + AB + AC + BC \quad \ldots (\because A \cdot A = A)$$
$$= A(1 + B + C) + BC$$
$$\therefore Y = A + BC \quad \ldots (\because 1 + B + C = 1)$$

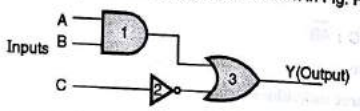
This expression can be expressed using logic gates as shown in Fig. P. 4.11.2.

4.11.5 To Write a Boolean Expression for a Logic Circuit :

- Sometimes a logic diagram is given to us and we have to write the switching equation or Boolean equation relating the inputs and outputs.
- The procedure for this is as follows :
  1. Start from the input side.
  2. Write the equation at the output of each input gate.
  3. Repeat this process and progress towards the output.
  4. Write the switching equation for output.
- This procedure is demonstrated in the following example.

Scanned by CamScanner

Ex. 4.11.3 : Obtain the Boolean equation for the logic circuit shown in Fig. P. 4.11.3(a).

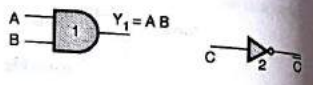


(C-162) **Fig. P. 4.11.3(a)**

Soln. :

Step 1 : Equations for input gates :

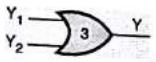
Gates - 1 and 2 are the input gates.



(C-163) **Fig. P. 4.11.3(b)**

Step 2 : Equations for output gate :

Gate - 3 is the output gate.



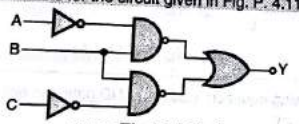
(C-163(a)) **Fig. P. 4.11.3(c)**

$$Y = Y_1 + Y_2$$

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

This is the final expression.

Ex. 4.11.4 : Determine the truth table for the circuit given in Fig. P. 4.11.4. **Dec. 11, 4 Marks**



(C-2197) **Fig. P. 4.11.4**

Soln. :

Inputs			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Ex. 4.11.5 : Simply using Boolean theorems and implement using AOI gates only.

1. $(C + \bar{C}D)(C + \bar{C}\bar{D})[(AB + \bar{A}\bar{B}) + (A \oplus B)]$
2. $\overline{\overline{AB + \bar{A}\bar{B}} + \overline{(A+B) \cdot \bar{A} + B}}$

Dec. 07, 10 Marks, Dec. 14, 5 Marks

Soln. :

1. $[(C + \bar{C}D)(C + \bar{C}\bar{D})]\ [(AB + \bar{A}\bar{B}) + (A \oplus B)]$:

$$= [CC + C\bar{C}\bar{D} + C\bar{C}D + \bar{C}\bar{C}D\bar{D}]\ [(AB + \bar{A}\bar{B})(\bar{A}B + A\bar{B})]$$

$$= [C + 0 + 0 + 0]\ [A\bar{A}BB + AAB\bar{B} + \bar{A}\bar{B}\bar{A}B + \bar{A}\bar{B}A\bar{B}]$$

$$= C[0 + 0 + 0 + 0] \qquad [\because A\bar{A} = 0]$$

$$= 0 \qquad [\because C \cdot 0 = 0]$$

2. $\overline{\overline{AB + \bar{A}\bar{B}} + \overline{(A+B) \cdot (\bar{A} + B)}}$:

$$= \overline{\overline{AB} \cdot \overline{\bar{A}\bar{B}} + \overline{A+B} + \overline{\bar{A}+B}} \qquad [\because \overline{A+B} = \bar{A} \cdot \bar{B} \text{ and } \overline{A \cdot B} = \bar{A} + \bar{B}]$$

$$= \overline{\bar{A} + \bar{B} \cdot \bar{\bar{A}} + \bar{\bar{B}} + \bar{A} \cdot \bar{B} + \overline{\bar{A}} \cdot \bar{B}} \qquad [\because \overline{A \cdot B} = \bar{A} + \bar{B} \text{ and } \overline{A+B} = \bar{A} \cdot \bar{B}]$$

$$= \overline{(\bar{A} + \bar{B}) \cdot (A+B) + \bar{A} \cdot \bar{B} + A \cdot \bar{B}} \qquad [\because \bar{\bar{A}} = A]$$

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

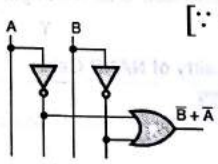
$$= 0 + \bar{A}B + A\bar{B} + 0 + \bar{A}\bar{B} \qquad [\because A\bar{A} = 0, A + A = A]$$

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

$$= \bar{A}B + \bar{B} \qquad [\because A + \bar{A} = 1]$$

$$= \bar{B} + \bar{A} \qquad [\because \bar{A} + A\bar{B} = \bar{A} + \bar{B}]$$

or



(C-3711)

4.12 Universal Gates :

MU : Dec. 02, May 03, Dec. 03, Dec. 06

University Questions

Q. 1 Why are NAND and NOR gates called universal logic gates ? **(Dec. 02, 3 Marks)**

Q. 2 Write short note on : Universal logic modules. **(May 03, 4 Marks)**

Q. 3 What are universal gates and why are they so called ?(give examples). **(Dec. 03, 4 Marks)**

Q. 4 Write short note on : Universal gates. **(Dec. 06, 5 Marks)**

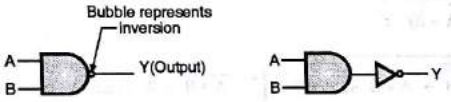
- The NAND and NOR gates are called as "Universal Gates" because it is possible to implement any Boolean expression with the help of only NAND or only NOR gates.
- Therefore a user can build any logical circuit with the help of only NAND gates or only NOR gates.

Scanned by CamScanner

- This is a great advantage because a user will have to make a stock of only NAND or NOR gate ICs with him.

4.12.1 The NAND Gate :

- The term NAND can be split as NOT-AND which means that the NAND operation can be implemented with the combination of an AND gate and a NOT gate i.e. inverter.
- Thus a NAND gate is equivalent to an AND gate followed by an inverter as shown in Fig. 4.12.1(b).



Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

(a) Logical symbol (B-467) (b) Equivalent circuit (c) Truth table

Fig. 4.12.1 : Two input NAND gate

- The symbol of a two input NAND gate is shown in Fig. 4.12.1(a) where a bubble (o) on the output side represents inversion.
- The truth table of a two input NAND gate is shown in Fig. 4.12.1(c), which shows that the output is low (0) if and only if both the inputs are high (1) simultaneously. For all other input combinations the output voltage will be high (1).
- A NAND gate is called as "Universal Gate" because we can construct AND, OR and NOT gates using only NAND gates.
- The Boolean expression for the NAND gate is,

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

4.12.2 Universality of NAND Gate :

MU : Dec. 08, May 09, Dec. 09, Dec. 10, Dec. 16

University Questions

Q. 1 Using Boolean law prove NAND as universal gate. (Dec. 08, May 09, Dec. 10, 5 Marks)

Q. 2 Prove NAND as universal gate. (Dec. 09, 10 Marks)

Q. 3 Prove using Boolean algebra : "NAND gate is universal gate". (Dec. 16, 10 Marks)

In this section we are going to illustrate the conversion of each logic function into an expression using only NAND gates.

Note : A NAND gate is expressed mathematically as $Y = \overline{A \cdot B}$
Hence we have to bring the given Boolean expression into this form.

1. NOT gate (Inverter) using NAND gate :

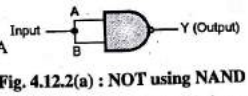
- Fig. 4.12.2(a) shows the realization of a NOT gate (inverter) using a two input NAND gate.
- As both the inputs of the NAND are connected together, we can write that,

$$\text{Input} = A = B = A$$

- So output is given by,

$$Y = \overline{A \cdot B} = \overline{A \cdot A} \quad \text{...since } A = B = A$$

But $A \cdot A = A$ $\quad \therefore Y = \overline{A}$



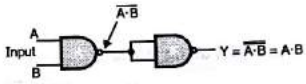
(B-505) **Fig. 4.12.2(a) : NOT using NAND**

- This expression is the Boolean expression for an inverter. Hence Fig. 4.12.2(a) is an inverter.

2. AND gate using NAND :

$$Y = A \cdot B \quad \text{...AND gate}$$

$$Y = \overline{\overline{A \cdot B}} \quad \text{...Double inversion}$$



(B-506) **Fig. 4.12.2(b) : AND gate using NAND**

$$\therefore Y = A \cdot B = \overline{\overline{A \cdot B}} \quad \text{...(4.12.1)}$$

- Equation (4.12.1) can be realized using only NAND gates as shown in Fig. 4.12.2(b).

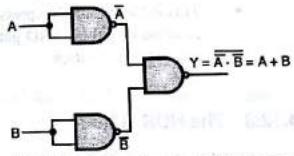
3. OR gate using NAND :

$$Y = A + B \quad \text{... OR gate}$$

$$Y = \overline{\overline{A + B}} \quad \text{... Double inversion}$$

$$\therefore Y = \overline{\overline{A} \cdot \overline{B}} \quad \text{... De Morgan's law}$$

This is the required expression.



(B-507) **Fig. 4.12.2(c) : OR gate using NAND gates**

- So $Y = A + B = \overline{\overline{A} \cdot \overline{B}}$ and Fig. 4.12.2(c) shows the realization.

4. NOR gate using NAND :

$$Y = \overline{A + B} \quad \text{... NOR gate}$$

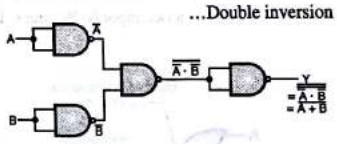
$$= \overline{A} \cdot \overline{B} \quad \text{...as per De Morgan's theorem}$$

$$Y = \overline{\overline{\overline{A} \cdot \overline{B}}} \quad \text{...Double inversion}$$

This is the required expression.

- Fig. 4.12.2(d) shows the realization of NOR gate using NAND gates.

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



(B-508) **Fig. 4.12.2(d) : NOR gate using only NAND gates**

5. EX-OR gate using NAND :

$$Y = A \oplus B = \overline{A}B + A\overline{B} \quad \text{... EX-OR gate}$$

$$Y = \overline{\overline{\overline{A}B + A\overline{B}}} \quad \text{...Double inversion}$$

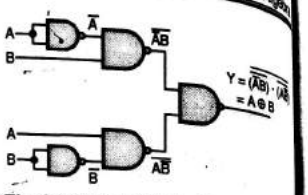
Let $X = \overline{A}B$ and $Z = A\overline{B}$. $\therefore Y = \overline{\overline{X + Z}}$

Scanned by CamScanner

- Using De Morgan's theorem $\overline{X+Z} = \bar{X}\cdot\bar{Z}$

$$\therefore\ Y = \overline{\bar{X}\cdot\bar{Z}} = \overline{(\overline{AB})\cdot(\overline{A\bar{B}})}$$

- This is the required expression for EX-OR using only NAND. Fig. 4.12.2(e) shows the realization.



(B-509) **Fig. 4.12.2(e) : EX-OR using NAND gates**

6. EX-NOR using NAND :

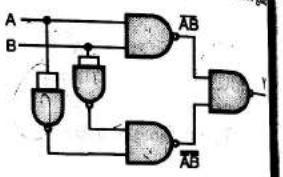
$$Y = A \odot B = \bar{A}\bar{B} + AB = X + Z$$

- Taking double inversion of RHS,

$$Y = \overline{\overline{X+Z}} = \overline{\bar{X}\cdot\bar{Z}} \text{ ...De Morgan's theorem}$$

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

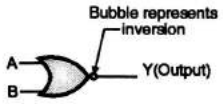
- This is the required expression for EX-NOR in terms of only NAND gates. Fig. 4.12.2(f) shows the realization.



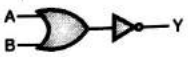
(B-510) **Fig. 4.12.2(f) : EX-NOR using only NAND gates**

4.12.3 The NOR Gate :

- This is another universal gate.
- The word NOR can be split as NOT-OR which means that a NOR operation can be implemented with the **combination of an OR gate and a NOT** gate, i.e. inverter. Thus a NOR gate is equivalent to an OR gate followed by an inverter as shown in Fig. 4.12.3(b).
- The symbol of a two input NOR gate is shown in Fig. 4.12.3(a), where a bubble (o) on the output side represents inversion.
- The truth table of a two input NOR gate is shown in Fig. 4.12.3(c), which shows that **"the output of a NOR gate is high (1) if and only if all its inputs are low (0) simultaneously".**
- The output of a two input NOR gate is low (0) if any one or all the inputs are at high (1) level.



(a) Logical symbol



(B-471) **(b) Equivalent circuit**

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

(c) Truth table

Fig. 4.12.3 : A two input NOR gate

- The Boolean expression for the NOR gate is given by,

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

4.12.4 Universality of NOR Gate :

MU : Dec. 08, May 09, Dec. 10, May 16

University Questions

Q. 1 Prove NOR as universal gate. **(Dec. 08, May 09, Dec. 10, 5 Marks)**

Q. 2 Implement Ex-OR gate using NOR gate only. **(May 16, 2 Marks)**

- In this section we are going to illustrate the implementation of every logic operation using only NOR gates. In order to do so, the Boolean expression of the given logic circuit must be first converted into the NOR format which is as follows,

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

- The implementation of various logic functions using only NOR gates is illustrated in Fig. 4.12.4.

1. NOT gate (inverter) using NOR :

- Fig. 4.12.4(a) shows the realization of a NOT gate using only NOR gate.
- As both the inputs of the NOR gate are connected together, we can write that,

$$A = B = A$$

- So output of NOR is given by,

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

- But $A + A = A$

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

(B-511) **Fig. 4.12.4(a) : NOT gate using NOR gate**

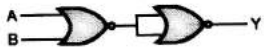
- This is the Boolean expression of an inverter. So Fig. 4.12.6(a) indeed represents an inverter.

2. OR using NOR :

$$Y = A + B \text{ ... OR gate}$$

$$Y = \overline{\overline{A+B}} \text{ ... Double inversion}$$

This is the required expression.



(B-512) **Fig. 4.12.4(b) : OR using only NOR gates**

- Fig. 4.12.4(b) shows the realization of OR gate using only NOR gates.

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

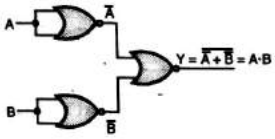
3. AND using only NOR gates :

$$Y = A\cdot B \text{ ... AND gate}$$

$$Y = \overline{\overline{A\cdot B}} \text{ ...Double inversion}$$

$$\therefore\ Y = \overline{\overline{A}+\overline{B}} \text{ ...De Morgan's theorem}$$

- This is the required expression, Fig. 4.12.4(c) shows the realization of AND gate using NOR gates only.



(B-513) **Fig. 4.12.4(c) : AND gate using only NOR gates**

4. NAND gate using only NOR gates :

- Boolean expression for a NAND gate is,

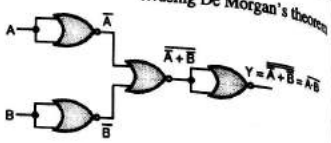
Scanned by CamScanner

$$Y = \overline{A \cdot B} = \bar{A} + \bar{B}$$ ...using De Morgan's theorem

- Take double inversion of RHS to get,

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

- This is the required expression. Fig. 4.12.4(d) shows the realization of NAND gate using only NOR gates.



(B-514) **Fig. 4.12.4(d) : NAND gate using only NOR gates**

5. EX-OR using only NOR :

$$Y = A \oplus B = \bar{A}B + A\bar{B}$$ ...EX-OR gate

- Let $X = \bar{A}B$ and $Z = A\bar{B}$

$$\therefore Y = X + Z$$

$$Y = \overline{\overline{X + Z}}$$ ...Double inversion

$$\therefore Y = \overline{\bar{X} \cdot \bar{Z}}$$ ...De Morgan's theorem

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

- But $\overline{\bar{A}B} = (A + \bar{B})$ and $\overline{A\bar{B}} = (\bar{A} + B)$ ...De Morgan's theorem

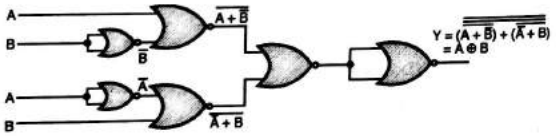
$$\therefore Y = \overline{(A + \bar{B}) \cdot (\bar{A} + B)}$$

$$\therefore Y = \overline{(A + \bar{B})} + \overline{(\bar{A} + B)}$$ ...De Morgan's theorem

- Take double inversion of RHS we get,

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

- This is the required expression. Fig. 4.12.4(e) shows the realization of EX-OR gate using only NOR gates.



(B-515) **Fig. 4.12.4(e) : EX-OR using only NOR gates**

6. EX-NOR using only NOR gates :

- Boolean expression for an EX-NOR gate is,

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

- Take double inversion of RHS to get,

$$Y = \overline{\overline{\bar{A}\bar{B} + AB}} = \overline{\overline{(\bar{A}\bar{B})} \cdot \overline{(AB)}}$$ ...De Morgan's theorem

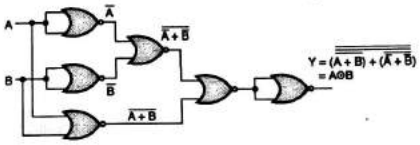
$$\therefore Y = \overline{(A + B) \cdot (\bar{A} + \bar{B})}$$ ...Using De Morgan's theorem

$$\therefore Y = \overline{(A + B)} + \overline{(\bar{A} + \bar{B})}$$ ...Using De Morgan's theorem

- Take double inversion of RHS to get,

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

- This is the required expression. Fig. 4.12.4(f) shows the realization of EX-NOR gate using only NOR gates.



(B-516) **Fig. 4.12.4(f) : EX-NOR using only NOR gates**

4.12.5 Solved Examples :

Ex. 4.12.1 : With sketch realize the expression Y = AB + CD by :

1. NAND gates only 2. NOR gates only.

Soln. :

1. **Using NAND gates only :** $Y = AB + CD$ ...Given

Let $AB = M$ and $CD = N$

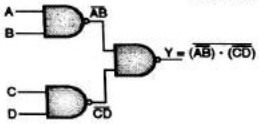
$Y = M + N$

Take double inversion of both the sides

$$\therefore Y = \overline{\overline{M + N}}$$

$$= \overline{\overline{M} \cdot \overline{N}}$$ ...As per De-Morgan's second law.

$$= \overline{\overline{AB} \cdot \overline{CD}}$$



(C-186) **Fig. P. 4.12.1(a)**

2. **Using NOR gates only :** $Y = AB + CD$

Take double inversion $\therefore Y = \overline{\overline{AB + CD}} = \overline{\overline{M + N}}$

$$\therefore Y = \overline{\overline{M} \cdot \overline{N}} = \overline{(\overline{AB}) \cdot (\overline{CD})}$$ ...(1)

Now let us use the De-Morgan's first law i.e.

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

$$\overline{AB} = \overline{A} + \overline{B} \text{ and } \overline{CD} = \overline{C} + \overline{D}$$

$$\therefore Y = \overline{(\overline{A} + \overline{B})\ (\overline{C} + \overline{D})}$$ ...(2)

Now let $\overline{A} + \overline{B} = P$ and $\overline{C} + \overline{D} = Q$

Scanned by CamScanner

$$\therefore\ Y = \overline{P \cdot Q}$$

Use De-Morgan's first law again to write

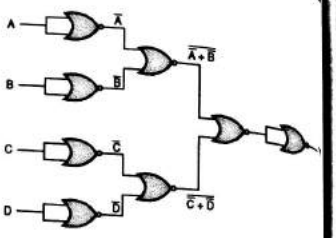
$$Y = \overline{P} + \overline{Q}$$

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

Take double inversion of RHS to write

$$Y = \overline{\overline{\overline{A} + \overline{B}} + \overline{\overline{C} + \overline{D}}} \quad \text{...Ans.}$$

The logic diagram for this expression is shown in Fig. P. 4.12.1(b).



(C-187) **Fig. P. 4.12.1(b)**

Ex. 4.12.2 : Realize the following Boolean expression using only NAND gates : $Y = (AB + BC)\,C$

Soln. :

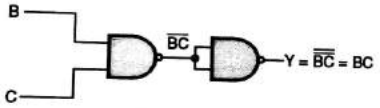
$$Y = (AB + BC)\,C = ABC + BCC$$

$$= ABC + BC \quad \text{...since } C \cdot C = C$$

$$= BC\,(A + 1)$$

$$\therefore\ Y = BC \quad \text{...since } A + 1 = 1$$

$$\therefore\ Y = \overline{\overline{BC}} \quad \text{...as } \overline{\overline{A}} = A$$



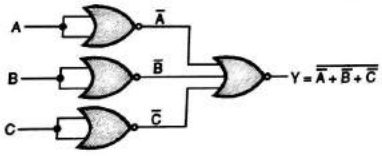
(C-188) **Fig. P. 4.12.2**

This expression can be realized using only NAND gates as shown in Fig. P. 4.12.2.

Ex. 4.12.3 : Realize the following expression using only NOR gates :

$$Y = (ABC + \overline{B}\,\overline{C})\,C$$

Soln. :



(C-189) **Fig. P. 4.12.3**

- The given equation is,

$$Y = (ABC + \overline{B}\,\overline{C})\,C$$

$$\therefore\ Y = ABCC + \overline{B}\,\overline{C}\,C$$

$$= ABC + 0 \quad \text{...since } \overline{C}C = 0 \text{ and } C \cdot C = C$$

$$\therefore\ Y = ABC = \overline{\overline{ABC}}$$

- Using De-Morgan's first theorem we get,

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

- This expression can be realized using NOR gates as shown in Fig. P. 4.12.3.

Ex. 4.12.4 : Reduce the following expression, implement it using basic logic gates and then implement it using only NAND gates.

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

Soln. : Using De-Morgan's theorems inside the bracket we get,

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

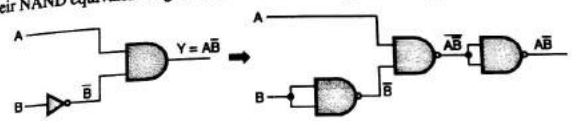
$$= \overline{A}\,A\,\overline{B} + \overline{B}\,A\,\overline{B} + \overline{A}\,A\,\overline{B}\,\overline{B}$$

$$= 0 + A\,\overline{B} + 0$$

...since $\overline{A}\,A = 0$ and $B\,\overline{B} = 0$ and $\overline{B} \cdot \overline{B} = \overline{B}$

$$\therefore\ Y = A\,\overline{B} \quad \text{...Ans.}$$

This equation is realized as shown in Fig. P. 4.12.4(a). Now replace the inverter and the AND gate by their NAND equivalent to get the circuit shown in Fig. P. 4.12.4(b).



(a) Realization using the basic gates **(b) Realization using the NAND gates**

(C-192) **Fig. P. 4.12.4**

Ex. 4.12.5 : Realize $y = AB + \overline{AB}$ using NAND gates only. **May 15, 2 Marks**

Soln. :

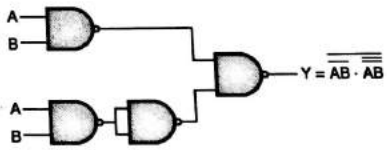
$$Y = A \cdot B + \overline{AB}$$

Taking double inversion of RHS,

$$= \overline{\overline{AB + \overline{AB}}}$$

$$= \overline{\overline{AB} \cdot \overline{\overline{AB}}}$$

Implementation using NAND gates is as shown in Fig. P. 4.12.5.



(C-5188) **Fig. P. 4.12.5 : Implementation using NAND gates**

Review Questions

Q. 1 Explain with circuit diagrams a two input EX-NOR gate using only NAND gates.

Q. 2 Write the applications of EX-OR gate.

Q. 3 Explain the AND gate. Specify its symbol and write its truth table.

Q. 4 Explain the OR gate. Specify its symbol and write its truth table.

Q. 5 Explain the EX-OR gate. Specify its symbol and write its truth table.

Q. 6 Explain the NAND gate. Specify its symbol and write its truth table.

Q. 7 Explain the NOR gate. Specify its symbol and write its truth table.

Scanned by CamScanner

Q. 8 Explain and mention the universal gates.
Q. 9 Name the universal and derived gates.
Q. 10 Why are NAND and NOR gates called universal gates ?
Q. 11 Write the truth table of following :
1. Three input EX-OR gate 2. Three input EX-NOR gate
Q. 12 State AND laws.
Q. 13 State OR laws.
Q. 14 State De Morgan's theorems.
Q. 15 What is AND-OR-NOT logic ?
Q. 16 Define truth table.
Q. 17 Write the Boolean expressions for the following :
1. OR gate 2. EX-NOR gate
Q. 18 Explain with circuit diagrams a two input EX-NOR gate using only NOR gates.
Q. 19 Draw the equivalent circuit of all gates using NAND gate.
Q. 20 Draw the equivalent circuit of all gates using NOR gate.
Q. 21 Define 'truth table'. What is a logic gate ? Classify the logic gates.
Q. 22 State the rules of Boolean algebra.
Q. 23 State and explain the commutative law.
Q. 24 State and explain the associative law.
Q. 25 State and explain the distributive law.
Q. 26 Write a note on : Duality theorem.
Q. 27 State and prove De-Morgan's first and second theorem.
Q. 28 Prove that OR-AND configuration is equivalent to NOR- NOR configuration.
Q. 29 Explain the rules of Boolean algebra.

CHAPTER 5

Logic Minimization and Reduction Techniques

Module 2

Syllabus :
K-map method : 2 Variable, 3 Variable, 4 Variable, Don't care condition, Quine Mc-Clusky method, NAND- NOR realization.

5.1 System or Circuit :

- A system or circuit is defined as the physical device or group of devices or algorithm which performs the required operations on the signal applied at its input which can be either analog or digital.
- Depending on the type of input signal (analog or digital) the systems or circuits can be classified into two types :
  1. Analog systems 2. Digital systems

5.1.1 Digital Systems :

- We have already defined the digital signals.
- Now we define the digital system as the system which processes or works upon the digital signals to produce another digital signal.

(C-215) **Fig. 5.1.1 : A digital system**

- Thus the input signal to a digital system is digital and its output signal is also digital.

5.1.2 Input Output Relation :

MU : Dec. 02

University Questions
Q. 1 Explain the term input variable. (Dec. 02, 1 Mark)

- In Fig. 5.1.1, A, B, C ... are the inputs whereas Y or f (A, B ...) represents the output of the system. The system may have a single output or it can have multiple outputs.
- The system output is a function of inputs. Hence it is denoted by f (A, B, C, ...)
- The relation between input and output can be represented in various different ways as given below :
  1. Truth table 2. Switching equations 3. Logic diagram

Scanned by CamScanner

1. **Truth table :**

- Here the relation between the input variables (A, B, C ...) with the output Y is presented in a tabular form as shown in Table 5.1.1.
- The state of output (0 or 1) is written for all the possible combinations of inputs.

Table 5.1.1

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	1
:			:
:			:
1	1	1	1

2. **Logic diagram :**

This is a diagrammatic way of expressing the input-output relationship of a digital circuit.

3. **Switching equations :**

- The relation between inputs and output(s) can be presented in the form of equation(s) called as switching equations.
- The input variables are called as switching variables. The output (Y) is written on the left hand side of the equation whereas the terms containing the input variables are written on the right hand side of the switching equation as shown below :

$$Y = ABC + A\bar{B}C + \bar{A}\bar{B}\bar{C} \quad ...(5.1.1)$$

$$\text{Or } f(A, B, C) = AB\bar{C} + \bar{A}BC + A\bar{B}\bar{C} \quad ...(5.1.2)$$

$$\text{Or } f(A, B, C) = (A + \bar{B} + C) \cdot (A + B + C) \quad ...(5.1.3)$$

- The switching equations are also called as the Boolean equations or the **system equations**.
- The system equations or switching equations can be of two different types :
  1. Sum of Products (SOP) format 2. Product of Sums (POS) format

5.1.3 Types of Digital Systems :

The digital systems in general are classified into two categories namely :

1. Combinational logic circuits 2. Sequential logic circuits.

Combinational circuits :

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuits do not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit.
- The sequence in which the inputs are being applied has no effect on the output of a combinational circuit.
- A combinational circuit is a logic circuit, the output of which depends only on the combination of the inputs. The output does not depend on the past value of inputs or outputs. Hence combinational circuits do not require any memory.

(C-332) **Fig. 5.1.2 : Block diagram of a combinational circuit**

- The block diagram of a combinational circuit is shown in Fig. 5.1.2.

- A combinational circuit can have a number of inputs and a number of outputs. The circuit of Fig. 5.1.2 has "n" inputs and "m" outputs.
- Between the inputs and outputs, logic gates are connected so combinational circuit basically consists of logic gates.
- A combinational circuit operates in three steps :
  1. It accepts n-different inputs. 2. The combination of gates operates on the inputs.
  3. "m" different outputs are produced as per requirement.

Examples of combinational circuits :

Following are the examples of some combinational circuits :

1. Adders, subtractors 2. Comparator
3. Code converters 4. Encoders, decoders 5. Multiplexers, demultiplexers

5.1.4 Combinational Circuit Design :

- In this chapter we are going to discuss the design of combinational circuits.
- To design a combinational circuit we have been given the specifications or the requirements of combinational circuits. Such specifications or requirements can be specified in the following ways :
  1. A list of statements 2. Boolean expressions 3. Truth table.
- From the specified requirements we have to design a combinational circuit using a combination of gates which will fulfill all the requirements.
- We can adopt one of the following two approaches to the combinational circuit design :
  1. Traditional methods 2. Use of Medium Scale Integration (MSI)
- In this chapter we will discuss the traditional method.

Steps followed in traditional circuit design method :

1. A truth table is given. 2. Obtain the Boolean expression from the truth table.
3. Simplify the Boolean equation using standard methods.
4. Realize the simplified equation using gates i.e. build the logic circuit.

Methods to simplify the boolean equations :

- The methods used for simplifying the Boolean functions are as follows :
  1. Algebraic method. 2. Karnaugh-map (K-map) simplification.
  3. Quine-Mc Cluskey method and 4. Variable Entered Mapping (VEM) technique.
- Out of these, the method of algebraic reduction using Boolean algebra has been already discussed in the earlier chapters.
- The Boolean theorems and De-Morgan's theorems are useful in simplifying the given Boolean expressions. We can then realize the logical expressions using either the conventional gates or universal gates.
- We should use the minimum number of logic gates for the realization of a logical expression.
- This is possible if we can simplify the logical expressions. In this chapter we will discuss **Karnaugh map or K-map** and Quine Mc-Cluskey method.

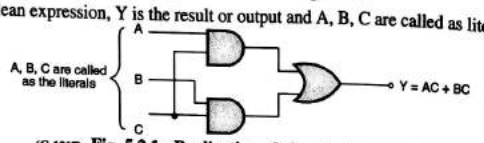
5.2 Standard Representations for Logical Functions :

- Consider that the logic expression given to us is as follows :

$$Y = AC + BC$$

Scanned by CamScanner

- Then it can be realized using basic gates as shown in Fig. 5.2.1.
- In this Boolean expression, Y is the result or output and A, B, C are called as literals.

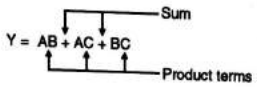


(C-1217) **Fig. 5.2.1 : Realization of given logic expression**

- When we realize the Boolean equation by using gates, each literal acts as an input as shown in Fig. 5.2.1.
- Any logic expression can be expressed in the following two standard forms :
  1. Sum-of-Products form (SOP) and 2. Product-of-Sums form (POS)
- These two forms are suitable for reducing the given logic expression to its simplest form.

5.2.1 Sum-of-Products (SOP) Form :

- Refer to logic expression shown in Fig. 5.2.2. It is in the form of sum of three terms AB, AC and BC with each individual term is product of two variables. Say A·B or A·C etc.



(C-1217(a)) **Fig. 5.2.2 : Sum-of-Products (SOP) form**

- Therefore such expressions are known as expression in SOP form.
- The sums and products in the SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions.
- A, B and C are the literals or the inputs of the combinational circuit.
- Some more examples of the SOP expressions are as follows :

$$Y = ABC + BCD + AB\bar{D}$$

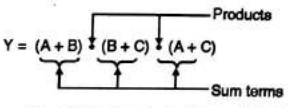
$$A = XY + \bar{X}Y + X\bar{Y}$$

$$Y = \bar{P}\bar{Q} + PQR + PQ\bar{R}$$

- Thus in each product term there can be one or more than one literals ANDed together. The literals can be in their complemented or uncomplemented form.

5.2.2 Product of the Sums Form (POS) :

- Refer to the logic expression shown in Fig. 5.2.3. It is in the form of product of three terms (A + B), (B + C) and (A + C), with each term is in the form of sum of two variables.
- Such expressions are said to be in the Product of Sums (POS) form.
- Some other examples of POS expressions form are as follows :



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

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

$$Y = (P + R) \cdot (P + \bar{Q}) \cdot (\bar{P} + R)$$

(C-1218) **Fig. 5.2.3 : Product of Sums (POS) form**

- The literals are ORed together to form the sum terms and the sum terms are ANDed to get the expression in the POS form.

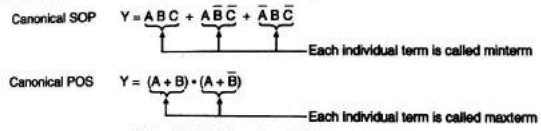
5.2.3 Standard or Canonical SOP and POS Forms : MU : May 10

University Questions

Q. 1 What is canonical SOP and POS form ? Explain with an example. **(May 10, 5 Marks)**

Canonical or standard forms :

- The word standard is used in order to describe a condition of switching equation. The meaning of the word standard is conforming to a general rule.
- This rule states that each term used in a switching equation must contain all the available input variables.
- The two formats of a switching equation in the standard form are :
  1. Sum of Product (SOP) format. 2. Product of Sum (POS) format.
- When we simplify a Boolean equation, sometimes an input variable is eliminated to simplify the equation.
- But standard expressions are not simplified. It is said that the standard expression is the opposite of simplification. So it contains redundancies.
- Many times, switching equations written in the SOP or POS form are not standard. That means each term may not contain all the input variables.
- Consider the SOP and POS expressions shown in Fig. 5.2.4.



(C-1220) **Fig. 5.2.4 : Standard SOP and POS forms**

- Referring to Fig. 5.2.4 we can say that a logic expression is said to be in the **standard** SOP or POS form if each product term (for SOP); and sum term (for POS) consists of all the literals in their complemented or uncomplemented form.

Non-standard forms :

- The two standard forms discussed earlier are the basic forms that are obtained from the truth table. However these forms are not used often because these equations are not in the minimized form because each term contains all the literals.
- There is another way to express the Boolean functions. It is called as the **non-standard form.** In this form each term may contain one, two or any number of literals. It is not necessary that each term should contain all the literals.
- There can be two types of non-standard forms :
  1. Non-standard SOP form. 2. Non-standard POS form.
- The examples of standard and non-standard SOP and POS expressions are given in Table 5.2.1.

Table 5.2.1 : Non-standard and standard SOP and POS forms

Sr. No.	Expression	Type
1.	$Y = AB + AB\bar{C} + \bar{A}BC$	Non-standard SOP
2.	$Y = AB + A\bar{B} + \bar{A}\bar{B}$	Standard SOP
3.	$Y = (\bar{A} + B) \cdot (A + B) \cdot (A + \bar{B})$	Standard POS
4.	$Y = (\bar{A} + B) \cdot (A + B + C)$	Non-standard POS

Scanned by CamScanner

5.2.4 Conversion of a Logic Expression to Standard SOP or POS Form :

Let us see how to convert the given non-standard SOP and POS expressions into the corresponding non-standard SOP and POS forms.

Conversion from Non-standard SOP to Standard SOP form :

The procedure to be followed for converting a non-standard SOP expression into a standard SOP expression is as follows :

Steps to be followed :

Step 1 : For each term in the given non-standard SOP expression find the missing literal.
Step 2 : Then AND this term with the term formed by ORing the missing literal and its complement.
Step 3 : Simplify the expression to get the standard SOP expression.

Ex. 5.2.1 : Convert the expression $Y = AB + A\bar{C} + BC$ into the standard SOP form.

Soln. :

The given expression has 3-input variables A, B and C.

Step 1 : Find the missing literal for each term :

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

Missing literal → C B A

(C-2383)

Step 2 : AND each term with (Missing literal + Its complement) :

ANDING

$$Y = \underbrace{AB}_{\text{Original product term}} \cdot \underbrace{(C + \bar{C})}_{\text{(Missing literal + Its complement)}} + A\bar{C}\,(B + \bar{B}) + BC\,(A + \bar{A})$$

(C-2384)

Step 3 : Simplify the expression to get the standard SOP :

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

But $A + A = A \quad \therefore (ABC + ABC) = ABC$ and $(AB\bar{C} + AB\bar{C}) = AB\bar{C}$

$$\therefore Y = \underbrace{ABC + AB\bar{C} + A\bar{B}\bar{C} + \bar{A}BC}_{\text{Each term contains all the literals}} \quad \text{Standard SOP form}$$

Conversion from Non-standard POS to standard POS form :

The conversion of non-standard POS expression into standard POS form can be obtained by following the steps given below :

Steps to be followed :

Step 1 : For each term in the given non-standard POS expression, find the missing literal.
Step 2 : Then OR each such term with the term formed by ANDing the missing literal in that term with its complement.
Step 3 : Simplify the expression to get the standard POS.

Ex. 5.2.2 : Convert the expression $Y = (A + B)(A + C)(B + \bar{C})$ into standard POS form.

Dec. 13. 5 Marks

Soln. :

Step 1 : Find the missing literal for each term :

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

Missing literal → C B A

(C-2384)

Step 2 : OR each term with (Missing literal. Its complement) :

$$Y = (\underbrace{A + B}_{\text{This term is ORed with the term formed by ANDing the missing literal with its complement.}} + \underbrace{C\bar{C}}_{\text{Missing literal ANDed with its complement}}) \cdot (A + C + B\bar{B}) \cdot (B + \bar{C} + A\bar{A})$$

Step 3 : Simplify the expression to get standard POS :

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

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

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

But $A \cdot A = A \quad \therefore (A + B + C)(A + C + B) = (A + B + C)$

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

$$\therefore Y = \underbrace{(A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + \bar{C})}_{\text{Each term contains all the literals.}} \quad \text{Standard POS form}$$

Ex. 5.2.3 : Convert the following expressions into their standard SOP or POS forms :

1. $Y = AB + AC + BC$ 2. $Y = (A + B)(\bar{B} + C)$ 3. $Y = A + BC + ABC$

Soln. :

1. $Y = AB + AC + BC$:

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

$ABC + ACB + BCA = ABC$ .....because $A + A = A$

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

This is the required expression in standard SOP form.

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

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

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

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

This is in the standard POS form.

Scanned by CamScanner

3. $Y = A + BC + ABC$:

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

But $A + A = A$ so $(ABC + BCA + ABC) = ABC$

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

This expression is in the standard SOP form.

5.3 Concepts of Minterm and Maxterm :

MU : Dec. 02

University Questions

Q. 1 Explain the terms : 1. Minterm 2. Maxterm **(Dec. 02, 2 Marks)**

Minterm : Each individual term in the standard SOP form is called as **minterm.** This is shown in Fig. 5.3.1.

Maxterm : Each individual term in the standard POS form is called as **maxterm.** This is shown in Fig. 5.3.1.

Standard SOP $Y = \underbrace{ABC} + \underbrace{A\bar{B}\bar{C}} + \underbrace{\bar{A}B\bar{C}}$ — Each individual term is called minterm

Standard POS $Y = \underbrace{(A+B)} \cdot \underbrace{(A+\bar{B})}$ — Each individual term is called maxterm

(C-1220(a)) **Fig. 5.3.1 : Concept of maxterm and minterm**

Table 5.3.1 : Minterms and maxterms for three variables

Variables			Minterms	Maxterms
A	B	C	m_i	M_i
0	0	0	$\bar{A}\bar{B}\bar{C} = m_0$	$A+B+C = M_0$
0	0	1	$\bar{A}\bar{B}C = m_1$	$A+B+\bar{C} = M_1$
0	1	0	$\bar{A}B\bar{C} = m_2$	$A+\bar{B}+C = M_2$
0	1	1	$\bar{A}BC = m_3$	$A+\bar{B}+\bar{C} = M_3$
1	0	0	$A\bar{B}\bar{C} = m_4$	$\bar{A}+B+C = M_4$
1	0	1	$A\bar{B}C = m_5$	$\bar{A}+B+\bar{C} = M_5$
1	1	0	$AB\bar{C} = m_6$	$\bar{A}+\bar{B}+C = M_6$
1	1	1	$ABC = m_7$	$\bar{A}+\bar{B}+\bar{C} = M_7$

- Table 5.3.1 gives the minterms and maxterms for a three variable/literal logic function. Let Y be the output and A, B, C be the inputs.
- The number of minterms and maxterms is $2^3 = 8$. In general for "n" number of variables the number of minterms or maxterms will be 2^n
- Each minterm is represented by m_i where $i = 0, 1, \ldots, 2^n - 1$ and each maxterm is represented by M_i.

5.3.1 How are the Minterm and Maxterm of Table 5.3.1 Obtained ?

Writing minterm for a particular combination of ABC :

- Consider ABC = 011
- Assume that A, B, C are input to an AND gate.
- We want the output of the AND gate to be 1. For that all its inputs should be 1.

- So take the complement of that input which is 0 i.e. A in this case and write the minterm.

$$\therefore\ \text{Minterm} = \bar{A}BC \text{ corresponding to } ABC = 0\,1\,1$$

- Similarly other minterms in Table 5.3.1 can be obtained.

Writing maxterm for a particular combination of ABC :

- Let ABC = 011.
- Assume that ABC are inputs to an OR gate.
- We want the output of the OR gate to be 0. So all the inputs to the OR gate should be 0s.
- So invert the inputs which are 1s. i.e. B and C in this case and write the maxterm.

$$\therefore\ \text{Maxterm} = (A + \bar{B} + \bar{C}) \text{ corresponding to } ABC = 0\,1\,1$$

- Similarly other maxterms in Table 5.3.1 can be obtained.

Ex. 5.3.1 : For the truth table of two variables write the minterms and maxterms.

Soln. : Refer Table P. 5.3.1 for solution.

Table P. 5.3.1 : Solution to Ex. 5.3.1

Variables/literals		Minterms	Maxterms
A	B	m_i	M_i
0	0	$m_0 = \bar{A}\bar{B}$	$M_0 = A + B$
0	1	$m_1 = \bar{A}B$	$M_1 = A + \bar{B}$
1	0	$m_2 = A\bar{B}$	$M_2 = \bar{A} + B$
1	1	$m_3 = AB$	$M_3 = \bar{A} + \bar{B}$

5.3.2 Representation of Logical Expressions using Minterms and Maxterms :

We can represent the logical expression using the minterms and maxterms as follows :

1. $$Y = \underbrace{ABC}_{m_7} + \underbrace{\bar{A}BC}_{m_3} + \underbrace{A\bar{B}\bar{C}}_{m_4} \quad \leftarrow \text{Given logic expression}$$

$\leftarrow$ Corresponding minterms

$$\therefore\ Y = m_7 + m_3 + m_4$$
$$\therefore\ Y = \Sigma m(3, 4, 7) \leftarrow \text{other way of representation}$$

where Σ denotes sum of products.

2. $$Y = \underbrace{(A+\bar{B}+C)}_{M_2}\ \underbrace{(A+B+C)}_{M_0}\ \underbrace{(\bar{A}+\bar{B}+C)}_{M_6} \leftarrow \text{Given expression}$$

$\leftarrow$ Corresponding maxterms

$$\therefore\ Y = M_2\ M_0\ M_6$$
$$\therefore\ Y = \Pi M(0, 2, 6) \leftarrow \text{Other way of representation.}$$

where Π denotes product of sums.

5.3.3 Writing SOP and POS Forms for a Given Truth Table :

- We know that a logic expression can be represented in the truth table form. For example, the expression $Y = AB + \bar{A}\bar{B}$ which is the Boolean expression for an EX-NOR gate can also be represented using a truth table.
- Hence it is possible to obtain the logic expression in the standard SOP or POS form if a truth table is given to us.

Scanned by CamScanner

5.3.4 To Write Standard SOP Expression for a Given Truth Table :

The procedure to be followed for writing the standard SOP expression from a given truth table is as follows :

Step 1 : From the given truth table, consider only those combinations of inputs which produce a output Y = 1.

Step 2 : Write down a product term interms of input variables for each such combination.

Step 3 : OR all these product terms produced in step 2 to get the standard SOP.

Ex. 5.3.2 : From the truth table P. 5.3.2, obtain the logical expression in the standard SOP form.

Table P. 5.3.2 : Given truth table

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

Soln. :

Step 1 : Consider only those combinations of inputs which correspond to Y = 1 :

Steps 2 and 3 : For the second and the third entries in Table P. 5.3.2(a) write the product terms :

Table P. 5.3.2(a)

	A	B	Y	
1	0	0	0	
2	0	1	1	$Y_1 = \bar{A}B$
3	1	0	1	$Y_2 = A\bar{B}$
4	1	1	0	

Boolean expressions in the product forms

OR (Add) all the product terms to write the final expression in standard SOP form as follows :

$$\therefore \quad Y = Y_1 + Y_2 = \bar{A}B + A\bar{B}$$

$$\therefore \quad Y = m_1 + m_2 = \Sigma m(1, 2)$$

Ex. 5.3.3 : For the truth table shown in Table P. 5.3.3 write the logic expression in the standard SOP form.

Soln. : Table P. 5.3.3 : Given truth table

A	B	C	Y	
0	0	0	0	
0	0	1	1	$\bar{A}\bar{B}C\ (m_1)$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\bar{B}\bar{C}\ (m_4)$
1	0	1	0	
1	1	0	0	
1	1	1	1	$ABC\ (m_7)$

Step 1 : Product terms corresponding to combinations of inputs for which Y = 1.

Step 2 : OR (Add) all the product terms :

ORing all the product terms we get,

$$Y = \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC \quad \text{...Ans.}$$

$$\therefore \quad Y = m_1 + m_4 + m_7 = \Sigma m(1, 4, 7)$$

5.3.5 To Write a Standard POS Expression for a Given Truth Table :

Follow the procedure given below to get the expression in standard POS expression from a given truth table.

Step 1 : From the given truth table, consider only those combinations of inputs which produce a low output (Y = 0).

Step 2 : Write the maxterms only for such combinations.

Step 3 : AND these maxterms to obtain the expression in standard POS form.

Ex. 5.3.4 : Write the logic expression in standard POS form for the truth table shown in Table P. 5.3.4.

Soln. :

Table P. 5.3.4 : Given truth table

A	B	C	Y	
0	0	0	0	$A + B + C\ (M_0)$
0	0	1	1	
0	1	0	1	
0	1	1	0	$A + \bar{B} + \bar{C}\ (M_3)$
1	0	0	1	
1	0	1	0	$\bar{A} + B + \bar{C}\ (M_5)$
1	1	0	0	$\bar{A} + \bar{B} + C\ (M_6)$
1	1	1	1	

Step 1 : Write maxterms for the combinations of input which produce Y = 0.

Step 2 : AND (take product of) all the maxterms to get standard POS form :

- ANDing (taking product of) all the maxterms written in step 1 we get,

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

$$\therefore \quad Y = M_0 \cdot M_3 \cdot M_5 \cdot M_6 = \pi(0, 3, 5, 6)$$

5.3.6 Conversion from SOP to POS and Vice Versa :

It is important to note that the SOP and POS forms written for the same truth table are always logically equivalent. This point can be proved by solving the following example.

Ex. 5.3.5 : For the given truth table write the logical expressions in the standard SOP and POS forms and prove their equivalence.

Soln. : **Table P. 5.3.5 : Given truth table**

Minterms	A	B	C	Y	Maxterms
	0	0	0	0	$A + B + C$
$\bar{A}\bar{B}C$	0	0	1	1	
	0	1	0	0	$A + \bar{B} + C$
	0	1	1	0	$A + \bar{B} + \bar{C}$
$A\bar{B}\bar{C}$	1	0	0	1	
	1	0	1	0	$\bar{A} + B + \bar{C}$
	1	1	0	0	$\bar{A} + \bar{B} + C$
ABC	1	1	1	1	

Step 1 : Minterms and maxterms for the combinations of inputs producing Y = 1 and 0 respectively.

Step 2 : Write the standard SOP and POS expressions :

Standard SOP form : $Y = \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

Scanned by CamScanner

Standard POS form : $Y = (A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)$

Step 3 : To prove the equivalence between SOP and POS forms :

Consider the standard POS form.

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

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

$$\therefore Y = \underset{(1)}{(A+C)}\underset{(2)}{(A+\bar{B}+\bar{C})}\underset{(3)}{(\bar{A}+B+\bar{C})}\underset{(4)}{(\bar{A}+\bar{B}+C)}$$

Multiply the terms (1×2) and (3×4) to get,

$$Y = [AA + A\bar{B} + A\bar{C} + AC + \bar{B}C + C\bar{C}]$$
$$[\bar{A}\bar{A} + \bar{A}\bar{B} + \bar{A}C + \bar{A}B + B\bar{B} + BC + \bar{A}\bar{C} + \bar{B}\bar{C} + C\bar{C}]$$

But $AA = A,\ C\bar{C} = 0,\ \bar{A}\bar{A} = \bar{A},\ B\bar{B} = 0,\ C\bar{C} = 0$

$$\therefore Y = [A + A\bar{B} + A\bar{C} + AC + \bar{B}C][\bar{A} + \bar{A}\bar{B} + \bar{A}C + \bar{A}B + BC + \bar{A}\bar{C} + \bar{B}\bar{C}]$$
$$= [A(1+\bar{B}) + A(\bar{C}+C) + \bar{B}C][(1+\bar{B})\bar{A} + \bar{A}(C+\bar{C}) + \bar{A}B + BC + \bar{B}\bar{C}]$$

But $(1+\bar{B}) = 1,\ (\bar{C}+C) = 1,\ (1+\bar{B}) = 1$

$$\therefore Y = [A + A + \bar{B}C][\bar{A} + \bar{A} + \bar{A}B + BC + \bar{B}\bar{C}]$$

But $A + A = A$ and $\bar{A} + \bar{A} = \bar{A}$

$$\therefore Y = (A+\bar{B}C)[\bar{A} + \bar{A}B + BC + \bar{B}\bar{C}] = (A+\bar{B}C)[\bar{A}(1+B) + BC + \bar{B}\bar{C}]$$

But $(1+B) = 1$

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

But $A\bar{A} = 0,\ (\bar{B}C\cdot BC) = 0$ and $(\bar{B}C\cdot\bar{B}\bar{C}) = 0$

$$\therefore Y = ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}C$$ ← This is same as standard SOP form. Thus the equivalence between SOP and POS forms is proved.

Step 4 : Express output in terms of minterms and maxterms :

The output can be expressed in terms of minterms and maxterms as follows :

$$Y = m_1 + m_4 + m_7 \quad \text{...In terms of minterms}$$
$$= \Sigma m(1, 4, 7)$$
$$\text{and } Y = M_0\cdot M_2\cdot M_3\cdot M_5\cdot M_6 \quad \text{...In terms of maxterms}$$
$$= \Pi M(0, 2, 3, 5, 6)$$

Due to equivalence between SOP and POS forms we can write,

$$\Sigma m(1, 4, 7) = \Pi M(0, 2, 3, 5, 6) \quad \text{...(5.3.1)}$$

Complementary relationship between minterms and maxterms :

- From Equation (5.3.1) we can conclude that the relationship between the expressions expressed using minterms and maxterms is complementary relationship.
- We can exploit this complementary relationship to write the expression in terms of maxterms if the expression in terms of minterms is known and vice versa.

- For example, if a SOP expression for 4-variables is given by,

$$Y = \Sigma m(0, 1, 3, 5, 6, 7, 11, 12, 15)$$

Then we can get the equivalent POS expression using the complementary relationship as follows :

$$Y = \Pi M(2, 4, 8, 9, 10, 13, 14)$$

Ex. 5.3.6 : Express the equation in standard SOP form : $F(A, B, C) = \Pi M(0, 2, 5, 7)$.

Dec. 16. 4 Marks

Soln. :

$$F(A, B, C) = \Pi M(0, 2, 5, 7) \quad \text{...(Given standard POS form)}$$

$\therefore$ Equation in standard SOP form is $\Sigma m(1, 3, 4, 6)$ **...Ans.**

5.4 Methods to Simplify the Boolean Functions :

- The methods used for simplifying the Boolean expressions are as follows :
  1. Algebraic method.
  2. Karnaugh-map simplification.
  3. Quine Mc-Cluskey method and
  4. Variable Entered Mapping (VEM) technique.
- The Boolean theorems and De-Morgan's theorems are useful in simplifying the given Boolean expressions. We can then realize the logical expressions using either the standard gates or universal gates.
- We should use the minimum possible number of logic gates for the realization of a logical expression.
- This is possible if we can simplify the logical expressions. In this chapter we will discuss one of the simplification techniques called **Karnaugh map or K-map** and the **Quine Mc-Cluskey Method** or the **Tabular Method.**

5.4.1 Algebraic Simplification :

- In the previous chapter we have studied the Boolean laws and De-Morgan's theorems.
- We can use them to simplify the given Boolean expression in the following way. The most important thing is to convert the given expression into SOP form.

Standard procedure for algebraic simplification :

Step 1 : Bring the given expression into the SOP form by using the Boolean laws and De-Morgan's theorems.

Step 2 : Simplify this SOP expression by checking the product terms for common factors.

Ex. 5.4.1 : Simplify the expression given below : $Y = AB + (A+B)(\bar{A}+B)$.

Soln. :

Step 1 : Bring the given expression in SOP form :

Given expression : $Y = AB + (A+B)(\bar{A}+B) = AB + (A\bar{A} + AB + \bar{A}B + BB)$

Step 2 : Search for common factors and simplify :

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

But $AB + AB = AB,\ BB = B$ and $A\bar{A} = 0$

Scanned by CamScanner

$$\therefore \quad Y = AB + B + \bar{A}B = B(A+1) + \bar{A}B$$

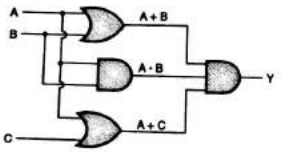
But $(A+1) = 1$

$$\therefore \quad Y = B + \bar{A}B = B(1+\bar{A}) = B \qquad \text{...since } (1+\bar{A}) = 1$$

$$\therefore \quad Y = B$$

This is simplified expression. ...Ans.

Ex. 5.4.2 : For the logic circuit shown in Fig. P. 5.4.2 write the Boolean expression and simplify it.



(C-1221) **Fig. P. 5.4.2 : Given logic circuit**

Soln. :

***Step 1 :* Write the Boolean expression :**

The expression for output of the given logic circuit is,

$$Y = (A+B)(AB)(A+C)$$

***Step 2 :* Bring this expression in SOP form :**

Multiply the terms to get the expression into SOP form.

$$\therefore \quad Y = (A+B)(AAB + ABC)$$

But $AA = A$ $\quad \therefore \quad Y = (A+B)(AB+ABC) = AAB + AABC + BAB + BABC$

$$= AB + ABC + AB + ABC = AB + AB + ABC + ABC$$

But $AB + AB = AB$ and $ABC + ABC = ABC$

$$\therefore \quad Y = AB + ABC = AB(1+C)$$

$$Y = AB \qquad \text{...since } 1 + C = 1.$$

This is the simplified expression.

Ex. 5.4.3 : Simplify the following three variable Boolean expression.
$Y = \sum m(2, 4, 6)$

Soln. : The given expression can be written in SOP form as follows :

$$Y = m_2 + m_4 + m_6 = \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} = \bar{A}B\bar{C} + A\bar{C}(\bar{B}+B)$$

But $\bar{B} + B = 1$ $\quad \therefore \quad Y = \bar{A}B\bar{C} + A\bar{C}$

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

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

$$\therefore \quad Y = \bar{C}(A+\bar{A})(A+B) = \bar{C}(1)(A+B)$$

$$\therefore \quad Y = \bar{C}(A+B) \qquad \text{...Ans.}$$

This is simplified expression.

Ex. 5.4.4 : Simplify the following three variable logic expression, $Y = \Pi M(1, 3, 5)$

Soln. : The given expression can be expressed in terms of maxterms as,

$$Y = M_1 \cdot M_3 \cdot M_5 = (A+B+\bar{C})(A+\bar{B}+\bar{C})(\bar{A}+B+\bar{C})$$

***Step 1 :* Bring the expression into SOP form :**

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

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

***Step 2 :* Simplify :**

But $A\bar{A} = 0$, $\bar{B}B = 0$ and $\bar{C}\bar{C} = \bar{C}$

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

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

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

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

$$\therefore \quad Y = (A+B+\bar{C})[AB + \bar{C} + \bar{A}\bar{B} + \bar{C} + \bar{C}]$$

But $\bar{C} + \bar{C} = \bar{C}$ $\quad \therefore \quad Y = (A+B+\bar{C})(AB + \bar{A}\bar{B} + \bar{C} + \bar{C}] = (A+B+\bar{C})(AB + \bar{A}\bar{B} + \bar{C})$

$$= AAB + A\bar{A}\bar{B} + A\bar{C} + ABB + \bar{A}\bar{B}B + B\bar{C} + AB\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{C}\bar{C}$$

But $AAB = AB$, $A\bar{A}\bar{B} = 0$, $ABB = AB$, $\bar{A}\bar{B}B = 0$ and $\bar{C}\bar{C} = \bar{C}$

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

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

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

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

$$= AB + A\bar{C} + B\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{C} \qquad \text{...since } (1+\bar{C}) = 1$$

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

$$\therefore \quad Y = AB + \bar{C} \qquad \text{...since } (A + B + \bar{A}\bar{B} + 1) = 1$$

$$\therefore \quad Y = AB + \bar{C} \qquad \text{...Ans.}$$

This is simplified expression.

5.4.2 Disadvantages of Algebraic Method of Simplification :

1. In the algebraic method of simplification, we need to write lengthy equations, find the common terms, and use various laws and theorems to minimize the expressions. So it is a time consuming work.
2. On top of that, sometimes we are not sure whether the further simplification is really possible or not. That means whether we have really obtained the minimized expression or not.

5.5 Karnaugh-Map Simplification (The Map Method) :

- This is another simplification technique to reduce the Boolean equation.
- It overcomes all the disadvantages of the algebraic simplification technique.

Scanned by CamScanner

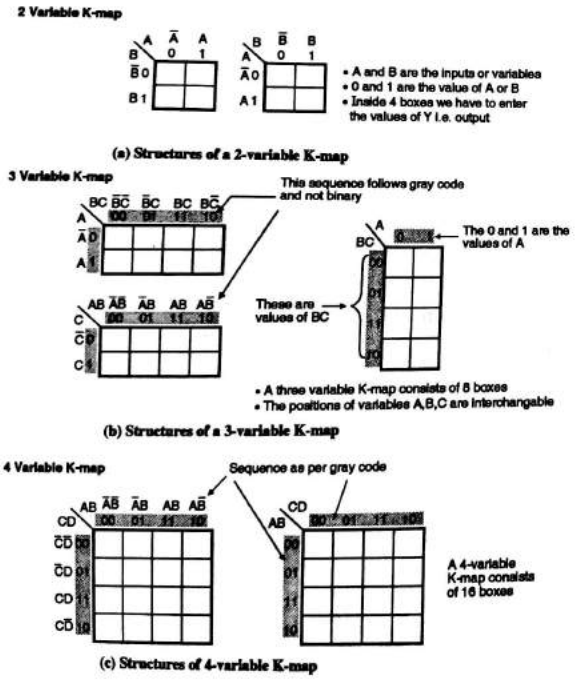
- K-map (short form of Karnaugh map) is a graphical method of simplifying a Boolean equation.
- K-map is a graphical chart made up of rectangular boxes.
- The information contained in a truth table or available in the SOP or POS form can be represented on a K-map.
- The K-map can be used for systematic simplification of Boolean expression.
- K-maps can be written for 2, 3, 4 ... upto 6 variables. Beyond that the K-map technique becomes very cumbersome.

K-map is ideally suitable for designing the combinational logic circuits using either a SOP method or a POS method.

- The K-map is drawn for output Y and the input variables (A, B, C... etc.) are used for making the entries in the boxes.

5.5.1 K-map Structure :

- The structure of a 2 input (variable) Karnaugh is shown in Fig. 5.5.1. This K-map is drawn for output Y of any two input combinational circuit such as a logic gate with inputs A and B. i.e. AB = 00, 01, 10 and 11.



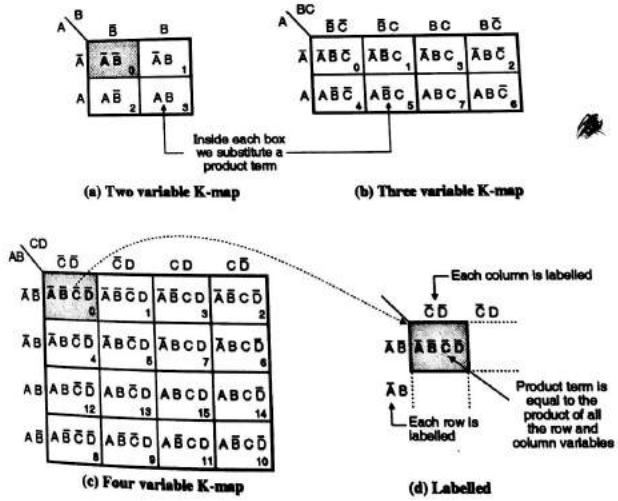
(C-217) **Fig. 5.5.1 : Structures of 2, 3, 4 variable K-map**

- A 2 variable K-map consists of $2^2 = 4$ rectangular boxes. Inside these boxes we have to enter the values of output Y for different combinations of inputs A and B.
- The K-map comprises a box for every line in the truth table. For a 2-input combinational circuit there are 4-lines in the truth table, so there will be 4-boxes in the 2 variable K-map.
- For a 3-variable K-map there will be 8 boxes, for 4-variable map there will be 16 boxes and so on.
- The 0's and 1's written on top or sides of the boxes represent the values of the corresponding variables.
- **The sequence is funny (It is gray code) :**

In truth tables, the values of inputs follow a standard binary sequence (00, 01, 10, 11). But in K-maps the input values are ordered in a different sequence (00, 01, 11, 10). This is as per the **gray code** and not binary code. So that as we move along a row or column only one variable will change its value at a time.

If the labeling is done as per the gray code, only then the elimination of variables and therefore the simplification will take place. When pairs, quads or octets are identified with the normal labeling, this elimination will not take place. Hence gray code is used in labeling the cells of a K-map.

5.5.2 K-map Boxes and Associated Product Terms :



(C-218) **Fig. 5.5.2 : K-map boxes and associated product terms**

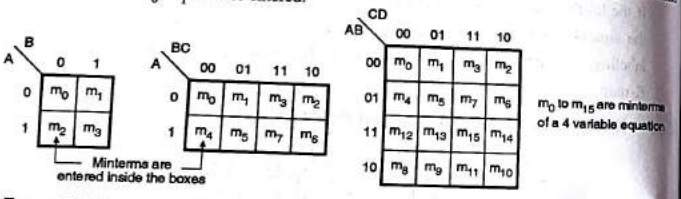
- The rectangular boxes in a K-map are to be filled with the values of output Y corresponding different combinations of inputs, as shown in Fig. 5.5.2.

Scanned by CamScanner

- Each row and column of a K-map is labelled with a variable, or a group of variables or their complements. For example, see the shaded box in Fig. 5.5.2(a).
- This box corresponds to the first row which is labelled by $\bar{A}$ and first column labelled by $\bar{B}$. Hence the product term written inside this box is $\bar{A}\,\bar{B}$ as shown in Fig. 5.5.2(a).
- Another example is the shaded box shown in Fig. 5.5.2(c). The first row is labelled with $\bar{A}\,\bar{B}$ and the first column with $\bar{C}\,\bar{D}$. Hence the product term written in this box is $\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$.
- Similarly the other product terms have been inserted, in the remaining boxes.

5.5.3 Alternative Way to Label the K-map :

- We can label the rows and columns of a K-map in a different way as shown in Fig. 5.5.3.
- Instead of labelling the rows and columns with the inputs and their complements (A, $\bar{A}$, $A\bar{B}$ etc.), their values in terms of 0s and 1s is used for labelling.
- And inside the boxes, instead of writing the actual product term, the corresponding shorthand minterm notations m_0, m_1 ..... are entered.



(a) Two variable K-map (b) Three variable K-map (c) Four variable K-map

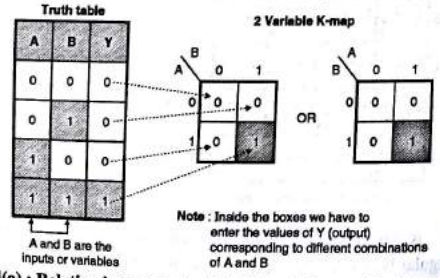
(C-219) **Fig. 5.5.3 : Alternative way to label K-map**

5.5.4 Truth Table to K-map :

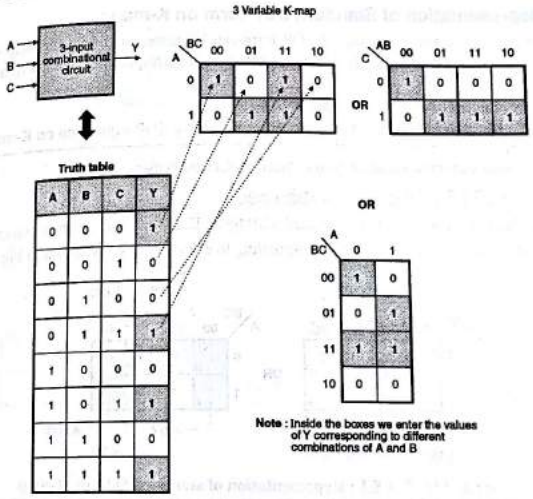
- Whenever the K-map is to be practically used for simplification, the entries inside the boxes are to be written by referring to the given truth table.
- In this section we are going to learn how to represent the given truth table on a Karnaugh map.

Relation between a truth table and K-map entries :

- Fig. 5.5.4(a) illustrates the mapping of a truth table on the K-map.



(C-220) **Fig. 5.5.4(a) : Relation between a truth table and K-maps for 2 variables**



(C-221) **Fig. 5.5.4(b) : Relation between a truth table and K-map for 3-variables**

- Referring to Fig. 5.5.4(a) we conclude that inside the boxes of the K-map we have to enter the values of output Y corresponding to various combinations of A and B.
- Fig. 5.5.4(b) shows the representation of a truth table using a 3-variable K-map and Fig. 5.5.4(c) shows the representation of a truth table using a 4-variable K-map.

Truth table

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

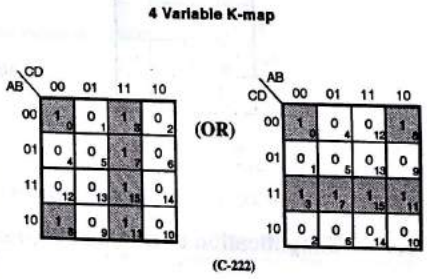


Fig. 5.5.4(c) : Relation between the truth table and 4-variable K-map

Scanned by CamScanner

5.5.5 Representation of Standard SOP form on K-map :

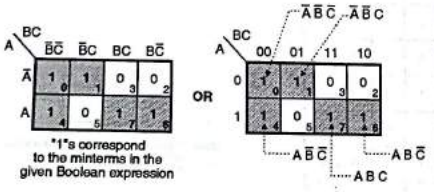
- The logical expression in standard SOP form can be represented with the help of a K-map by simply entering 1's in the cells (boxes) of the K-map corresponding to each minterm present in the equation.
- The remaining cells (boxes) are filled with zeros.
- Ex. 5.5.1 illustrates the concept of transferring a standard SOP expression on K-map.

Ex. 5.5.1 : Represent the equation given below on Karnaugh map.

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

Soln. : The given expression is in the standard SOP form. Each term represents a minterm.

We have to enter 1's in the boxes corresponding to each minterm, as shown in Fig. P. 5.5.1.

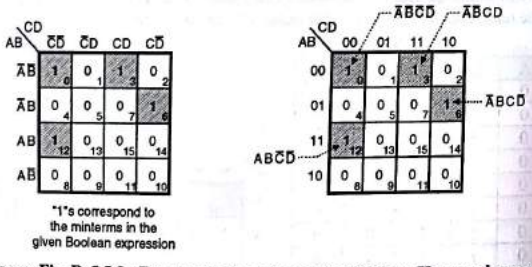


(C-223) **Fig. P. 5.5.1 : Representation of standard SOP on K-map**

Ex. 5.5.2 : Plot the following Boolean expression on K-map.

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

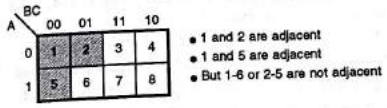
Soln. : Refer Fig. P. 5.5.2.



(C-224) **Fig. P. 5.5.2 : Representation of canonical SOP on Karnaugh map**

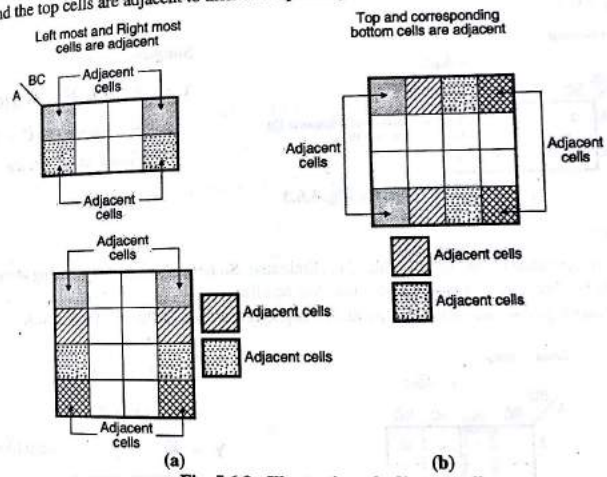
5.6 Simplification of Boolean Expressions using K-map :

- Simplification of Boolean expressions using K-map is based on combining or grouping the terms in the adjacent cells (or boxes) of a K-map.
- Two cells of a K-map are said to be adjacent if they differ in only one variable as shown in Fig. 5.6.1.



(C-225) **Fig. 5.6.1 : Adjacent or non-adjacent cells**

- Note the cells on left or right side or at the bottom and top of cells are adjacent cells. But the cells connected diagonally are not the adjacent ones.
- The left most cells are adjacent to their corresponding right most cells as shown in Fig. 5.6.2(a).
- And the top cells are adjacent to their corresponding bottom cells.



(C-226) **Fig. 5.6.2 : Illustration of adjacent cells**

5.6.1 How does Simplification Takes Place ?

- Once we transfer the logic function or truth table on a Karnaugh map, we have to use the grouping technique for simplifying the logic function.
- Grouping means combining the terms in the adjacent cells.
- The grouping of either adjacent 1's or adjacent 0's results in the simplification of Boolean expression in the SOP or POS forms respectively.
- If we group the adjacent 1's then the result of simplification is in SOP (sum of products) form.
- And if the adjacent 0's are grouped, then the result of simplification is in the POS (product of sums) form.

5.6.2 Way of Grouping (Pairs, Quads and Octets) :

- While grouping, we should group most number of 1's (or 0's).

Scanned by CamScanner

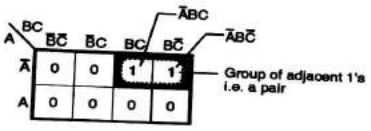
- The grouping follows the binary rule i.e. we can group 1, 2, 4, 8, 16, 32 ..... number of 1's or 0's. We cannot group 3, 5, 7, ... number of 1's or 0's.
  1. **Pairs :** A group of two adjacent 1's or 0's is called as a pair.
  2. **Quad :** A group of four adjacent 1's or 0's is called as a quad.
  3. **Octet :** A group of eight adjacent 1's or 0's is called as octet.

5.6.3 Grouping Two Adjacent One's (Pairs) :

If we group two adjacent 1's on a K-map, to form a pair, then the resulting term will have one less literal (variable) than the original term. That means by pairing two adjacent 1's we can eliminate one variable.

- The grouping of two adjacent 1's and elimination of one variable due to pairing is illustrated in Fig. 5.6.3.

Given K-map



Simplification :

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

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

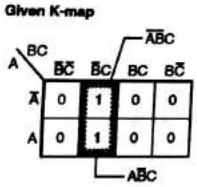
Thus C is eliminated

(C-228) **Fig. 5.6.3**

Conclusion :

- Due to formation of pair the variable C is eliminated. So henceforth just by looking at the pair we should be able to identify the variable that will be eliminated.
- The other types of pairs and corresponding simplifications are as shown in Fig. 5.6.4.

Given K-map



Simplification :

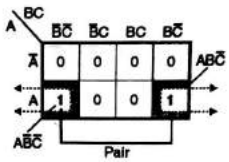
$Y = \bar{A}\bar{B}C + A\bar{B}C$

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

$Y = \bar{B}C$ ....since $(\bar{A} + A) = 1$

Thus A is eliminated.

Given K-map



Simplification :

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

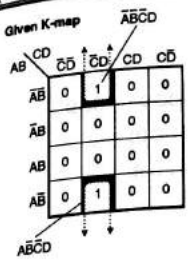
$= A\bar{C}(B + \bar{B})$

$Y = A\bar{C}$ ...since $(B + \bar{B}) = 1$

Conclusion : B is eliminated.

(C-228) **Fig. 5.6.4 : (Contd...)**

Given K-map



Simplification :

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

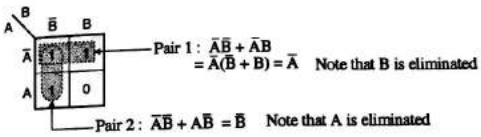
$= \bar{B}\bar{C}D(\bar{A} + A)$

$\therefore Y = \bar{B}\bar{C}D$

Conclusion : A is eliminated.

Example of overlap :

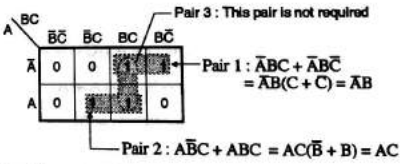
Given K-map :



$\therefore$ Final expression $Y = \bar{A} + \bar{B}$

Note that in order to cover all the 1's we have to overlap two pairs as shown.

Given K-map :



Add the two product terms to get,

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

(C-229) **Fig. 5.6.4 : Different types of pairs and the corresponding simplifications**

Note : In this K-map three pairs were possible to be formed. However only two pairs are sufficient to include all the 1's present in the K-map. Then the third pair is not required.

5.6.4 Grouping Four Adjacent Ones (Quad) :

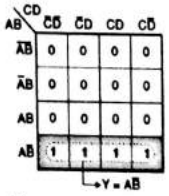
- If we group four 1's from the adjacent cells of a K-map, then the group is called as a **Quad.**
- In such a quad two variables associated with the minterms are same and the other two are not the same.
- After forming a Quad, the simplification takes place in such a way that the two variables which are not same will be eliminated. **Thus a quad eliminates two variables.**
- Fig. 5.6.5 shows various types of Quads and the corresponding mathematical simplification.

Scanned by CamScanner

- Note that overlapping is possible in Quads.

Given K-map :

Simplification : (C-230)

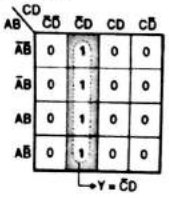


$$Y = A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}C\bar{D}$$
$$= A\bar{B}[\bar{C}\bar{D} + \bar{C}D + CD + C\bar{D}]$$
$$= A\bar{B}[\bar{C}(\bar{D}+D) + C(D+\bar{D})]$$
$$\therefore Y = A\bar{B}[\bar{C}+C] = A\bar{B}$$

Thus C and D are eliminated.

Given K-map :

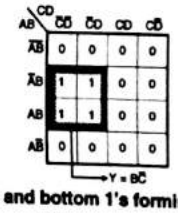
Simplification : (C-231)



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

Thus A and B are eliminated.

Four adjacent ones forming a square

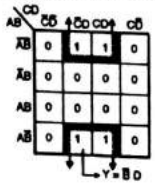


$$Y = \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}B\bar{C}D + AB\bar{C}D$$
$$= B\bar{C}\bar{D}(\bar{A}+A) + B\bar{C}D(\bar{A}+A)$$
$$= B\bar{C}\bar{D} + B\bar{C}D = B\bar{C}(\bar{D}+D)$$
$$\therefore Y = B\bar{C}$$

Thus A and D are eliminated

(C-231(a))

Top and bottom 1's forming a Quad

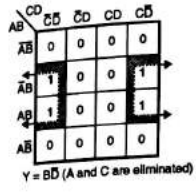


$$Y = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + A\bar{B}\bar{C}D + A\bar{B}CD$$
$$= \bar{A}\bar{B}D(\bar{C}+C) + A\bar{B}D(\bar{C}+C)$$
$$= \bar{A}\bar{B}D + A\bar{B}D$$
$$\therefore Y = \bar{B}D(\bar{A}+A) = \bar{B}D$$

Thus A and C are eliminated

(C-231(b)) **Fig. 5.6.5 (Contd...)**

Leftmost and rightmost 1's forming a Quad

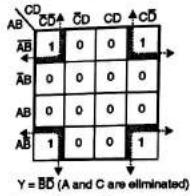


(C-231(c))

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

Thus A and C are eliminated

1's corresponding to corners forming a Quad

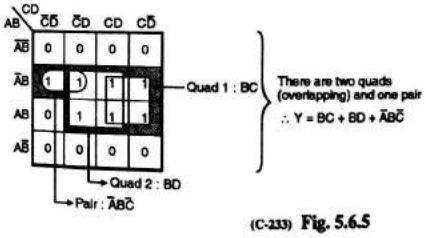


(C-232)

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

Thus A and C are eliminated

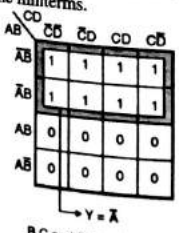
Overlapping of Quads and pairs



(C-233) **Fig. 5.6.5**

5.6.5 Grouping Eight Adjacent Ones (Octet) :

- It is possible to form a group of eight adjacent ones. Such a group is called as octet.
- When an octet is formed three variables will change and only one variable will remain same in all the minterms.



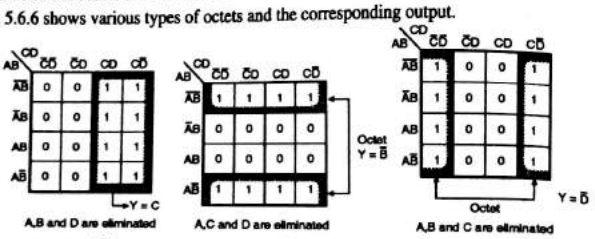
B,C and D are eliminated

(C-234) **Fig. 5.6.6(a)**

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D}$$
$$+ \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D}$$
$$\therefore Y = \bar{A}\bar{B}\bar{C}(\bar{D}+D) + \bar{A}\bar{B}C$$
$$(D+\bar{D}) + \bar{A}B\bar{C}(\bar{D}+D) + \bar{A}BC(D+\bar{D})$$
$$\therefore Y = \bar{A}\bar{B}(\bar{C}+C) + \bar{A}B(\bar{C}+C)$$
$$\therefore Y = \bar{A}(\bar{B}+B) = \bar{A}$$

Scanned by CamScanner

- The only variable that remains same is $\bar{A}$. So it appears as output.
- The three variables that change will be eliminated and the variable which does not change will appear as output.
- Thus octet eliminates three variables.
- Fig. 5.6.6 shows various types of octets and the corresponding output.



(C-234(a)) (b) (C-234(b)) (c) (C-235) (d)

Fig. 5.6.6

5.6.6 Summary of Rules Followed for K-Map Simplification :

Summary :

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 number of cells in each group.
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups possible.

5.7 Minimization of SOP Expressions (K Map Simplification) :

- The K-map can be used to simplify the logical expression to a level beyond which it cannot be further simplified.
- After such a simplification, it will require minimum number of gates with minimum number of inputs to the gates. Such an expression is called as a **minimized** expression.
- For minimizing the logical expression, follow the procedure given below.

Minimization procedure :

Step 1 : Prepare the K-map and place 1's according to the given truth table or logical expression. Fill the remaining cells by 0's.

Step 2 : Locate the isolated 1's i.e. the 1's which cannot be combined with any other 1. Encircle such 1's.

Step 3 : Identify the 1's which can be combined to form a pair in only one way and encircle them.

Step 4 : Identify the 1's which can form a quad in only one way and encircle them.

Step 5 : Identify the 1's which can form an octet in only one way and encircle them.

Step 6 : After identifying the pairs, quads and octets, check if any 1 is yet to be encircled. If yes then encircle them with each other or with the already encircled 1's (by means of overlapping).

- Note that the number of groups should be minimum.

- Also note that any 1 can be included any number of times without affecting the expression.
- Let us solve some examples on this to make the concept clear.

Ex. 5.7.1 : A logical expression in the standard SOP form is as follows :

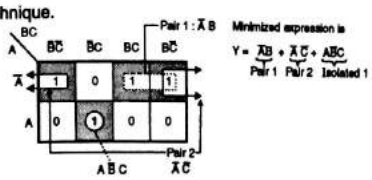
$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C$$

Minimize it using the K-map technique.

Soln. :

$$Y = \Sigma m(0, 2, 3, 5)$$

The required K-map is as shown in Fig. P. 5.7.1.



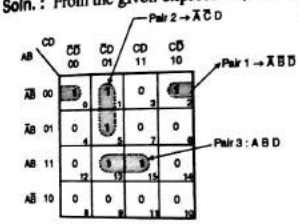
(C-250) **Fig. P. 5.7.1**

Ex. 5.7.2 : The logical expression representing a logic circuit is $Y = \Sigma m(0, 1, 2, 5, 13, 15)$. Draw the K-map and find the minimized logical expression.

Soln. : From the given expression, it is clear that the number of variables is 4.

$$Y = m_0 + m_1 + m_2 + m_5 + m_{13} + m_{15}$$

The required K-map is as shown in Fig. P. 5.7.2.



(C-251) **Fig. P. 5.7.2**

Minimized expression is

$$Y = \underbrace{\bar{A}\bar{B}\bar{D}}_{\text{Pair 1}} + \underbrace{\bar{A}\bar{C}D}_{\text{Pair 2}} + \underbrace{ABD}_{\text{Pair 3}}$$

Ex. 5.7.3 : For the logical expression given below draw the K-map and obtain the simplified logical expression.

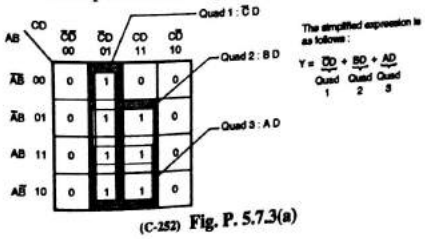
$$Y = \Sigma m(1, 5, 7, 9, 11, 13, 15).$$

Realize the minimized expression using the basic gates.

Soln. : The given expression is,

$$Y = m_1 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}$$

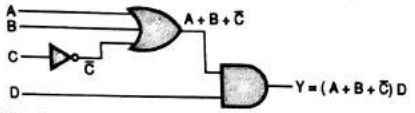
It can be expressed on K-map as shown in Fig. P. 5.7.3(a).



(C-252) **Fig. P. 5.7.3(a)**

Scanned by CamScanner

Realization : Equation (1) can be realized as shown in Fig. P. 5.7.3(b).



(C-253) **Fig. P. 5.7.3(b) : Realization with minimum number of gates**

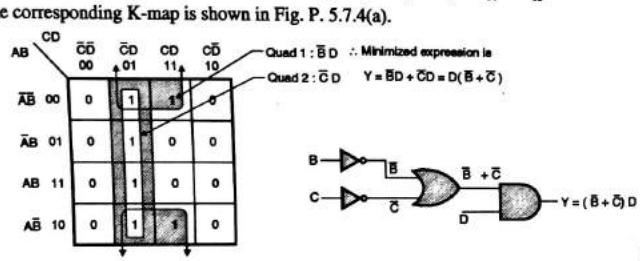
Ex. 5.7.4 : Minimize the following Boolean expression using K-map and realize it using the basic gates.

$$Y = \sum m(1, 3, 5, 9, 11, 13)$$

Soln. : The given expression can be expressed in terms of the minterms as,

$$Y = m_1 + m_3 + m_5 + m_9 + m_{11} + m_{13}$$

The corresponding K-map is shown in Fig. P. 5.7.4(a).



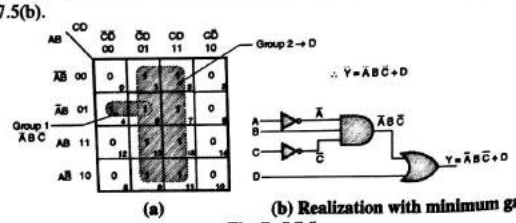
(a) (b) Realization

(C-254) **Fig. P. 5.7.4**

Ex. 5.7.5 : Using K-map realize the following expression using minimum number of gates.

$$Y = \sum m(1, 3, 4, 5, 7, 9, 11, 13, 15)$$

Soln. : The required K-map is shown in Fig. P. 5.7.5(a) and the realization using minimum gates is shown in Fig. P. 5.7.5(b).



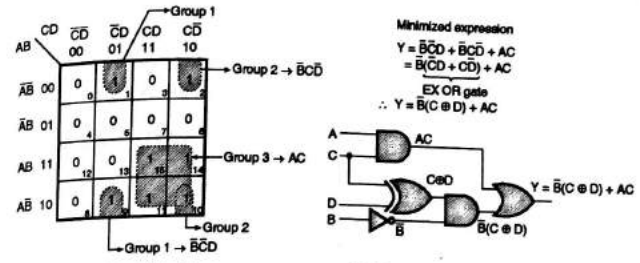
(a) (b) Realization with minimum gates

(C-255) **Fig. P. 5.7.5**

Ex. 5.7.6 : Minimize the following expression using K-map and realize using the basic gates.

$$Y = \sum m(1, 2, 9, 10, 11, 14, 15)$$

Soln. :



(a) K-map simplification (b) Realization with minimum number of gates

(C-256) **Fig. P. 5.7.6**

5.7.1 Elimination of a Redundant Group :

- If all the 1's in a group are already being used in some other groups, then that group is called as a redundant group.
- A redundant group has to be eliminated, because it increases the number of gates required.
- Effect of a redundant group and its elimination is illustrated in the Ex. 5.7.7.

Ex. 5.7.7 : Minimize the following expression using the K-map.

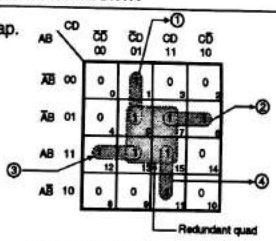
$$Y = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$$

Soln. :

1. The given expression can be expressed in the standard SOP form as follows :

   $Y = m_1 + m_5 + m_6 + m_7 + m_{11} + m_{12} + m_{13} + m_{15}$

   where $m_1, m_5 \ldots m_{15}$ are the minterms.
2. The required K-map is shown in Fig. P. 5.7.7(a).



(C-258) **Fig. P. 5.7.7(a)**

3. There are no isolated 1's. So encircle the separate pairs as shown in Fig. P. 5.7.7(a).
4. We would visualize a quad shown by dotted lines in Fig. P. 5.7.7(a). The question is should we encircle this quad ? The answer will be obtained by writing the expression of Y without and with this quad as follows :

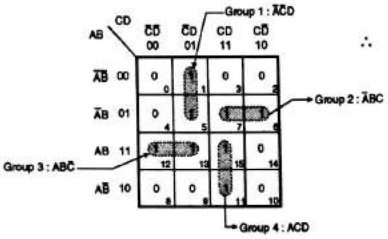
Expression without the quad : $Y = \bar{A}\bar{C}D + \bar{A}BC + AB\bar{C} + ACD$ (1) (2) (3) (4)

Expression with quad : $Y = \bar{A}\bar{C}D + \bar{A}BC + AB\bar{C} + ACD + BD$ (1) (2) (3) (4) → Redundant Quad

Conclusion :

- If we encircle the quad, then the expression for output consists of an additional term. So quad should not be encircled. It is called as the **redundant group.**
- Hence the correct K-map is shown in Fig. P. 5.7.7(b).

Scanned by CamScanner



$$\therefore\ Y = \bar{A}\bar{C}D + \bar{A}BC + AB\bar{C} + ACD$$
$$= \bar{A}\bar{C}D + ACD + \bar{A}BC + AB\bar{C}$$
$$= D(\underbrace{\bar{A}\bar{C} + AC}_{\text{EX-NOR}}) + B(\underbrace{\bar{A}C + A\bar{C}}_{\text{EX-OR}})$$

(C-259) **Fig. P. 5.7.7(b)**

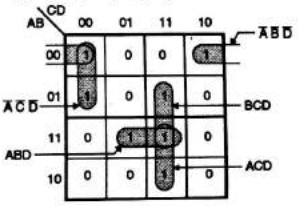
$$\therefore\ Y = D(\overline{A \oplus C}) + B(A \oplus C)$$

Ex. 5.7.8 : What is De Morgan's theorem ? Solve the following using minimization technique :

$z = f(A, B, C, D) = \Sigma(0, 2, 4, 7, 11, 13, 15)$

Soln. : $z = f(A, B, C, D) = \Sigma(0, 2, 4, 7, 11, 13, 15)$:

Simplification using K-map :



(C-1692) **Fig. P. 5.7.8**

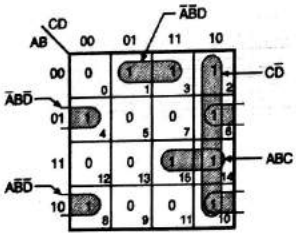
$$z = f(A, B, C, D) = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{C}\bar{D} + ABD + BCD + ACD$$

Ex. 5.7.9 : Simplify following function using K-map.

$F(A, B, C, D) = \Sigma m(1, 2, 3, 4, 6, 8, 10, 14, 15)$ May 14, 10 Marks

Soln. : $F(A, B, C, D) = \Sigma m(1, 2, 3, 4, 6, 8, 10, 14, 15)$

Simplification using K-map is as shown in Fig. P. 5.7.9.



(C-4967) **Fig. P. 5.7.9 : K-map**

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

5.7.2 Minimization of Logic Functions not Specified in Standard SOP Form :

- Till now we have seen how to use the K-map to minimize an expression which is given in standard SOP form.
- Now let us see the use of K-map for minimization when the given expression is not in the standard form. The procedure to be followed under such conditions is as follows :

The given expression $Y = \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{D} + A\bar{C} + \bar{B}$

Procedure :

Step 1 : Enter 1 for minterms (i.e. $\bar{A}\bar{B}CD$ and $\bar{A}BC\bar{D}$) present in the given expression.

Step 2 : Enter a pair of 1's for each term with one less variable than total (i.e. for $\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{B}\bar{D}$ because these terms have one less variable)

Step 3 : Enter four adjacent 1's for each term with two less variables than total (i.e. $A\bar{C}$).

Step 4 : Repeat for the other terms in a similar way.

Step 5 : Once the K-map is obtained, the minimization procedure is same as the one discussed earlier.

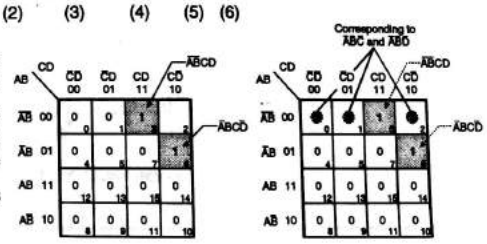
This procedure will be clear as you solve the example given below.

Ex. 5.7.10 : Minimize the logic equation given below :

$$Y = \underset{(1)}{\bar{A}\bar{B}CD} + \underset{(2)}{\bar{A}BC\bar{D}} + \underset{(3)}{\bar{A}\bar{B}\bar{C}} + \underset{(4)}{\bar{A}\bar{B}\bar{D}} + \underset{(5)}{A\bar{C}} + \underset{(6)}{\bar{B}}$$

Soln. :

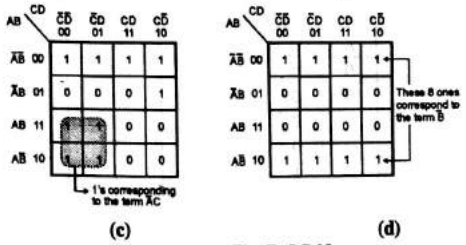
Step 1 : Enter 1 in the cell with A = 0, B = 0, C = 1, D = 1 for the first term $\bar{A}\bar{B}CD$ and in the cell with A = 0, B = 1, C = 1, D = 0 for the second term $\bar{A}BC\bar{D}$ as shown in Fig. P. 5.7.10(a).



(C-261) **Fig. P. 5.7.10**

Step 2 : Consider the third term $\bar{A}\bar{B}\bar{C}$. Enter 1 in two cells which correspond to A = 0, B = 0 and C = 0, as shown in Fig. P. 5.7.10(b). Now consider the fourth term $\bar{A}\bar{B}\bar{D}$ and enter 1 in two cells which correspond to A = 0, B = 0 and D = 0 as shown in Fig. P. 5.7.10(b).

Step 3 : Now consider the fifth term $A\bar{C}$. Enter 1 in four cells corresponding to A = 1 and C = 0, as shown in Fig. P. 5.7.10(c).

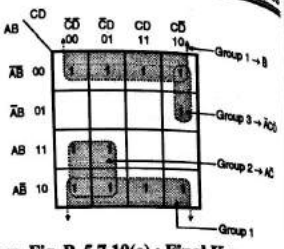


(C-262) **Fig. P. 5.7.10**

Scanned by CamScanner

Step 4 : Finally corresponding to the sixth term, enter eight 1's corresponding to B = 0 as shown in Fig. P. 5.7.10(d).

- The final K-map is as shown in Fig. P. 5.7.10(e). Use the normal simplification techniques discussed earlier to simplify this K-map and get the minimized expression.
- The simplified equation is $Y = \bar{B} + A\bar{C} + \bar{A}C\bar{D}$.

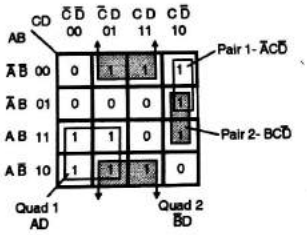


(C-263) **Fig. P. 5.7.10(e) : Final K-map**

Ex. 5.7.11 : What is Karnaugh map ? Give the structure of 4-variable K-map. Simplify the following Boolean equation using K-map.

$$Y = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}BC$$

Soln. : For K-map and its structure refer section 5.5. The K-map for given equation is as follows.



(C-3539) **Fig. P. 5.7.11**

$$\therefore \quad Y = AD + \bar{B}D + \bar{A}C\bar{D} + BC\bar{D}$$ ...Ans.

Ex. 5.7.12 : A digital system has 4-bit input from 0000 to 1111. Design a logic circuit that produces high output when input is less than 1000. Use K-map technique.

Soln. :

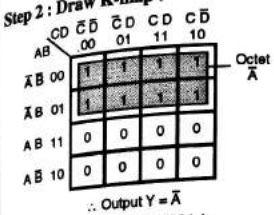
Step 1 : Write the truth table :

The truth table for the required circuit is as follows :

Table P. 5.7.12

Input A	B	C	D	Output Y	Input A	B	C	D	Output Y
0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	1	1	0	1	0	0
0	0	1	1	1	1	0	1	1	0
0	1	0	0	1	1	1	0	0	0
0	1	0	1	1	1	1	0	1	0
0	1	1	0	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0

Step 2 : Draw K-map :



(C-3541) **Fig. P. 5.7.12(a)**

Step 3 : Draw the logic circuit :

(C-3541) **Fig. P. 5.7.12(b) : Logic circuit**

5.7.3 Don't Care Conditions :

- For SOP form, we enter 1's corresponding to the combinations of input variables which produce a high output. And we enter 0's in the remaining cells of the K-map.
- For the POS form we enter 0's corresponding to the combinations of inputs which produce a low output and enter 1's in the remaining cells of the K-map.
- But it is not always true that the cells not containing 1's (in SOP) will contain 0's, because some combinations of input variable do not occur. Take the example of a 4 bit BCD counter. It will have valid outputs from 0000 to 1001 only.
- Also for some functions the outputs corresponding to certain combinations of input variables **do not matter.** That means for such input combinations it does not matter whether the value of output is 0 or 1.
- In such situation we are free to assume a 0 or 1 as output for each of such input combinations. These conditions are known as the **"Don't care conditions"** and in the K-map it is represented as × (cross) mark in the corresponding cell.

Ex. 5.7.13 : Simplify the expression given below using K-map. The don't care conditions are indicated by d ( ).

$Y = \Sigma m\,(1, 3, 7, 11, 15) + d\,(0, 2, 5)$.

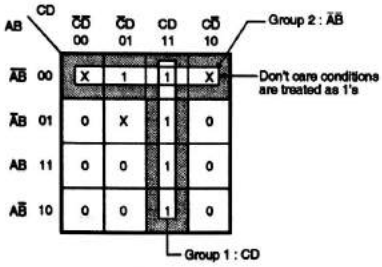
Soln. : The given equation is,

$$Y = \underbrace{m_1 + m_3 + m_7 + m_{11} + m_{15}}_{\text{Regular minterms so enter 1's}} + \underbrace{d\,(0, 2, 5)}_{\text{Don't care conditions so enter × marks}}$$

The required K-map is shown in Fig. P. 5.7.13.

Simplified expression :

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



(C-264) **Fig. P. 5.7.13**

Note : Every don't care mark need not be considered while grouping.

Scanned by CamScanner

Ex. 5.7.14 : Simplify the following expression once by considering the don't care conditions and once by ignoring the don't care condition.

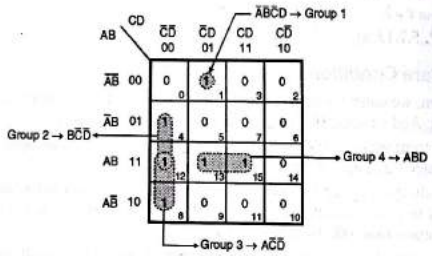
$Y = \Sigma m\,(1, 4, 8, 12, 13, 15) + d\,(3, 14)$

Soln. :

Part I : Simplification without don't care condition

The K-map by neglecting the don't care condition is shown in Fig. P. 5.7.14(a).

$$\therefore \text{ Simplified equation is } Y = \underset{(1)}{\bar{A}\bar{B}\bar{C}D} + \underset{(2)}{B\bar{C}\bar{D}} + \underset{(3)}{A\bar{C}\bar{D}} + \underset{(4)}{ABD}$$



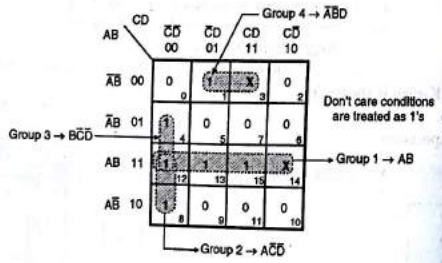
(C-265) **Fig. P. 5.7.14(a) : Simplification neglecting don't care condition**

Part II : Simplification with don't care conditions

The K-map including the don't care conditions is shown in Fig. P. 5.7.14(b).

$$\therefore \text{ Simplified equation is } Y = \underset{(1)}{AB} + \underset{(2)}{A\bar{C}\bar{D}} + \underset{(3)}{B\bar{C}\bar{D}} + \underset{(4)}{\bar{A}\bar{B}D}$$

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



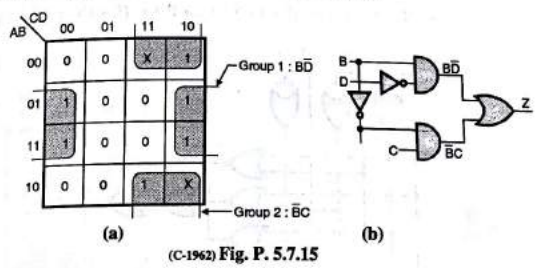
(C-266) **Fig. P. 5.7.14(b) : Simplification using don't care conditions**

Note : The don't care condition reduces the number of gates required for implementation.

Ex. 5.7.15 : Solve the following equation using corresponding minimization technique. Draw the diagram for the output :

$Z = f\,(A, B, C, D) = \Sigma m\,(2, 4, 6, 11, 12, 14) + d\,(3, 10).$

Soln. : The K-map for the given function is as shown in Fig. P. 5.7.15.



(C-1962) **Fig. P. 5.7.15**

$$\therefore Z = f\,(A, B, C, D) = B\bar{D} + \bar{B}C$$ **...Ans.**

Ex. 5.7.16 : Consider a chemical mixing tank for which there are 3 variable of interest : liquid level, pressure and temperature. The alarm will be triggered under the following conditions :

1. Low level with high pressure
2. High level with low pressure
3. High level with low temperature and high pressure.

Design the system, implement using only NOR gates. **May 09. 10 Marks**

Soln. :

Step 1 : Assign the variables :

Let L ⟶ Liquid level
P ⟶ Pressure
T ⟶ Temperature

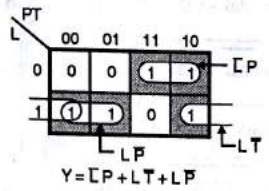
and Output ⟶ Y

Step 2 : Write the truth table :

Table P. 5.7.16

Inputs			Output
L	P	T	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 3 : Simplification using K-map :



$Y = \bar{L}P + L\bar{T} + L\bar{P}$

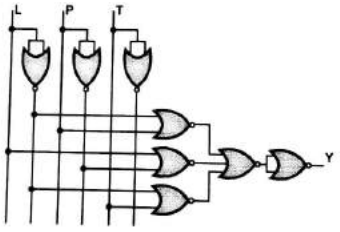
(C-3295) **Fig. P. 5.7.16(a)**

Scanned by CamScanner

Step 4 : Implementation using only NOR gates :

$$Y = L\bar{P} + \bar{L}P + L\bar{T} = \overline{\overline{L\bar{P} + \bar{L}P + L\bar{T}}} = \overline{\overline{L\bar{P}} \cdot \overline{\bar{L}P} \cdot \overline{L\bar{T}}}$$

$$= \overline{(\bar{L}+P)\cdot(L+\bar{P})\cdot(\bar{L}+T)} = \overline{\overline{(\bar{L}+P)} + \overline{(L+\bar{P})} + \overline{(\bar{L}+T)}}$$

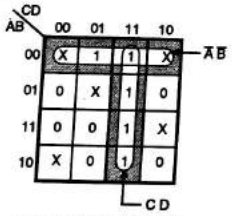


(C-2191) **Fig. P. 5.7.16(b)**

Ex. 5.7.17 : Simplify using K-map and realize using NOR gates.
f(A, B, C, D) = Σ m (1, 3, 7, 11, 15) + d(0, 2, 5, 8, 14). **May 09. 10 Marks**

Soln. : $f(A, B, C, D) = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5, 8, 14)$

Step 1 : Simplification using K-map :



(C-3257) **Fig. P. 5.7.17(a)**

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

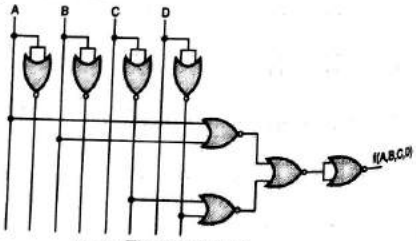
Step 2 : Realization using NOR gates :

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

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

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

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



(C-2192) **Fig. P. 5.7.17(b)**

Ex. 5.7.18 : Using K-map simplify :
F (P, Q, R, S) = Σm (2, 3, 6, 7, 8, 9, 10, 11) + d(12) and implement using minimum number of gates. **May 06. 10 Marks**

Soln. :

Step 1 : Simplify the given function using K-map :

$F(P, Q, R, S) = \Sigma m(2, 3, 6, 7, 8, 9, 10, 11) + d(12)$

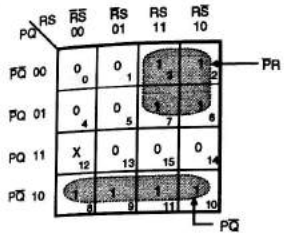
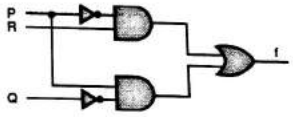


Fig. P. 5.7.18(a)

Step 2 : Implement the circuit using number of gates :

The implementation is as shown in Fig. 5.7.18(b).



(C-3739) **Fig. P. 5.7.18(b)**

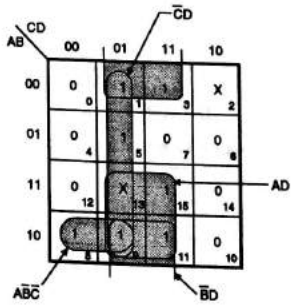
$$\therefore Y = P\bar{Q} + \bar{P}R$$

Ex. 5.7.19 : Minimize the following logic function and realize using NAND gates.
f (A, B, C, D) = Σm (1, 3, 5, 8, 9, 11, 15) + d (2, 13) **Dec. 10. 10 Marks**

Soln. :

$$f(A, B, C, D) = \Sigma m(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$$

Step 1 : Minimization using K-map :



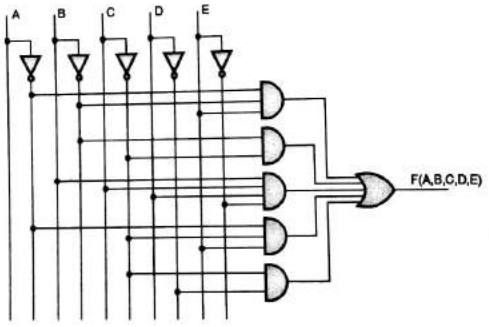
(C-2195) **Fig. 5.7.19(a)**

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

Scanned by CamScanner

Step 2 : Realization using NAND gates :

$$f(A, B, C, D) = \overline{\overline{A\bar{B}\bar{C} + \bar{C}D + AD + \bar{B}D}} = \overline{\overline{A\bar{B}\bar{C}} \cdot \overline{\bar{C}D} \cdot \overline{AD} \cdot \overline{\bar{B}D}}$$



(C-2196) **Fig. P. 5.7.19(b)**

5.7.4 Disadvantages of K-Map Technique :

1. It is a manual technique. We cannot use computers for K-map solution.
2. Minimization is extremely complicated if the number of variables is more than 6.
3. The simplification process depends heavily on the skills and abilities of the user.

5.8 NAND-NAND Implementation : MU : Dec. 02, May 10

University Questions

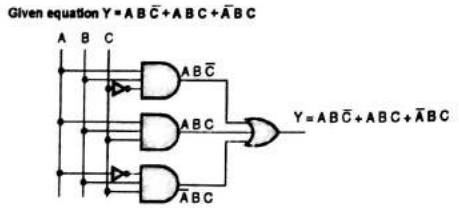
Q. 1 Prove that a two level AND-OR circuit can be replaced by NAND-NAND circuit and a two level OR-AND circuit can be replaced by a NOR-NOR circuit. **(Dec. 02, 3 Marks)**

Q. 2 Prove OR-AND configuration is equivalent to a NOR-NOR configuration. **(May 10, 3 Marks)**

- The SOP equations are implemented using the AND and OR gates. Refer to the SOP equation given below :

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

- And see its implementation using AND–OR–NOT gates as shown in Fig. 5.8.1.



(C-278) **Fig. 5.8.1 : Realization using AND–OR–NOT gates**

- From the implementation it is clear that the implementation procedure is as follows :

Step 1 : Implement all the product terms using AND gates.

Step 2 : Logically OR all the product terms to get the output.

However many a times it is convenient to implement the given circuit with only NAND or only NOR gates. Hence it is essential to convert the AND–OR–NOT circuit to a NAND–NAND or NOR–NOR circuit.

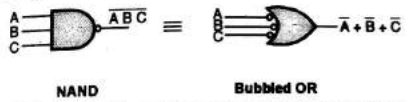
5.8.1 NAND - NAND Implementation using Alternative Representation :

Alternative Representation of a NAND gate

According to De-Morgan's first theorem

NAND ≡ Bubbled OR

This is as shown in Fig. 5.8.2. So a NAND gate can also be represented as a bubbled OR gate.



(C-279) **Fig. 5.8.2 : Alternative representation of a NAND gate**

This alternative representation is used for converting the AND–OR–NOT logic to NAND NAND logic.

Steps to be followed :

Step 1 : Simplify the given logical expression and convert it in the SOP form.

Step 2 : Draw the AND–OR–NOT realization.

Step 3 : Replace every AND gate by a NAND, every OR gate by a bubbled OR gate and NOT gate by a NAND inverter.

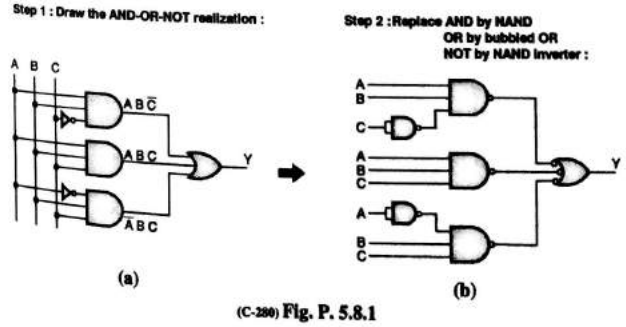
Step 4 : Draw the circuit using only NAND gates.

Go through Ex. 5.8.1 to understand this procedure.

Ex. 5.8.1 : Implement the following Boolean function using only NAND gates,

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

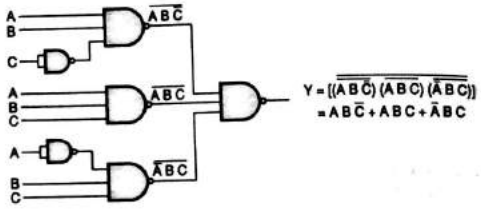
Soln. :



(C-280) **Fig. P. 5.8.1**

Scanned by CamScanner

Step 3 : Draw the circuit using only NAND gates :



(c)

(C-280) **Fig. P. 5.8.1 : Implementation using only NAND**

Ex. 5.8.2 : Implement the following expression using NAND - NAND logic, $Y = \Sigma m\,(0, 1, 5)$

Dec. 14, 5 Marks

Soln. :

Step 1 : Write the K-map :

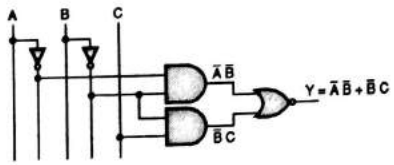
Obtain the simplified expression for Y.

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

(C-282) **Fig. P. 5.8.2(a) : K-map for given expression**

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

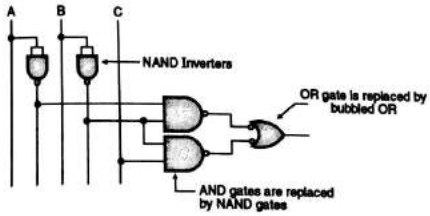
Step 2 : Implement using AND–OR–NOT logic :



(C-283) **Fig. P. 5.8.2(b) : AND – OR - NOT implementation**

Step 3 : Convert AND–OR–NOT into NAND–NAND logic :

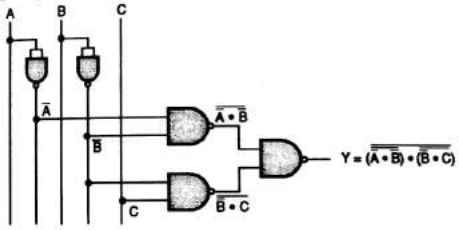
Replace every AND by NAND, every OR by a bubbled OR and every inverter by a NAND inverter to get the NAND - NAND logic shown in Fig. P. 5.8.2(c).



(C-284) **Fig. P. 5.8.2(c)**

Step 4 : Draw the circuit using only NAND gates :

The circuit using only NAND gates is shown in Fig. P. 5.8.2(d).



(C-285) **Fig. P. 5.8.2(d) : Implementation using NAND-NAND logic**

Ex. 5.8.3 : Given the logic expression :

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

1. Express in standard SOP
2. Draw the K-map for the equation
3. Minimize and realise using NAND gates only.

May 07, 12 Marks

Soln. :

1. **Standard SOP form :**

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

$$Y = ABCD + AB\bar{C}D + ABC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}\bar{C}D$$
$$+ A\bar{B}\bar{C}\bar{D} + ABCD + \bar{A}BCD + A\bar{B}CD + ABC\bar{D} + A\bar{B}C\bar{D} + \bar{A}\bar{B}CD$$
$$+ \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + ABCD + AB\bar{C}D + A\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}CD$$
$$+ A\bar{B}C\bar{D} + ABCD + ABC\bar{D}$$

$$Y = ABCD + AB\bar{C}D + ABC\bar{D} + AB\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + \bar{A}BCD$$
$$+ A\bar{B}C\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D}$$

Scanned by CamScanner

2. **K-map :**

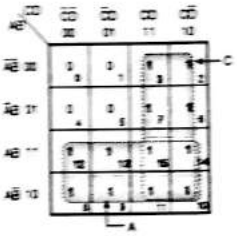
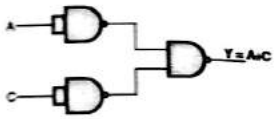


Fig. P. 5.8.3(a)



(C-274) Fig. P. 5.8.3(b)

$\therefore \quad Y = A + C$

3. **Implementation using NAND gates only :**

The minimized expression is

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

5.9 Product of Sum (POS) Simplification :

5.9.1 K-map Representation of POS Form :

- The POS form equations consist of maxterms.
- In the K-map corresponding to POS form equation we have to enter a 0 corresponding to each maxterm.
- The K-maps corresponding to the POS form are shown in Fig. 5.9.1.

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

OR

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

(C-286) **Fig. 5.9.1(a) : Two variable K-map for POS form**

A \ BC	00	01	11	10
0	A+B+C	A+B+$\bar{C}$	A+$\bar{B}$+$\bar{C}$	A+$\bar{B}$+C
1	$\bar{A}$+B+C	$\bar{A}$+B+$\bar{C}$	$\bar{A}$+$\bar{B}$+$\bar{C}$	$\bar{A}$+$\bar{B}$+C

OR

C \ AB	00	01	11	10
0	A+B+C	A+$\bar{B}$+C	$\bar{A}$+$\bar{B}$+C	$\bar{A}$+B+C
1	A+B+$\bar{C}$	A+$\bar{B}$+$\bar{C}$	$\bar{A}$+$\bar{B}$+$\bar{C}$	$\bar{A}$+B+$\bar{C}$

(C-286) **Fig. 5.9.1(b) Three variable K-map for POS form**

AB \ CD	00	01	11	10
00	A+B+C+D	A+B+C+$\bar{D}$	A+B+$\bar{C}$+$\bar{D}$	A+B+$\bar{C}$+D
01	A+$\bar{B}$+C+D	A+$\bar{B}$+C+$\bar{D}$	A+$\bar{B}$+$\bar{C}$+$\bar{D}$	A+$\bar{B}$+$\bar{C}$+D
11	$\bar{A}$+$\bar{B}$+C+D	$\bar{A}$+$\bar{B}$+C+$\bar{D}$	$\bar{A}$+$\bar{B}$+$\bar{C}$+$\bar{D}$	$\bar{A}$+$\bar{B}$+$\bar{C}$+D
10	$\bar{A}$+B+C+D	$\bar{A}$+B+C+$\bar{D}$	$\bar{A}$+B+$\bar{C}$+$\bar{D}$	$\bar{A}$+B+$\bar{C}$+D

(C-286) **Fig. 5.9.1(c) : Four variable K-map for POS form**

- The entries in the K-map can be shown in terms of the maxterms M_0, M_1 .... etc. as shown in Fig. 5.9.2.

A \ B	0	1
0	M_0	M_1
1	M_2	M_3

(a)

A \ BC	00	01	11	10
0	M_0	M_1	M_3	M_2
1	M_4	M_5	M_7	M_6

(b)

AB \ CD	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	M_5	M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	M_9	M_{11}	M_{10}

(c)

(C-287) **Fig. 5.9.2 : K-map in terms of maxterms**

5.9.2 Representation of Standard POS Form on K-map :

- Logical expressions in the standard POS form can be represented on K-map by entering 0's in the cells of K-map corresponding to each maxterm present in the given equation.
- The remaining cells are filled with 1's.
- This technique is illustrated in Ex. 5.9.1.

Ex. 5.9.1 : Represent the following standard POS expression on Karnaugh map.

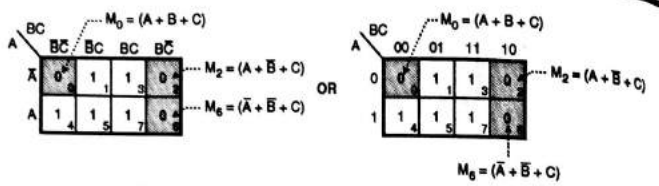
$$Y = (A + B + C)\,(A + \bar{B} + C)\,(\bar{A} + \bar{B} + C).$$

Soln. :

- Each term in the given logical equation is a maxterm.
- Enter a 0 corresponding to each maxterm as shown in Fig. P. 5.9.1.
- The given expression has three maxterms as follows,

  $(A + B + C) = M_0$, $(A + \bar{B} + C) = M_2$, $(\bar{A} + \bar{B} + C) = M_6$
- Hence we have to write the structure of 3 variable K-map as usual and enter 0's at M_0, M_2 and M_6 as shown in Fig. P. 5.9.1.

Scanned by CamScanner



(C-288) **Fig. P. 5.9.1 : Representation of standard POS on K-map**

Ex. 5.9.2 : Represent the following standard POS equation on the Karnaugh map.

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

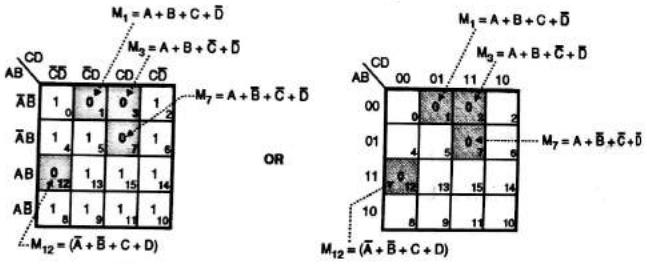
Soln. :

- The given expression contains four maxterms as follows :

$(A + B + C + \bar{D}) = M_1$ $\qquad$ $(A + \bar{B} + \bar{C} + \bar{D}) = M_7$

$(A + B + \bar{C} + \bar{D}) = M_3$ $\qquad$ $(\bar{A} + \bar{B} + C + D) = M_{12}$

- Enter a 0 corresponding to each maxterm as shown in Fig. P. 5.9.2.



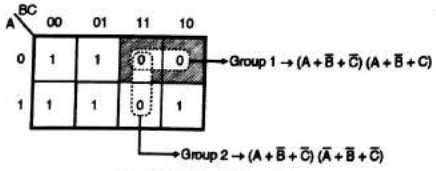
(C-289) **Fig. P. 5.9.2 : Representation of standard POS on K-map**

5.9.3 Simplification of Standard POS Form using K-map :

Minimization procedure :

1. The given POS expression consists of maxterms.
2. Corresponding to every maxterm enter a 0 in the K-map.
3. Enter 1's in the remaining cells of K-map.
4. Encircle/group 0's instead of 1's for carrying out the simplification.
5. Rules of simplification are exactly same as those used for the SOP form.

Ex. 5.9.3 : Find the expression in the POS form for the K-map shown in Fig. P. 5.9.3.



(C-290) **Fig. P. 5.9.3 : Given K-map**

Soln. :

Simplification : Group 1 → $(A + \bar{B} + \bar{C})\,(A + \bar{B} + C)$

$$= AA + A\bar{B} + AC + A\bar{B} + \bar{B}\,\bar{B} + \bar{B}C + A\bar{C} + \bar{B}\,\bar{C} + C\bar{C}$$

But $AA = A,\ \bar{B}\,\bar{B} = \bar{B},\ C\bar{C} = 0,\ A\bar{B} + A\bar{B} = A\bar{B}$

$$\therefore \text{ Group 1} = A + A\bar{B} + AC + \bar{B} + \bar{B}C + A\bar{C} + \bar{B}\,\bar{C}$$

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

$$= \underbrace{A + A}_{} + \bar{B} + \bar{B}\,\bar{C}$$

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

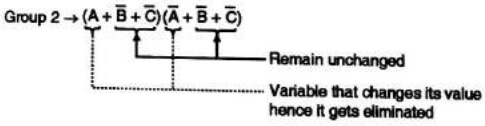
$$\text{Group 1} = A + \bar{B}$$

Conclusion : When a pair of 0s is formed, the variable which changes will get eliminated e.g. C changes for the pair of 0's in group 1. Hence it is eliminated.

∴ Group 2 → $\bar{B} + \bar{C}$

∴ Final minimized equation is as follows :

$$Y = \underset{(1)}{(A + \bar{B})}\ \underset{(2)}{(\bar{B} + \bar{C})}$$



(C-291)

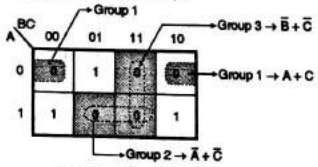
Ex. 5.9.4 : Minimize the following standard POS expression using K-map,

$Y = \Pi M\ (0, 2, 3, 5, 7)$

Soln. :

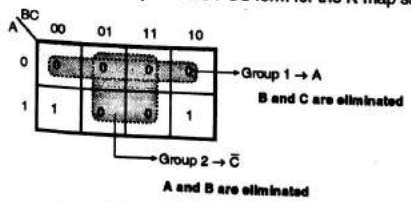
∴ Minimized expression is as follows :

$$Y = (A + C)\,(\bar{A} + \bar{C})\,(\bar{B} + \bar{C}) \qquad \textbf{...Ans.}$$



(C-292(a)) **Fig. P. 5.9.4**

Ex. 5.9.5 : Find the expression for the output in the POS form for the K-map shown in Fig. P. 5.9.5.



(C-293) **Fig. P. 5.9.5 : Given K-map**

Scanned by CamScanner

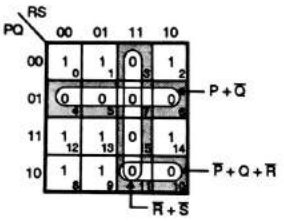
Soln. : Minimized expression is as follows :

$Y = A \cdot \overline{C}$ → Group 2, → Group 1 (C-293(a))

Ex. 5.9.6 : Simplify F(P, Q, R, S) = ΠM (3, 4, 5, 6, 7, 10, 11, 15) and implement using minimum number of gates. **Dec. 08, 10 Marks**

Soln. : $F(P, Q, R, S) = \Pi M(3, 4, 5, 6, 7, 10, 11, 15)$

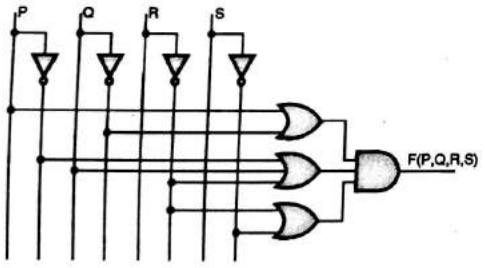
Step 1 : Simplification using K-map :



(C-3255) **Fig. P. 5.9.6(a)**

$$F(P, Q, R, S) = (P + \overline{Q})(\overline{P} + Q + \overline{R})(\overline{R} + \overline{S})$$

Step 2 : Implementaion using logic gates :



(C-2189) **Fig. P. 5.9.6(b)**

Ex. 5.9.7 : Simplify using K-map, obtain SOP equation and realize using only NAND gates. f (A, B, C, D) = ∏ M (1, 2, 3, 8, 9, 10, 11,14) + d(7, 15). **May 09, Dec. 11, Dec. 12, 10 Marks, Dec. 14, 10 Marks**

Soln. :

$$f(A, B, C, D) = \prod M(1, 2, 3, 8, 9, 10, 11, 14) + d(7, 15)$$

i.e. $$f(A, B, C, D) = \sum m(0, 4, 5, 6, 12, 13) + d(7, 15)$$

Step 1 : Simplification using K-map :

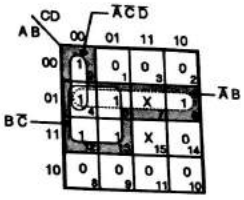
$$f(A, B, C, D) = \overline{A}B + B\overline{C} + \overline{A}\,\overline{C}\,\overline{D}$$

In SOP form :

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

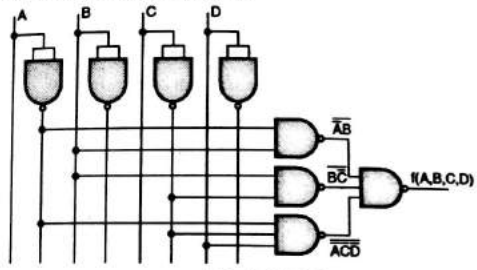
$$= \overline{\overline{\overline{A}B + B\overline{C} + \overline{A}\,\overline{C}\,\overline{D}}}$$

$$= \overline{\overline{\overline{A}B} \cdot \overline{B\overline{C}} \cdot \overline{\overline{A}\,\overline{C}\,\overline{D}}}$$



(C-3256) **Fig. P. 5.9.7(a)**

Step 2 : Implementation using NAND gates only :



(C-2190) **Fig. P. 5.9.7(b)**

Ex. 5.9.8 : Simplify F (L, M, N, O) = Π M (3, 4, 5, 6, 7, 10, 11, 15) and implement using minimum number of gates. **May 06, 10 Marks**

Soln. : $F(L, M, N, O) = \Pi M(3, 4, 5, 6, 7, 10, 11, 15)$

Step 1 : Simplify the function using K-map :

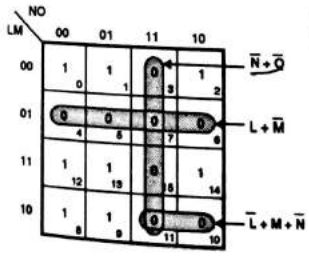
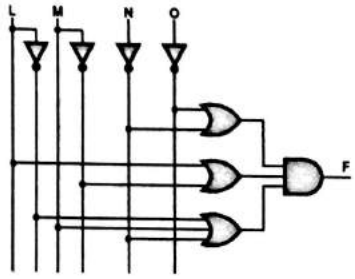


Fig. P. 5.9.8(a)

Step 2 : Implementation of function using gates :

The implementation is as shown in Fig. P. 5.9.8(b).



(C-3741) **Fig. P. 5.9.8(b)**

$$F(L, M, N, O) = (\overline{N} + \overline{O})(L + \overline{M})(\overline{L} + M + \overline{N})$$

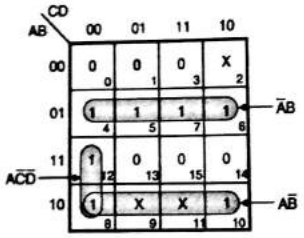
Scanned by CamScanner

Ex. 5.9.9 : Simplify the logic function using K-map :

$f(A, B, C, D) = \sum m(4, 5, 6, 7, 8, 10, 12) + d(2, 9, 11)$

Draw the logic diagram using NAND gates only. Dec. 13, 5 Marks

Soln. : Refer Fig. P. 5.9.9 for required K-map.



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

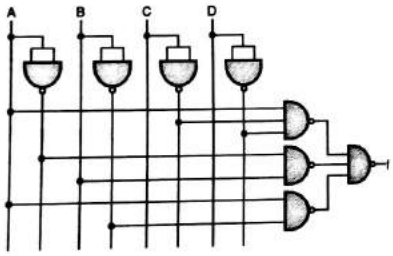
(C-4204) **Fig. P. 5.9.9**

Implementation using NAND gates is as shown in Fig. P. 5.9.9(a). Taking double inversion of RHS,

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

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

...(using De-Morgan's theorem)



(C-4205) **Fig. P. 5.9.9(a) : Implementation using NAND gates**

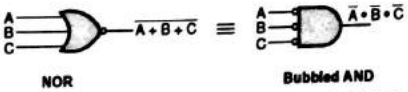
5.10 The NOR - NOR Implementation :

Alternative Representation of a NOR gate

- According to De-Morgan's second theorem

NOR ≡ Bubbled AND

- This is as shown in Fig. 5.10.1. So a NOR gate can also be represented as a bubbled AND gate.



(C-299) **Fig. 5.10.1 : Alternative representation of a NOR gate**

- This alternative representation is used for converting the circuit realized with AND-OR-NOT logic into a realization using the NOR-NOR logic.
- The steps to be followed for such a conversion are as follows :

Steps to be followed :

Step 1 : Simplify the given logical expression and convert it into Product of Sums (POS) form.

Step 2 : Draw the circuit using AND – OR – NOT gates.

Step 3 : Replace every OR gate by NOR, every AND gate by a bubbled AND gate and every inverter by a NOR inverter. Then replace every bubbled AND gate by a NOR gate.

Step 4 : Draw the final circuit using only the NOR gates.

Go through Ex. 5.10.1 to understand this procedure.

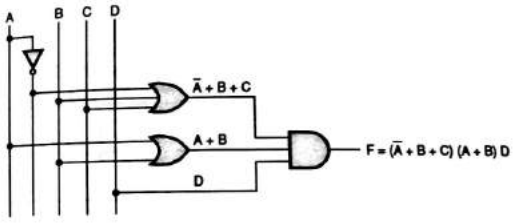
Ex. 5.10.1 : Implement the following Boolean function using only NOR gates.

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

Soln. :

Step 1 : Implement the given expression using AND–OR–NOT logic :

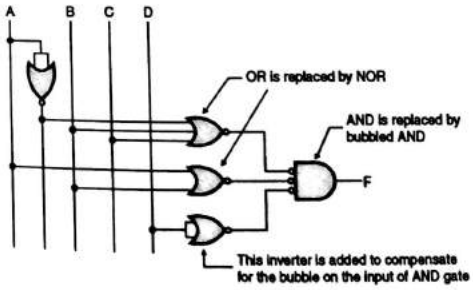
The implementation using AND–OR–NOT gates is shown in Fig. P. 5.10.1(a).



(C-300) **Fig. P. 5.10.1(a) : Implementation using AND–OR–NOT gates**

Step 2 : Replace OR by NOR, AND by bubbled AND and NOT by NOR inverter :

With these replacements, the circuit is as shown in Fig. P. 5.10.1(b).

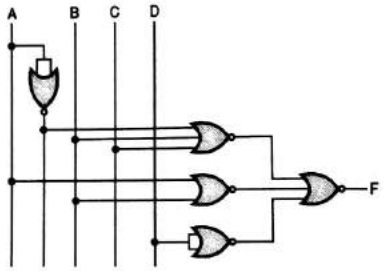


(C-301) **Fig. P. 5.10.1(b)**

Step 3 : Implement the circuit using only NOR gates :

The NOR - NOR implementation is shown in Fig. P. 5.10.1(c).

Scanned by CamScanner



(C-302) **Fig. P. 5.10.1(c) : Implementation using the NOR - NOR logic**

Ex. 5.10.2 : Implement the following POS expression using NOR - NOR logic.
$F = \Pi M (0, 1, 2, 4, 6)$

Soln. :

Step 1 : Simplify the given function using K-map :

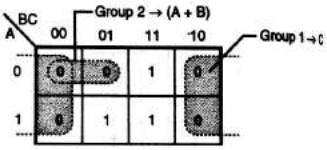
The given Boolean function is,

$$F = M_0 M_1 M_2 M_4 M_6$$

Hence the corresponding K-map is as shown in Fig. P. 5.10.2(a).

∴ Simplified Boolean equation is as follows :

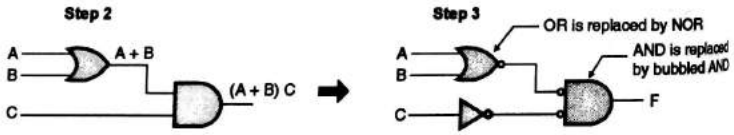
$$F = (A + B) C \quad ...(1)$$



(C-303) **Fig. P. 5.10.2(a)**

Step 2 : Implement this Boolean equation using AND-OR gates :

The implementation using AND–OR gates is shown in Fig. P. 5.10.2(b).



(b) Implementation using AND - OR logic **(c)**

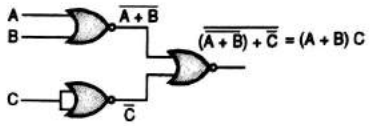
(C-304) **Fig. P. 5.10.2**

Step 3 : Replace OR by NOR, AND by bubbled AND and NOT by NOR inverter :

These replacements are shown in Fig. P. 5.10.2(c).

Step 4 : Implement the circuit using only NOR gates :

The NOR–NOR implementation is shown in Fig. P. 5.10.2(d).



(C-305) **Fig. P. 5.10.2(d) : Implementation using NOR-NOR logic**

Cross check :

Referring Equation (1) we get, $F = (A + B) C = \overline{\overline{(A + B) \cdot C}} = \overline{\overline{(A + B)} + \overline{C}}$ ...(2)

We get Equation (2) at the output of NOR–NOR logic shown in Fig. P. 5.10.2(d).

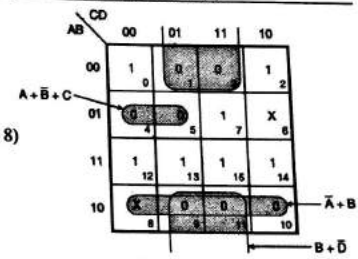
Ex. 5.10.3 : Simplify using K-map, obtain POS equation and realize using NOR gates :
$f(A, B, C, D) = \Pi M (1, 3, 4, 5, 9, 10, 11) \cdot d(6, 8)$ **Dec. 09. 10 Marks**

Soln. :

$$f (A, B, C, D) = \Pi M (1, 3, 4, 5, 9, 10, 11) \cdot d (6, 8)$$

Step 1 : Simplification using K-map :

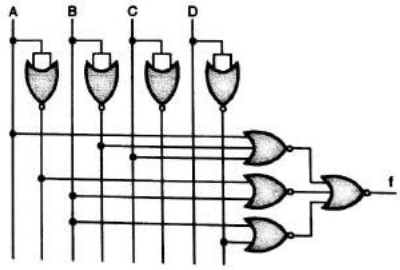
$$f (A, B, C, D) = (A + \overline{B} + C) (\overline{A} + B) (B + \overline{D})$$



(C-2192) **Fig. P. 5.10.3(a)**

Step 2 : Realization using NOR gates :

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



(C-2193) **Fig. P. 5.10.3(b)**

Ex. 5.10.4 : Given the logic expression : $A + \overline{B}\overline{C} + AB\overline{D} + ABCD$

1. Express it in standard SOP form.
2. Draw K-map and simplify.
3. Draw logic diagram using NOR gates only.
4. Draw logic diagram using NAND gates only.
5. Express it in standard POS form.

May 06. 14 Marks. May 15. 12 Mar

Soln. :

1. $Y = A + \overline{B}\,\overline{C} + AB\overline{D} + ABCD$:

Standard SOP form

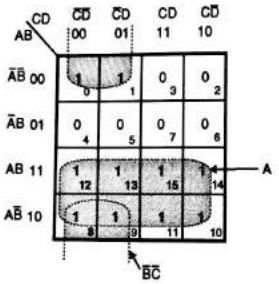
$$Y = A (B + \overline{B}) (C + \overline{C}) (D + \overline{D}) + \overline{B}\,\overline{C} (A + \overline{A}) (D + \overline{D}) + AB\overline{D} (C + \overline{C}) + ABCD$$

$$Y = ABCD + A\overline{B}CD + AB\overline{C}D + ABC\overline{D} + A\overline{B}\overline{C}D + AB\overline{C}\overline{D} + A\overline{B}C\overline{D} + A\overline{B}\overline{C}\overline{D}$$
$$+ A\overline{B}\overline{C}D + A\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D} + ABC\overline{D} + AB\overline{C}\overline{D} + ABCD$$

Scanned by CamScanner

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

2. **K-map :**



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

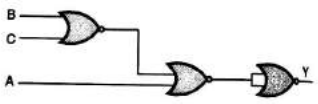
(C-3746) **Fig. P. 5.10.4(a)**

3. **Logic diagram using NOR gates only :**

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

...According to De-Morgan's second theorem

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



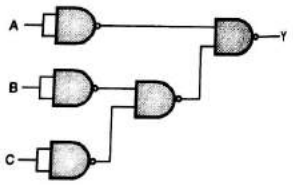
(C-3747) **Fig. P. 5.10.4(b)**

4. **Logic diagram using NAND gates only :**

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

According to De-Morgan's theorem

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



(C-3748) **Fig. P. 5.10.4(c)**

5. **Expression in standard POS form :**

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

Ex. 5.10.5 : Given logic equation :

$F = AB + AC + C + AD + ABC + ABC$

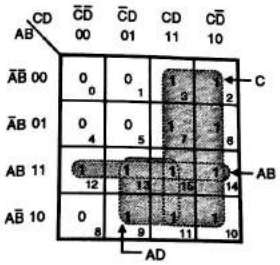
1. Design K – map for the given equation.
2. Express in SOP equation.
3. Minimise and realise the above equation using NOR gates only.
4. Realise the above equation in Standard POS and using NAND gates only.

Dec. 06. 16 Marks

$$F = AB + AC + C + AD + ABC + ABC$$

Soln. :

1. **K-map :**



(C-3742) **Fig. P. 5.10.5(a)**

$$\therefore F = AB + C + AD$$

This is the minimized expression.

2. **Standard SOP form :**

$$F = AB(C + \bar{C})(D + \bar{D}) + AC(B + \bar{B})(D + \bar{D}) + C(A + \bar{A})(B + \bar{B})(D + \bar{D})$$
$$+ AD(B + \bar{B})(C + \bar{C}) + ABC(D + \bar{D})$$
$$= ABCD + AB\bar{C}D + AB\bar{C}\bar{D} + ABC\bar{D} + ABCD + A\bar{B}CD + ABC\bar{D}$$
$$+ A\bar{B}C\bar{D} + ABCD + \bar{A}BCD + A\bar{B}CD + ABC\bar{D} + A\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}CD$$
$$+ ABCD + AB\bar{C}D + A\bar{B}CD + A\bar{B}\bar{C}D + ABCD + ABC\bar{D}$$
$$= ABCD + AB\bar{C}D + AB\bar{C}\bar{D} + ABC\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D}$$
$$+ \bar{A}\bar{B}CD + A\bar{B}\bar{C}D$$

3. **Implementation using NOR gates :**

$$F = AB + C + AD$$

Simplified equation for output is

$$F = \overline{\overline{AB + C + AD}}$$

$$= \overline{\overline{AB} \cdot \overline{C} \cdot \overline{AD}} \quad \text{...De-Morgan's theorem}$$

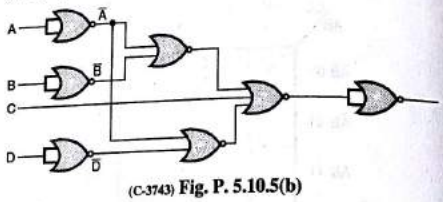
$$= \overline{(\bar{A} + \bar{B}) \cdot \bar{C} \cdot (\bar{A} + \bar{D})} \quad \text{...De-Morgan's theorem}$$

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

Taking double inversion of R. H. S.

Scanned by CamScanner

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



(C-3743) **Fig. P. 5.10.5(b)**

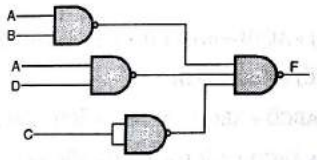
4. **Realization in standard POS form :**

$$F = AB + C + AD$$

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

Implementation using NAND gates :

$$F = AB + C + AD = \overline{\overline{AB+C+AD}} = \overline{\overline{AB}\cdot\overline{C}\cdot\overline{AD}}$$



(C-3744) **Fig. P. 5.10.5(c)**

Ex. 5.10.6 : Simplify f (P, Q, R, S) = ∏M (0, 2, 5, 7, 8, 13, 15) d (10) using K-map and realise using NOR gates only. **May 07. 8 Marks**

Soln. :

$$f(P, Q, R, S) = \prod M(0, 2, 5, 7, 8, 13, 15)\ d(10)$$

Simplification using K-map :

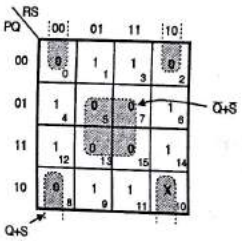
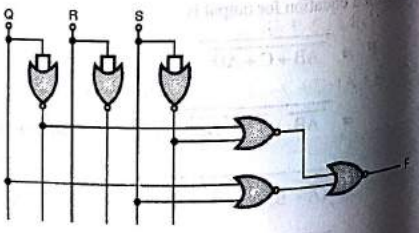


Fig. P. 5.10.6(a)

$$F = (\bar{Q}+\bar{S})(Q+S)$$

Implementation of expression using NOR gates only :



(C-3745) **Fig. P. 5.10.6(b)**

Ex. 5.10.7 : Given the logical expression :

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

1. Express in standard POS form
2. Draw K map for the equation
3. Minimize and realize using NOR gates only. **Dec. 07. 10 Marks**

Soln. :

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

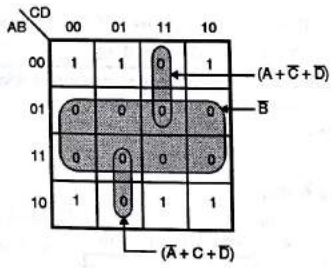
Standard POS form :

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

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

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

K-map :



(C-3749) **Fig. P. 5.10.7(a)**

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

Minimization and Realization using NOR gates only :

$$Y = \bar{B}(A+\bar{C}+\bar{D})(\bar{A}+C+\bar{D})$$
$$= \bar{B}(A\bar{A}+AC+A\bar{D}+\bar{C}\bar{A}+C\bar{C}+C\bar{D}+\bar{D}\bar{D}+\bar{D}\bar{A}+\bar{D}C)$$
$$= \bar{B}(0+AC+A\bar{D}+\bar{A}\bar{C}+0+C\bar{D}+\bar{D}+\bar{A}\bar{D}+C\bar{D}) \qquad [\because A\bar{A}=0]$$
$$= \bar{B}(AC+\bar{A}\bar{C}+\bar{D}(A+\bar{C}+\bar{A}+C+1))$$

Scanned by CamScanner

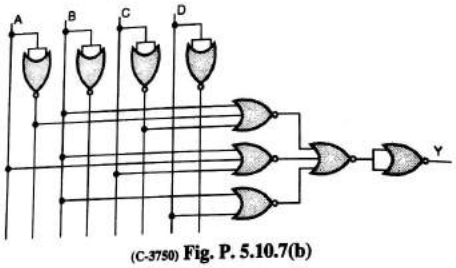
$$= \bar{B}(AC + \bar{A}\bar{C} + \bar{D}) = \bar{B}AC + \bar{B}\bar{A}\bar{C} + \bar{B}\bar{D} \quad [\because 1 + A + \bar{C} + \bar{A} + C = 1]$$

Implementation using NOR gates :

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

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

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



(C-3750) **Fig. P. 5.10.7(b)**

Ex. 5.10.8 : Express the equation in standard POS form : F (A, B, C) = $\sum$ m (0, 2, 5,7).

Dec. 15, 2 Marks

Soln. : F (A, B, C) = $\sum$ m (0, 2, 5, 7) ... (Given standard SOP form)

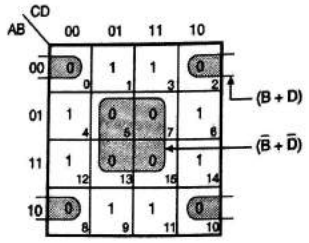
$\therefore$ Equation in standard POS form is πM (1, 3, 4, 6) ...Ans.

Ex. 5.10.9 : Simplify the following equation using K-map to obtain minimum POS equation and realize the minimum equation using only NOR gates.
F (A, B, C, D) = $\sum$ m (1, 3, 4, 6, 9, 11, 12, 14)

Dec. 15, 10 Marks

Soln. :

Step 1 : Simplification using K-map :



(C-5266) **Fig. P. 5.10.9(a) : K-map**

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

This is required expression in POS form.

Step 2 : Implementation using NOR gates :

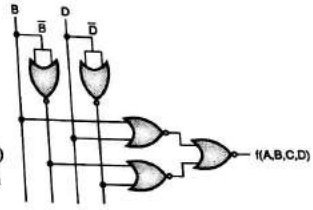
Taking double inversion of R.H.S.

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

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

...(Using De-Morgan's theorem)

Implementation using NOR gates is as shown in Fig. P. 5.10.9(b).



(C-5267) **Fig. P. 5.10.9(b) : Implementation using NOR gates**

Ex. 5.10.10 : Simplify the following equation using K-map to obtain minimum SOP equation and realize the minimum equation using two level NAND gates only.
F (A, B, C, D) = Π M (1, 3, 4, 6, 9, 11, 12, 14)

Dec. 16, 10 Marks

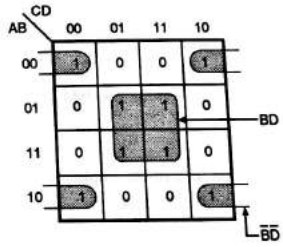
Soln. :

Step 1 : Convert given POS equation into standard SOP :

F (A, B, C, D) = Π M (1, 3, 4, 6, 9, 11, 12, 14) ...(Given standard POS form)

$\therefore$ Standard SOP equation is $\sum$m (0, 2, 5, 7, 8, 10, 13, 15)

Step 2 : Simplification using K-map :



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

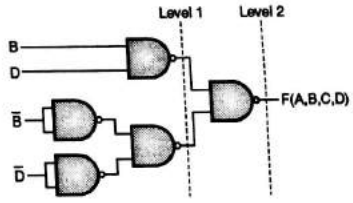
(C-5719) **Fig. P. 5.10.10(a) : K-map**

Step 3 : Realization using two level NAND gates :

Taking double inversion of R.H.S.,

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

$$= \overline{\overline{BD} \cdot \overline{\bar{B}\bar{D}}}$$



(C-5720) **Fig. P. 5.10.10(b) : Realization using two level NAND gates**

5.11 Quine Mc-Cluskey Minimization Technique (Tabular Method) :

- There are some combinational circuits with a large number of inputs. So the simplification using K-maps becomes too difficult. This happens when the number of variables is greater than 6.

Scanned by CamScanner

- So some other techniques for realizing the combinational circuits are used. Some of them are as follows :
  1. Use of programmable logic array 2. Programmable logic devices
  3. Quine Mc-Cluskey minimization technique

  Quine Mc-Cluskey technique is based on automatic or computer driven simplification routine. Therefore if we use Quine-Mc-Cluskey method then it is possible to use computer for simplifying the logic equations.
- A logic minimization technique should have the following characteristics :
  1. It should be capable of handling large number of variables.
  2. The technique should not be dependent on the abilities of the user otherwise the success of using that technique varies on the basis of user's skills.
  3. It should be compatible with computers so that we can use computers to obtain the solution.
  4. It should give us the minimized expression.
- The Quine Mc-Cluskey technique satisfies all these requirements. We can use this technique for designing various complex digital systems. Q.M. technique is also known as the **Tabular method.**

5.11.1 Important Definitions :

MU : Dec. 02, May 11

University Questions

Q. 1 Explain the term prime implicant. **(Dec. 02, 1 Mark)**

Q. 2 What is the essential prime implicant in Quine Mc-Cluskey method ? **(May 10, 2 Marks)**

Prime Implicant (PI) :

The prime implicant is a group of minterms which cannot be combined with any other minterm or groups.

Essential Prime Implicant (EPI) :

The essential prime implicant is a prime implicant in which one or more minterms are unique i.e. it contains at least one minterm which is not contained in any other prime implicant.

- The Quine Mc-Cluskey technique consists of two parts :
  1. First is to find all the prime implicants by an exhaustive search.
  2. To identify the essential prime implicants obtained in part 1 and select from them the remaining prime implicants which can give the perfectly minimized expression.
- The Quine Mc-Cluskey method can be better understood by solving the following examples.

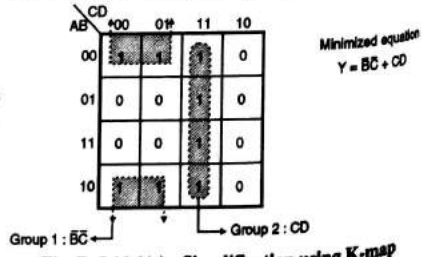
Ex. 5.11.1 : Simplify the following Boolean expression using K-map and verify it using the Quine-Mc Cluskey method. $Y(A, B, C, D) = \sum m(0, 1, 3, 7, 8, 9, 11, 15)$.

Soln. :

Step 1 :

Simplification using K-map

The K-map for the given Boolean function is shown in Fig. P. 5.11.1(a).



(C-306) **Fig. P. 5.11.1(a) : Simplification using K-map**

Simplification using Quine-Mc Cluskey Method

Step 2 : Arrange all the minterms according to number of 1's contained and form the groups having no ones, one 1, two 1's, three 1's and so on, as shown in Table P. 5.11.1(a).

Table P. 5.11.1(a)

Group	Minterm	Representation in binary form A	B	C	D		
1	m_0	0	0	0	0	✓	No 1's
2	m_1	0	0	0	1	✓	One 1
	m_8	1	0	0	0	✓	
3	m_3	0	0	1	1	✓	Two 1's
	m_9	1	0	0	1	✓	
4	m_7	0	1	1	1	✓	Three 1's
	m_{11}	1	0	1	1	✓	
5	m_{15}	1	1	1	1	✓	Four 1's

Step 3 :

- Now compare each minterm in group "n" with each minterm in group "(n + 1)", and identify the matched pairs. Put a check (✓) mark on each matched pair as shown in Table P. 5.11.1(a). A matched pair is a pair of minterms which differ only in one variable e.g. m_0 and m_4 is a matched pair because they differ only in the third position.
- After comparing each minterm in the nth group with every term in the $(n + 1)^{th}$ group, another table i.e. Table P. 5.11.1(b) is prepared. This table lists the results of all the matchings (all the matched terms) alongwith new terms generated as a result of matching of the corresponding terms.

Table P. 5.11.1(b) : Combination of minterms into groups of two

Group	Minterms (Matched pairs)	Binary Representation A	B	C	D	
0	$m_0 - m_4$	0	0	0	–	✓
	$m_0 - m_8$	–	0	0	0	✓
1	$m_1 - m_3$	0	0	–	1	✓
	$m_1 - m_9$	–	0	0	1	✓
	$m_8 - m_9$	1	0	0	–	✓
2	$m_3 - m_7$	0	–	1	1	✓
	$m_3 - m_{11}$	–	0	1	1	✓
	$m_9 - m_{11}$	1	0	–	1	✓
3	$m_7 - m_{15}$	–	1	1	1	✓
	$m_{11} - m_{15}$	1	–	1	1	✓

First entry →

Matched pairs of minterms obtained from Table P. 5.11.1(a)

New terms generated from the matched pairs of minterms

Scanned by CamScanner

- For example consider the first entry in Table P. 5.11.1(b). The matched pair is $m_0 - m_1$. Here A, B and C are equal and m_0 and m_1 differ in only "D". So the new term is written in front of $m_0 - m_1$ as 0 0 0 –.

	A	B	C	D
m_0 :	0	0	0	0
m_1 :	0	0	0	1

Only these bits are different

- Note that a (–) is put in place of D which indicates that "D" gets eliminated. Similarly the other entries are made.
- Note that pairs cannot be formed between minterms 8 and 3 or 9 and 7 in Table P. 5.11.1(a).

Step 4 :

- Now compare all the pairs of minterms in Table P. 5.11.1(b) with those in the adjacent groups, to find if we can form groups of four minterms.
- For example we compare pair $(m_0 - m_1)$ of group 0 with each pair in group 1. The pair $(m_0 - m_1)$ of group 0 is compared with each pair in group 1.
- The matching between two pairs of minterms in the adjacent groups is obtained if the "dashes" are in the same position and only one variable differs.
- Such matched pairs are tickmarked (✓) in Table P. 5.11.1(b). The matched pairs are also shown by dotted lines, in Table P. 5.11.1(b).
- For example, $(m_0 - m_1)$ of group 0 matches with $(m_8 - m_9)$ of group 1 or $(m_0 - m_8)$ of group 0 matches with $(m_1 - m_9)$ of group 1 and so on.
- Prepare a new table as shown in Table P. 5.11.1(c) which contains the pairs of pairs. For example $m_0 - m_1 - m_8 - m_9$ i.e. groups of four minterms. In front of each group of four minterms, the new terms are written with dashes (–). By comparing the minterm pairs of Table P. 5.11.1(b), bit by bit we put the dashes (–) where the bits do not match.

Table P. 5.11.1(c) : Combination of minterms into groups of four

Group	Minterms	Binary representation A	B	C	D	
0	$m_0 - m_1 - m_8 - m_9$	–	0	0	–	$\bar{B}\bar{C}$
	$m_0 - m_8 - m_1 - m_9$	–	0	0	–	
1	$m_1 - m_3 - m_9 - m_{11}$	–	0	–	1	$\bar{B}D$
	$m_1 - m_9 - m_3 - m_{11}$	–	0	–	1	
2	$m_3 - m_7 - m_{11} - m_{15}$	–	–	1	1	CD
	$m_3 - m_{11} - m_7 - m_{15}$	–	–	1	1	

Step 5 :

- Repeat the procedure of grouping. So see if we can group the "Quads" of minterms in the adjacent groups of Table P. 5.11.1(c) to obtain groups of eight minterms.
- There are no such matchings in Table P. 5.11.1(c). So the process of grouping will end here.

Step 6 :

- Collect all the nonchecked (non ✓ marked) terms from Tables P. 5.11.1(a), (b) and (c). These are the Prime Implicants (PI) and they will appear in the simplified Boolean expression of Y.
- There are no such terms in Tables P. 5.11.1(a) and (b). But there are three such terms in Table P. 5.11.1(c) as $\bar{B}\bar{C}$, $\bar{B}D$ and CD marked on that table.

$$\therefore \quad Y(A, B, C, D) = \bar{B}\bar{C} + \bar{B}D + CD \qquad ...(1)$$

Step 7 :

- Now prepare the prime-implicant table as shown in Table P. 5.11.1(d).

(C-308) **Table P. 5.11.1(d) : Prime implicant table**

PI	Decimal numbers corresponding to PI	Given Minterms 0	1	3	7	8	9	11	15
$\bar{B}\bar{C}$ ✓	0, 1, 8, 9	⊗	×			⊗	×		
$\bar{B}D$	1, 3, 9, 11		×	×			×	×	
CD ✓	3, 7, 11, 15			×	⊗			×	⊗

Obtained form Table P. 5.11.1(c)

- This table consists of the minterms contained in the given function, the PI terms included in Equation (1) and the decimal numbers of the minterms corresponding to these prime implicants.
- A (X) is put in each row under the minterms contained in a particular PI.
- Find the minterms that contain only one (X) in its column and encircle these cross marks as shown in Table P. 5.11.1(d) and put a tick mark (✓) infront of the corresponding PIs i.e $\bar{B}\bar{C}$ and CD.
- The encircled minterms 0 and 8 are contained in only one PI i.e. $\bar{B}\bar{C}$ whereas the minterms 7 and 15 are contained by only PI "CD".
- Hence the prime implicants $\bar{B}\bar{C}$ and CD are called as the **Essential Prime Implicants (EPI).**
- Now consider the other minterms (those which are not circle). The minterms 1 and 9 are contained in $\bar{B}\bar{C}$ (row 1 of Table P. 5.11.1(d)) and minterms 3 and 11 are contained in CD.
- Thus all the minterms in the given function are included in the two EPIs. So the minimized expression is given by,

$$Y(A, B, C, D) = \bar{B}\bar{C} + CD \qquad \text{...Ans.}$$

Note : The answer obtained by the Quine Mc Cluskey method is exactly same as that obtained by the K–map method.

Ex. 5.11.2 : Minimize the following logic function using K-map and verify the answer using the Quine-Mc Cluskey method.

$Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 5, 7, 8, 9, 11, 14)$.

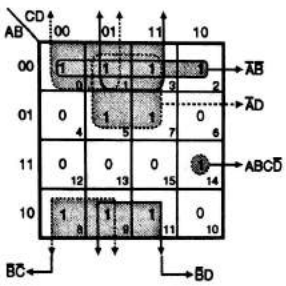
Soln. :

Part I : Minimization using K-map

The K-map for the given logic function is shown in Fig. P. 5.11.2.

$\therefore$ Simplified expression :

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



(C-309) **Fig. P. 5.11.2 : K-map**

Scanned by CamScanner

Part II : Quine – Mc Cluskey Method

Step 1 : Arrange all minterms according to the number of 1's :

The minterms are arranged according to the number of 1's and groups are formed as no ones, one 1, two 1s etc as shown in Table P. 5.11.2(a).

Table P. 5.11.2(a)

	Group	Minterm	Representation in binary form A	B	C	D	
No 1's	0	0	0	0	0	0	✓
One 1	1	1	0	0	0	1	✓
		2	0	0	1	0	✓
		8	1	0	0	0	✓
Two 1's	2	3	0	0	1	1	✓
		5	0	1	0	1	✓
		9	1	0	0	1	✓
Three 1's	3	7	0	1	1	1	✓
		11	1	0	1	1	✓
		14	1	1	1	0	→

Minterm 0 forms a pair with 1, 2 and 8

Minterm 14 does not appear in any pair so it is not ✓ marked

Step 2 : Combine the minterms into a group of two :

- Table P. 5.11.2(b) shows the matched pairs of minterms in the adjacent groups of Table P. 5.11.2(a) which differ at only one location (bit position) with respect to each other. Place (✓) mark on the matched pairs in Table P. 5.11.2(a).

Table P. 5.11.2(b) : Combination of minterms into groups of two

Group	Minterms (matched pairs)	Binary Representation A	B	C	D	
0	0 – 1	0	0	0	–	✓
	0 – 2	0	0	–	0	✓
	0 – 8	–	0	0	0	✓
1	1 – 3	0	0	–	1	✓
	1 – 5	0	–	0	1	✓
	1 – 9	–	0	0	1	✓
	2 – 3	0	0	1	–	✓
	8 – 9	1	0	0	–	✓
2	3 – 7	0	–	1	1	✓
	3 – 11	–	0	1	1	✓
	5 – 7	0	1	–	1	✓
	9 – 11	1	0	–	1	✓

Matched quads of minterms

New terms generated from the matched pairs of minterms

Matched pairs of minterms obtained from Table P. 5.11.2(a)

- The bit position where the minterms differ are represented by dashes (–) in the new terms written in front of matched pairs.

Step 3 : Combine the minterm pairs into groups of four (Quad) :

Group the minterm pairs of Table P. 5.11.2(b) (adjacent groups) to form minterm quads as shown in Table P. 5.11.2(c) and place a (✓) mark on the matched pairs in Table P. 5.11.2(b).

Table P. 5.11.2(c) : Combination of minterms into groups of four

Group	Minterm (Group of four)	Binary representation A	B	C	D
0	0 – 1 – 2 – 3	0	0	–	–
	0 – 1 – 8 – 9	–	0	0	–
	0 – 2 – 1 – 3	0	0	–	–
	0 – 8 – 1 – 9	–	0	0	–
1	1 – 5 – 3 – 7	0	–	–	1
	1 – 9 – 3 – 11	–	0	–	1
	1 – 3 – 5 – 7	0	–	–	1
	1 – 3 – 9 – 11	–	0	–	1

Prime implicants

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

$\bar{B}\,\bar{C}$

$\bar{A}\,D$

$\bar{B}\,D$

No further grouping is possible after this. So the process of grouping stops here.

Step 4 : Collect all nonchecked terms :

Collect all the nonchecked (non tickmarked) terms from Tables P. 5.11.2(a), (b) and (c), because they are the Prime-Implicants (PI).

In Table P. 5.11.2(a), the minterm 14 is nontickmarked so $A\,B\,C\,\bar{D}$ is a PI. In Table P. 5.11.2(b) we do not have any such term and in Table P. 5.11.2(c) we have four such terms i.e. $\bar{A}\,\bar{B}$, $\bar{B}\,\bar{C}$, $\bar{A}\,D$ and $\bar{B}\,D$.

$$\therefore Y(A, B, C, D) = A\,B\,C\bar{D} + \bar{A}\,\bar{B} + \bar{B}\,\bar{C} + \bar{A}\,D + \bar{B}\,D \qquad \ldots(1)$$

Step 5 : Prepare the PI table and obtain the EPIs :

The PI table is shown in Table P. 5.11.2(d).

- In the PI table find the columns containing only 1 cross (×) and encircle those (×) marks. Put (✓) mark in front of the corresponding PIs.
- These (✓) marked prime implicants in Table P. 5.11.2(d) are the Essential Prime Implicants (EPIs).
- Hence the simplified expression for Y is

$$\therefore Y(A, B, C, D) = A\,B\,C\bar{D} + \bar{A}\,\bar{B} + \bar{B}\,\bar{C} + \bar{A}\,D + \bar{B}\,D. \qquad \ldots\textbf{Ans.}$$

Note that this expression is same as the one obtained with K-map simplification.

Scanned by CamScanner

(C-310) **Table P. 5.11.2(d) : Prime Implicants (PI) table**

PI	Decimal number corresponding to PI	Given Minterms									
		0	1	2	3	5	7	8	9	11	14
$ABC\bar{D}$ ✓	14										⊗
$\bar{A}\bar{B}$ ✓	0, 1, 2, 3	×	×	⊗	×						
$\bar{B}\bar{C}$ ✓	0, 1, 8, 9	×	×					⊗	×		
$\bar{A}D$ ✓	1, 3, 5, 7		×		×	⊗	⊗				
$\bar{B}D$ ✓	1, 3, 9, 11		×		×				×	⊗	

Ex. 5.11.3 : Obtain the minimal expression using Quine-Mc Cluskey method.
$f(A, B, C, D) = \sum m(1, 5, 6, 12, 13, 14) + d(2, 4)$

Soln. :

Step 1 : Group the minterms according to number of 1s :

Table P. 5.11.3(a)

Group	Minterm	Binary representation A	B	C	D	
0	1	0	0	0	1	✓
	2*	0	0	1	0	✓
	4*	0	1	0	0	✓
1	5	0	1	0	1	✓
	6	0	1	1	0	✓
	12	1	1	0	0	✓
2	13	1	1	0	1	✓
	14	1	1	1	0	✓

* Represents the don't care terms

Step 2 : Group the minterms to form pairs :

Table P. 5.11.3(b)

Group	Minterm pair	Binary representation A	B	C	D		
0	1, 5	0	–	0	1	→ $\bar{A}\bar{C}D$	Prime implicants (non ✓ marked terms)
	2*, 6	0	–	1	0	→ $\bar{A}C\bar{D}$	
	4*, 5	0	1	0	–	✓	
	4*, 6	0	1	–	0	✓	
	4*, 12	–	1	0	0	✓	
1	5, 13	–	1	0	1	✓	
	6, 14	–	1	1	0	✓	
	12, 13	1	1	0	–	✓	
	12, 14	1	1	–	0	✓	

Step 3 : Group the minterms to form groups of four :

Table P. 5.11.3(c)

Group	Minterm Quads	Binary representation A	B	C	D	
0	4, 5, 12, 13	–	1	0	–	$B\bar{C}$
	4, 6, 12, 14	–	1	–	0	$B\bar{D}$
	4, 12, 5, 13	–	1	0	–	$B\bar{C}$
	4, 12, 6, 14	–	1	–	0	$B\bar{D}$

Prime implicants (non ✓ marked terms).

Step 4 : Group the prime implicants :

The non tick (✓) marked terms from Tables P. 5.11.3(b) and (c) are the Prime Implicants (PI). They are likely to appear in the simplified expression for output as follows :

$$f(A, B, C, D) = \bar{A}\bar{C}D + \bar{A}C\bar{D} + B\bar{C} + B\bar{D} \quad \text{...(1)}$$

Step 5 : Prepare the table of prime implicants :

(C-311) **Table P. 5.11.3(d) : Table of PI**

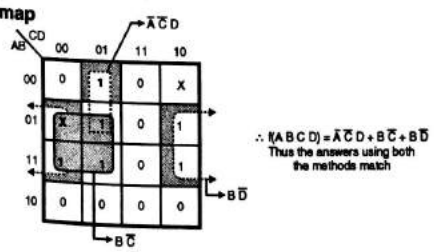
Prime implicants	Decimal numbers (minterms)	Minterms given 1	5	6	12	13	14
$\bar{A}\bar{C}D$ ✓	1, 5	⊗	X				
$\bar{A}C\bar{D}$	2, 6			X			
$B\bar{C}$ ✓	4, 5, 12, 13		X		X	⊗	
$B\bar{D}$ ✓	4, 6, 12, 14			X	X		⊗
		✓				✓	✓

← Don't care terms are not listed in this table

The encircled crosses (×) represent the Essential Prime Implicants (EPIs). They are $\bar{A}\bar{C}D$, $B\bar{C}$, $B\bar{D}$ and they cover the minterms 1, 5, 6, 12, 13 and 14 i.e. all the given minterms (Do not worry about 2 and 4 which are don't cares).

$$\therefore \quad f(A, B, C, D) = \bar{A}\bar{C}D + B\bar{C} + B\bar{D} \quad \text{...Ans.}$$

Crosscheck using K-map



(C-312) **Fig. P. 5.11.3**

Ex. 5.11.4 : Realize the following logic function in SOP form using Quine–Mc Cluskey method.
$f(A, B, C, D) = \Pi M(2, 7, 8, 9, 10, 12)$

Scanned by CamScanner

Soln. : Note that the given function is in the POS form or maxterm form and we want the solution in the SOP form. Hence we have to first convert the given expression into SOP form using the complementary relationship between the POS and SOP form.

$$f(A, B, C, D) = \Pi M(2, 7, 8, 9, 10, 12) = \sum m(0, 1, 3, 4, 5, 6, 11, 13, 14, 15).$$

Step 1 : Group the minterms according to number of 1s :

Table P. 5.11.4(a)

Group	Minterms	Binary Representation A	B	C	D	See Step 2	
0	0	0	0	0	0	✓	No ones
1	1	0	0	0	1	✓	One 1
	4	0	1	0	0	✓	
2	3	0	0	1	1	✓	Two 1s
	5	0	1	0	1	✓	
	6	0	1	1	0	✓	
3	11	1	0	1	1	✓	Three 1s
	13	1	1	0	1	✓	
	14	1	1	1	0	✓	
4	15	1	1	1	1	✓	Four 1s

Step 2 : Group the minterms to form pairs :

Table P. 5.11.4(b)

Group	Minterm pairs	Binary representation A	B	C	D	See Step 3	
0	0 – 1	0	0	0	–	✓	
	0 – 4	0	–	0	0	✓	
1	1 – 3	0	0	–	1		← $A\bar{B}D$
	1 – 5	0	–	0	1	✓	
	4 – 5	0	1	0	–	✓	
	4 – 6	0	1	–	0		← $\bar{A}B\bar{D}$
2	3 – 11	–	0	1	1		→ $\bar{B}CD$
	5 – 13	–	1	0	1		→ $B\bar{C}D$
	6 – 14	–	1	1	0		→ $BC\bar{D}$
3	11 – 15	1	–	1	1		→ ACD
	13 – 15	1	1	–	1		→ ABD
	14 – 15	1	1	1	–		→ ABC

Prime implicants (non ✓ marked terms)

Step 3 : Group the minterms to form quads (groups of 4) :

Table P. 5.11.4(c)

Group	Minterms quads	Binary reepresentation A	B	C	D	
0	0-4-1-5	0	–	0	–	$\bar{A}\bar{C}$ – Prime implicant
	0-1-4-5	0	–	0	–	

Step 4 : Collect all the Prime Implicants (PI) :

All the nontick marked terms in Tables P. 5.11.4 (b) and (c) are the Prime Implicants (PI) so they will appear in the expression for output, as follows :

$$f(A, B, C, D) = \bar{A}\bar{B}D + \bar{A}B\bar{D} + \bar{B}CD + B\bar{C}D + BC\bar{D} + ACD + ABD + ABC + \bar{A}\bar{C} \quad ...(1)$$

Step 5 : Prepare the PI table :

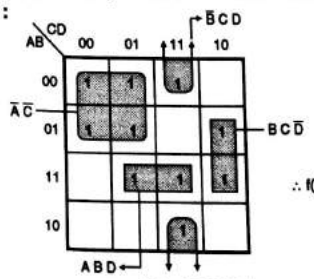
(C-313) **Table P. 5.11.4(d)**

P I term	Decimal equivalent	Given minterms 0	1	3	4	5	6	11	13	14	15
$\bar{A}\bar{B}D$	1, 3		X	X							
$\bar{A}B\bar{D}$	4, 6				X		X				
$\bar{B}CD$ ✓	3, 11			X				X			
$B\bar{C}D$	5, 13					X			X		
$BC\bar{D}$ ✓	6, 14						X			X	
ACD	11, 15							X			X
ABD ✓	13, 15								X		X
ABC	14, 15									X	X
$\bar{A}\bar{C}$ ✓	0, 1, 4, 5	⊗	X		X	X					

- The encircled term in Table P. 5.11.4(d) is called as the Essential Prime Implicant (EPI) which covers the minterms 0, 1, 4 and 5. This term will appear in the minimized expression.
- The remaining minterms are 3, 6, 11, 13, 14, 15 so find the terms which cover these minterms.
- From Table P. 5.11.4(d) such terms are $\bar{B}CD$ which covers the minterms 3 and 11, $BC\bar{D}$ which covers 6, 14 and ABD which covers 13 and 15. So these terms also will appear in the minimized expression.
- Hence the minimized expression in SOP form is given by

$$f(A, B, C, D) = \bar{A}\bar{C} + \bar{B}CD + BC\bar{D} + ABD \quad \text{...Ans.}$$

Cross check using K-map :



$\therefore f(A, B, C, D) = \bar{A}\bar{C} + \bar{B}CD + BC\bar{D} + ABD$

(C-314) **Fig. P. 5.11.4**

Scanned by CamScanner

Thus the results obtained by the two methods are exactly identical.

Ex. 5.11.5 : Simplify the function using Quine McCluskey method.

$f(A, B, C, D) = \sum m(4, 5, 8, 9, 11, 12, 13, 15)$. Draw the logical diagram using NAND gates. **Dec. 08, May 11, Dec. 12, 10 Marks**

Soln. :

$$f(A, B, C, D) = \sum m(4, 5, 8, 9, 11, 12, 13, 15)$$

Step 1 : Group the minterms according to number of 1's :

Group	Minterm	Binary representation A	B	C	D	
0	4	0	1	0	0	✓
	8	1	0	0	0	✓
1	5	0	1	0	1	✓
	9	1	0	0	1	✓
	12	1	1	0	0	✓
2	11	1	0	1	1	✓
	13	1	1	0	1	✓
3	15	1	1	1	1	✓

Step 2 : Group the minterm to form pairs :

Group	Minterm pairs	Binary representation A	B	C	D	
0	4 – 5	0	1	0	–	✓
	4 – 12	–	1	0	0	✓
	8 – 9	1	0	0	–	✓
	8 – 12	1	–	0	0	✓
1	5 – 13	–	1	0	1	✓
	9 – 11	1	0	–	1	✓
	9 – 13	1	–	0	1	✓
	12 – 13	1	1	0	–	✓
2	11 – 15	1	–	1	1	✓
	13 – 15	1	1	–	1	✓

Step 3 : Group the minterm to form groups of four :

Group	Minterm quad	Binary Representation A	B	C	D	Prime Implicants
0	4, 5, 12, 13	–	1	0	–	$B\bar{C}$
	4, 12, 5, 13	–	1	0	–	
	8, 9, 12, 13	1	–	0	–	$A\bar{C}$
	8, 12, 9, 13	1	–	0	–	
1	9,11, 13, 15	1	–	–	1	AD
	9, 13, 11, 15	1	–	–	1	

Step 4 : Prepare the table of prime implicants

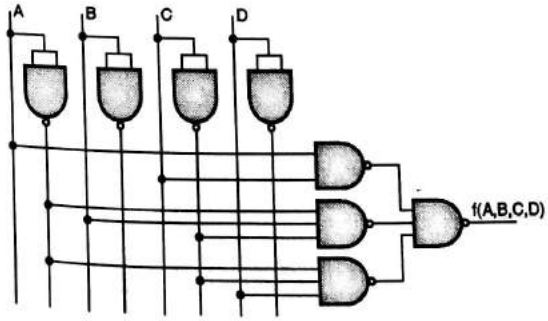
Prime implicant	Decimal number (minterm)	Minterms given 4	5	8	9	11	12	13	15
$B\bar{C}$	4, 5, 12, 13	⊗	⊗				×	×	
$A\bar{C}$	8, 9, 12, 13			⊗	×		×	×	
AD	9, 11, 13, 15				×	⊗		×	⊗

The encircled crosses represent the EPIs. They cover minterms 4, 5, 8, 9, 11, 12, 13, 15.

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

Step 5 : Implementation using NAND gates :

$$f(A, B, C, D) = \overline{\overline{B\bar{C} + A\bar{C} + AD}} = \overline{\overline{B\bar{C}} \cdot \overline{A\bar{C}} \cdot \overline{AD}}$$



(C-2188) **Fig. P. 5.11.5**

Scanned by CamScanner

Ex. 5.11.6 : Simplify using Quine McCluskey method.
$f(A, B, C, D) = \prod M(0, 2, 3, 6, 7, 8, 9, 12, 13)$.
Realize the equation using any universal gate. **May 09, Dec. 10, 10 Marks**

Soln. :

$$f(A, B, C, D) = \prod M(0, 2, 3, 6, 7, 8, 9, 12, 13)$$

$$\therefore \quad f(A, B, C, D) = \sum m(1, 4, 5, 10, 11, 14, 15)$$

Step 1 : Group the minterms according to number of 1's :

Group	Minterm	Binary representation A	B	C	D	
0	1	0	0	0	1	✓
	4	0	1	0	0	✓
1	5	0	1	0	1	✓
	10	1	0	1	0	✓
2	11	1	0	1	1	✓
	14	1	1	1	0	✓
3	15	1	1	1	1	✓

Step 2 : Group the minterms to form pairs :

Group	Minterm pair	Binary representation A	B	C	D	Prime Implicants
0	1 – 5	0	–	0	1	$\bar{A}\bar{C}D$
	4 – 5	0	1	0	–	$\bar{A}B\bar{C}$
1	10 – 11	1	0	1	–	✓
	10 – 14	1	–	1	0	✓
2	11 – 15	1	–	1	1	✓
	14 – 15	1	1	1	–	✓

Step 3 : Group the minterms to form quad :

Group	Minterm Quad	Binary representation A	B	C	D	Prime Implicant
0	10 – 11 – 14 – 15	1	–	1	–	AC
	10 – 14 – 11 – 15	1	–	1	–	

Step 4 : Prepare the table of prime implicants :

Prime Implicant	Decimal number (minterms)	Minterms given 1	4	5	10	11	14	15
$\bar{A}\bar{C}D$	1,5	(X)		X				
$\bar{A}B\bar{C}$	4,5		(X)	X				
AC	10,11,14,15				(X)	(X)	(X)	(X)

(C-3288)

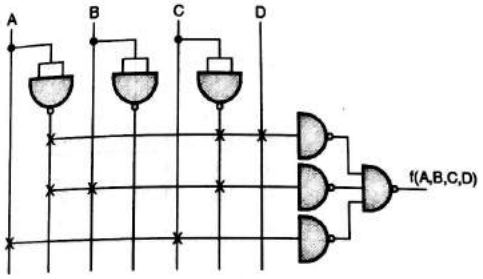
The encircled terms are EPI's (essential prime implicants)

It includes minterms 1, 4, 5, 10, 11, 14, 15.

$$\therefore \quad f(A, B, C, D) = \bar{A}\bar{C}D + \bar{A}B\bar{C} + AC \quad \text{...Ans.}$$

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

$$= \overline{\overline{(\bar{A}\bar{C}D)} \cdot \overline{(\bar{A}B\bar{C})} \cdot \overline{AC}}$$



(C-3647) **Fig. P. 5.11.6**

Ex. 5.11.7 : Using the Quine Mc-Cluskey method simplify
$F = \sum m(1, 3, 7, 9, 11, 13, 15) + d(2, 4)$ **May 10, 10 Marks**

Soln. :

$$F = \sum m(1, 3, 7, 9, 11, 13, 15) + d(2, 4)$$

Step 1 : Group the minterms according to number of 1's :

Group	Minterm	Binary representation A	B	C	D	Prime implicant
0	1	0	0	0	1	✓
	2	0	0	1	0	✓
	4	0	1	0	0	$\bar{A}B\bar{C}\bar{D}$
1	3	0	0	1	1	✓
	9	1	0	0	1	✓
2	7	0	1	1	1	✓
	11	1	0	1	1	✓
	13	1	1	0	1	✓
3	15	1	1	1	1	✓

Scanned by CamScanner

Step 2 : Group the minterms to form pairs :

Group	Minterm pair	Binary representation A	B	C	D	Prime implicant
0	1-3	0	0	–	1	✓
	1-9	–	0	0	1	✓
	2-3	0	0	1	–	$\bar{A}\bar{B}C$
1	3-7	0	–	1	1	✓
	3-11	–	0	1	1	✓
	9-11	1	0	–	1	✓
	9-13	1	–	0	1	✓
2	7-15	–	1	1	1	✓
	11-15	1	–	1	1	✓
	13-15	1	1	–	1	✓

Step 3 : Group the minterm to form quad :

Group	Minterm quads	Binary representation A	B	C	D	Prime implicant
0	1-3-9-11	–	0	–	1	$\bar{B}D$
	1-9-3-11	–	0	–	1	
1	3-7-11-15	–	–	1	1	CD
	3-11-7-15	–	–	1	1	
	9-13-11-15	1	–	–	1	AD
	9-11-13-15	1	–	–	1	

Step 4 : Prepare the table of prime implicants : (C-3710)

Prime implicants	Decimal numbers (minterms)	Minterms given 1	3	7	9	11	13	15
$\bar{A}B\bar{C}\bar{D}$	4							
$\bar{A}\bar{B}\bar{C}$	2, 3		X					
$\bar{B}D$	1, 3, 9, 11	Ⓧ	X		X	X		
CD	3, 7, 11, 15		X	Ⓧ		X		X
AD	9, 11, 13, 15				X	X	Ⓧ	X

The encircled crosses represent the Essential Prime Implicants (EPIs). They are $\bar{B}D$, CD, AD and they cover the minterms 1, 3, 7, 9, 11, 13, 15.

$$\therefore \quad f(A, B, C, D) = \bar{B}D + CD + AD$$

Ex. 5.11.8 : Use Quine Mc-Cluskey method to simplify the logic function as given below.
$f(A, B, C, D, E) = \Sigma m(0, 1, 8, 10, 11, 12, 20, 21, 30) + d(14, 19)$
Realize the above function using NAND gates. **Dec. 13. 15 Marks**

Soln. :

$$f(A, B, C, D, E) = \Sigma m(0, 1, 8, 10, 11, 12, 20, 21, 30) + d(14, 19)$$

Step 1 : Group the minterms according to number of 1's :

Group	Min term	Binary representation A	B	C	D	E	
0	0	0	0	0	0	0	✓
1	1	0	0	0	0	1	✓
	8	0	1	0	0	0	✓
2	10	0	1	0	1	0	✓
	12	0	1	1	0	0	✓
	20	1	0	1	0	0	✓
3	11	0	1	0	1	1	✓
	14	0	1	1	1	0	✓
	19	1	0	0	1	1	$A\bar{B}\bar{C}DE$
	21	1	0	1	0	1	✓
4	30	1	1	1	1	0	$ABCD\bar{E}$

Step 2 : Group the minterms to form pairs :

Group	Min term	Binary representation A	B	C	D	E	
0	0–1	0	0	0	0	–	$\bar{A}\bar{B}\bar{C}\bar{D}$
	0–8	0	–	0	0	0	$\bar{A}\bar{C}\bar{D}\bar{E}$
1	8–10	0	1	0	–	0	✓
	8–12	0	1	–	0	0	✓
2	10–11	0	1	0	1	–	$\bar{A}B\bar{C}D$
	10–14	0	1	–	1	0	✓
	20–21	1	0	1	0	–	$A\bar{B}\bar{C}\bar{D}$
	12–14	0	1	1	–	0	✓
3	14–30	–	1	1	1	0	$BCD\bar{E}$

Scanned by CamScanner

Step 3 : Group the minterms to form groups of four :

Group	Minterm quad	Binary representation					
		A	B	C	D	E	
1	8–10–12–14	0	1	–	–	0	$\bar{A}B\bar{E}$
	8–12–10–14	0	1	–	–	0	

Step 4 : Prepare the table prime implicants :

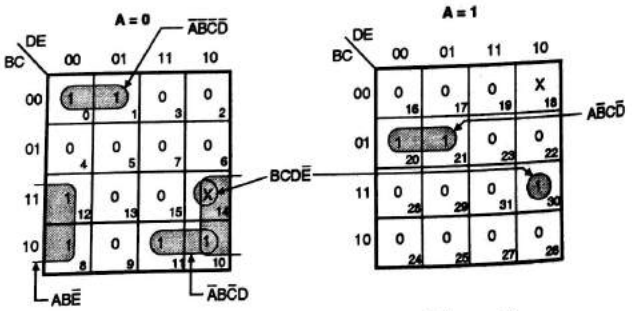
Prime implicants	Decimal numbers	Given minterms										
		0	1	8	10	11	12	20	21	30	14	19
$A\bar{B}\bar{C}DE$	19											X
$ABCD\bar{E}$	30									X		
$\bar{A}\bar{B}\bar{C}\bar{D}$	0, 1	X	X									
$\bar{A}\bar{C}\bar{D}\bar{E}$	0, 8	X		X								
$\bar{A}B\bar{C}D$	10, 11				X	X						
$A\bar{B}C\bar{D}$	20, 21							X	X			
$BCD\bar{E}$	14, 30									X	X	
$\bar{A}B\bar{E}$	8, 10, 12, 14			X	X		X				X	

(C-4211)

The encircled crosses represent the essential prime implicants. They are $\bar{A}\bar{B}\bar{C}\bar{D}$, $\bar{A}B\bar{C}D$, $A\bar{B}\bar{C}\bar{D}$, $\bar{A}B\bar{E}$ which cover minterms. 0, 1, 8, 10, 11, 12, 20, 21. $BCD\bar{E}$ cover minterm 30.

$$\therefore \quad f(A, B, C, D, E) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + A\bar{B}C\bar{D} + \bar{A}B\bar{E} + BCD\bar{E}$$

Step 5 : Cross check using K-map :



$$\therefore \quad f(A,B,C,D,E) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + A\bar{B}C\bar{D} + \bar{A}B\bar{E} + BCD\bar{E}$$

(C-4212) **Fig. P. 5.11.8**

Ex. 5.11.9 : Using Quine Mc Cluskey method determine minimal SOP form for :

$$F(A, B, C, D) = \Sigma\, m\,(1, 2, 3, 6, 7, 10, 12, 14)$$

May 14, 10 Marks

Soln. :

$$(A, B, C, D) = \Sigma\, m\,(1, 2, 3, 6, 7, 10, 12, 14)$$

Step 1 : Group the minterms according to number of 1's :

Group	Minterm	Binary representation				
		A	B	C	D	
1	1	0	0	0	1	✓
	2	0	0	1	0	✓
2	3	0	0	1	1	✓
	6	0	1	1	0	✓
	10	1	0	1	0	✓
	12	1	1	0	0	✓
3	7	0	1	1	1	✓
	14	1	1	1	0	✓

Step 2 : Group the minterms to form pairs :

Group	Minterm pair	Binary representation				
		A	B	C	D	
1	1-3	0	0	–	1	$\bar{A}\bar{B}D$
	2-3	0	0	1	–	$\bar{A}\bar{B}C$
	2-6	0	–	1	0	✓
	2-10	–	0	1	0	✓
2	3-7	0	–	1	1	$\bar{A}CD$
	6-14	–	1	1	0	✓
	10-14	1	–	1	0	✓
	12-14	1	1	–	0	$AB\bar{D}$
	6-7	0	1	1	–	

Step 3 : Group the minterms to form groups of four :

Group	Minterm quad	Binary representation				
		A	B	C	D	
1	2-6-10-14	–	–	1	0	$C\bar{D}$
	2-10-6-14	–	–	1	0	
	2-6-3-7	0	–	1	–	$\bar{A}C$
	2-3-6-7	0	–	1	–	

Scanned by CamScanner

Step 4 : Prepare table of prime implicants :

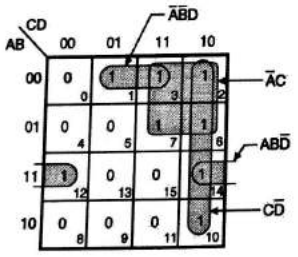
(C-4963)

Prime implicants	Decimal numbers	Given minterms							
		1	2	3	6	7	10	12	14
$\bar{A}\bar{B}D$	1, 3	(X)		X					
$\bar{A}\bar{B}C$	2, 3		X	X					
$\bar{A}CD$	3, 7			X		X			
$AB\bar{D}$	12, 14							(X)	X
$C\bar{D}$	2, 6, 10, 14		X		X		(X)		X
$\bar{A}C$	2, 3, 6, 7		X	X	X	X			

The encircled crosses represent the essential prime implicants. They are $\bar{A}\bar{B}D$, $C\bar{D}$, $AB\bar{D}$ which cover the minterms 1, 2, 3, 6, 10, 12, 14. Now check the remaining minterms. i.e. 7. The PI $\bar{A}C$ cover minterm 7. So they should be included.

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

Step 5 : Crosscheck using K-map :



(C-4964) **Fig. P. 5.11.9 : K-map**

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

Ex. 5.11.10 : Minimize the logic function using Quine-Mc Cluskey method.

$$f(A, B, C, D) = \sum m(1, 3, 7, 9, 10, 11, 13, 15)$$

May 15. 12 Marks. May 16. 10 Marks

Soln. : Step 1 : Group the minterms according to number of 1's :

Table P. 5.11.10(a)

Group	Minterm	Binary representation				
		A	B	C	D	
1	1	0	0	0	1	One 1
2	3	0	0	1	1	Two 1's
	9	1	0	0	1	
	10	1	0	1	0	
3	7	0	1	1	1	Three 1's
	11	1	0	1	1	
	13	1	1	0	1	
4	15	1	1	1	1	Four 1's

(C-5196)

Step 2 : Group the minterms to form the pairs :

Table P. 5.11.10(b)

Group	Minterm pair	Binary representation				
		A	B	C	D	
1	1 – 3	0	0	–	1	
	1 – 9	–	0	0	1	
2	3 – 7	0	–	1	1	
	9 – 11	1	0	–	1	
	9 – 13	1	–	0	1	
	10 – 11	1	0	1	–	$A\bar{B}C$
	3 – 11	–	0	1	1	
3	7 – 15	–	1	1	1	
	11 – 15	1	–	1	1	
	13 – 15	1	1	–	1	

Step 3 : Group the minterms to form groups of four :

Table P. 5.11.10(c)

Group	Minterm quad	Binary representation				Prime implicants
		A	B	C	D	
1	1, 3, 9, 11	–	0	–	1	$\bar{B}D$
	1, 9, 3, 11	–	0	–	1	
2	3, 7, 11, 15	–	–	1	1	CD
	3, 11, 7, 15	–	–	1	1	
	9, 11, 13, 15	1	–	–	1	AD
	9, 13, 11, 15	1	–	–	1	

(C-5197)

Scanned by CamScanner

Step 4 : **Prepare the table of prime implicants :**

Table P. 5.11.10(d)

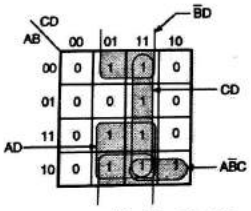
Prime implicants	Decimal numbers	Given minterms							
		1	3	7	9	10	11	13	15
$A\bar{B}C$	10, 11					(X)	X		
$\bar{B}D$	1, 3, 9, 11	(X)	X		X		X		
CD	3, 7, 11, 15		X	(X)			X		X
AD	9, 11, 13, 15				X		X	(X)	X

(C-5199)

The encircled crosses represent the EPIs. They cover minterms 1, 3, 7, 9, 10, 11, 13 and 15.

$\therefore f(A, B, C, D) = A\bar{B}C + \bar{B}D + CD + AD$ ...Ans.

Step 5 : **Crosscheck using K-map :**



$\therefore f(A,B,C,D) = A\bar{B}C + \bar{B}D + CD + AD$

(C-5199) **Fig. P. 5.11.10**

Ex. 5.11.11 : Reduce using Quine Mc-Cluskey method and realize the equation using only NAND gates.

$F(P, Q, R, S) = \sum m(0, 1, 2, 8, 10, 11, 14, 15)$ Dec. 15, Dec. 16, 10 Marks

Soln. :

Step 1 : **Group the minterms according to number of 1's :**

Table P. 5.11.11(a)

Group	Minterm	Binary representation				
		P	Q	R	S	
0	0	0	0	0	0	No 1's
1	1	0	0	0	1	
	2	0	0	1	0	One 1
	8	1	0	0	0	
2	10	1	0	1	0	Two 1's
3	11	1	0	1	1	
4	14	1	1	1	0	Three 1's
	15	1	1	1	1	Four 1's

Step 2 : **Group the minterms to form pairs :**

Table P. 5.11.11(b)

Group	Minterm pair	Binary representation				
		P	Q	R	S	
0	0, 1	0	0	0	–	$\bar{P}\bar{Q}\bar{R}$
	0, 2	0	0	–	0	
	0, 8	–	0	0	0	✓
1	2, 10	–	0	1	0	✓
	8, 10	1	0	–	0	✓
2	10, 11	1	0	1	–	✓
	10, 14	1	–	1	0	✓
3	11, 15	1	–	1	1	✓
	14, 15	1	1	1	–	✓

Step 3 : **Group the minterms to form quads :**

Table P. 5.11.11(c)

Group	Minterm Quad	Binary representation				
		P	Q	R	S	
0	0, 2, 8, 10	–	0	–	0	$\bar{Q}\bar{S}$
	0, 8, 2, 10	–	0	–	0	
2	10, 11, 14, 15	1	–	1	–	PR
	10, 14, 11, 15	1	–	1	–	

Step 4 : **Prepare the table of prime implicants :**

(C-5268) **Table P. 5.11.11(d)**

Prime implicant	Decimal numbers	Given minterms							
		0	1	2	8	10	11	14	15
$\bar{P}\bar{Q}\bar{R}$	0, 1	X	(X)						
$\bar{Q}\bar{S}$	0, 2, 8, 10	X		(X)	(X)	X			
PR	10, 11, 14, 15					X	(X)	(X)	(X)

The encircled crosses represent the EPs. So the terms $\bar{P}\bar{Q}\bar{R}$, $\bar{Q}\bar{S}$ and PR are EPs. They cover all the minterms 0, 1, 2, 8, 10, 11, 14 and 15.

$\therefore f(P, Q, R, S) = \bar{P}\bar{Q}\bar{R} + \bar{Q}\bar{S} + PR$ ...Ans.

Scanned by CamScanner

Step 5 : Crosscheck using K-map :

PQ \ RS	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	1	1
10	1	0	1	1

$\overline{P}\,\overline{Q}\,\overline{R}$ $\overline{Q}\,\overline{S}$ PR

(C-5269) **Fig. P. 5.11.11 : K-map**

$$\therefore\ f = (P, Q, R, S) = \overline{P}\,\overline{Q}\,\overline{R} + \overline{Q}\,\overline{S} + PR$$

This is same as the result obtained with Quine Mc-Cluskey method.

Review Questions

Q. 1 Explain the K-map reduction technique.

Q. 2 Draw the structure of four input K-map.

Q. 3 Why is gray code followed for K-map ?

Q. 4 Define a redundant group.

Q. 5 State the importance of don't care terms in K-map.

Q. 6 Draw the structure of four variable K-map to represent the standard POS form.

Q. 7 Solve the following with K-maps :

1. $f(A,B,C) = \Sigma m(0,1,3,4,5)$ 2. $f(A,B,C) = \Sigma m(0,1,2,3,6,7)$

Q. 8 Explain the different methods used to simplify the Boolean function.

□□□

CHAPTER 6

Combinational Logic Design

Module 3

Syllabus :

Introduction, Half and Full adder, Half subtractor, Full subtractor, Four Bit ripple adder, Look ahead carry adder, 4 bit adder subtractor, One digit BCD Adder, Multiplexer, Multiplexer tree, Demultiplexer, Demultiplexer tree, Encoders : Priority encoder, Decoders, One bit, Two bit , 4-bit Magnitude comparator, ALU IC 74181.

6.1 Introduction to Combinational Circuits :

Types of digital circuits :

- The digital systems in general are classified into two categories namely :
  1. Combinational logic circuits 2. Sequential logic circuits.

Combinational circuits :

- The combinational circuit is a digital system the output of which at any instant of time, depends only on the levels present at its input terminals.
- The combinational circuits do not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit.
- The sequence in which the inputs are being applied also does not have any effect on the output of a combinational circuit.
- A combinational circuit is a logic circuit the output of which depends only on the combination of the inputs. The output does not depend on the past value of inputs or outputs. Hence combinational circuits do not require any memory (to store the past values of inputs or outputs).
- The block diagram of a combinational circuit is shown in Fig. 6.1.1.
- A combinational circuit can have a number of inputs and a number of outputs. The circuit of Fig. 6.1.1 has "n" inputs and "m" outputs.

(C-332) **Fig. 6.1.1 : Block diagram of a combinational circuit**

- A combinational circuit is made up of logic gates. These gates are connected suitably between the inputs and outputs of the combinational circuit.
- A combinational circuit operates in three steps :
  1. It accepts n-different inputs. 2. The combination of gates operates on the inputs.
  3. "m" different outputs are produced as per requirement.

Examples of combinational circuits :

- Following are some of the examples of combinational circuits :
  1. Adders, subtractors 2. Comparator 3. Code converters
  4. Encoders, decoders 5. Multiplexers, demultiplexers

Scanned by CamScanner

6.1.1 Analysis of a Combinational Circuit :

- In the analysis problem, the logic diagram of a combinational circuit is given to us. We have to obtain the Boolean expression for its output or write a truth table or explain the operation of circuit.

Procedure :

- The general procedure for the analysis is as follows :
  1. Write down the Boolean function for the output of each input gate in the given circuit.
  2. Obtain the Boolean expression at all other gates.
  3. Write down the Boolean expression for the final output in terms of the input variables.

Illustration : Analyze the combinational circuit shown in Fig. 6.1.2.

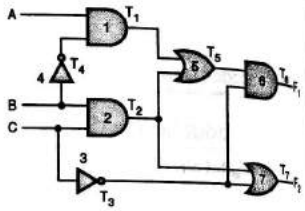
Analysis :

Step 1 : Boolean expressions for the outputs of all the input gates :

Input gates are 1, 2, 3 and 4. The Boolean expressions for their outputs are as follows :

$$T_1 = A\bar{B} \qquad T_3 = \bar{C}$$

$$T_2 = BC \qquad T_4 = \bar{B}$$



(C-333) **Fig. 6.1.2 : Given combinational circuit**

Step 2 : Boolean expressions for remaining gates :

$$T_5 = T_1 + T_2 = A\bar{B} + BC$$

$$T_6 = F_1 = T_5 \cdot T_3 = (A\bar{B} + BC)\,\bar{C} = A\bar{B}\bar{C} + BC\bar{C} = A\bar{B}\bar{C}$$

$$T_7 = F_2 = T_2 + T_3 = BC + \bar{C}$$

Step 3 : Boolean expressions for final outputs :

$$F_1 = T_6 = A\bar{B}\bar{C}$$

$$F_2 = T_7 = \bar{C} + BC$$

Step 4 : Write the truth table :

Table 6.1.1

Inputs			Outputs						
A	B	C	F_1	F_2	T_1	T_2	T_3	T_4	T_5
0	0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	0	1	0
0	1	0	0	1	0	0	1	0	0
0	1	1	0	1	0	1	0	0	1
1	0	0	1	1	1	0	1	1	1
1	0	1	0	0	1	0	0	1	1
1	1	0	0	1	0	0	1	0	0
1	1	1	0	1	0	1	0	0	1

- Note that the outputs of all the gates (T_1 to T_5) are first written in the truth table for various combinations of inputs and then the values of F_1 and F_2 are entered using the following equations.

$$F_1 = T_3 \cdot T_5 \qquad \text{and} \qquad F_2 = T_2 + T_3$$

6.1.2 Design of Combinational Logic using Statements :

- The steps involved in designing a combinational logic are as follows :

Steps to be followed :

Step 1 : You will be given a problem.

Step 2 : Determine the number of inputs and outputs and assign letter symbols to input and output variables. For example F_1, F_2 ... for outputs and A, B, C. .... for the inputs.

Step 3 : Prepare a truth table relating the inputs and outputs.

Step 4 : Write K-map for each output in terms of inputs and obtain the simplified Boolean expression for each output.

Step 5 : Draw the logic diagram (combinational circuit).

Ex. 6.1.1 : Design and draw a combinational circuit that multiplies two 2-bit numbers A_1A_2 and B_1B_2 to produce 4-bit product $C_3C_2C_1C_0$. **May 10, Dec. 12, 10 Marks**

Soln. :

Step 1 : Write the truth table :

Table P. 6.1.1

Inputs				Outputs			
A_1	A_2	B_1	B_2	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Step 2 : Write K-maps for C_3, C_2, C_1, C_0 and simplify :

For C_3

A_1A_2 \ B_2B_1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	0	0

$A_1A_2B_1B_2$

$\therefore C_3 = A_1A_2B_1B_2$

(C-3262) **Fig. P. 6.1.1(a)**

For C_2

A_1A_2 \ B_2B_1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	1
10	0	0	1	1

$A_1B_1\bar{B}_2$

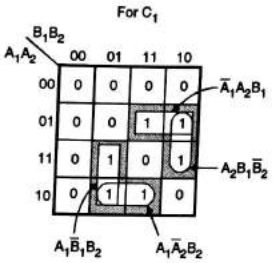
$A_1\bar{A}_2B_1$

$\therefore C_2 = A_1\bar{A}_2B_1 + A_1B_1\bar{B}_2$

$\therefore C_2 = A_1B_1(\bar{A}_2 + \bar{B}_2)$

(C-3263) **Fig. P. 6.1.1(b)**

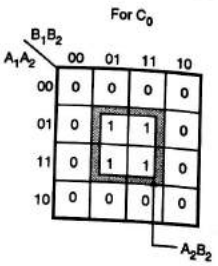
Scanned by CamScanner



$\therefore\ C_1 = A_1\bar{B}_1B_2 + A_1\bar{A}_2B_2 + \bar{A}_1A_2B_1 + A_2B_1\bar{B}_2$

$\therefore\ C_1 = A_1B_2(\bar{B}_1 + \bar{A}_2) + A_2B_1(\bar{A}_1 + \bar{B}_2)$

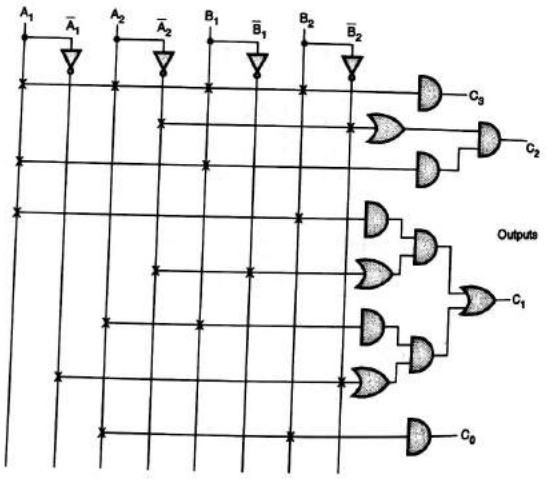
(C-3264) **Fig. P. 6.1.1(c)**



$\therefore\ C_0 = A_2B_2$

(C-3265) **Fig. P. 6.1.1(d)**

Step 3 : Implementation :



(C-3266) **Fig. P. 6.1.1(e)**

Ex. 6.1.2 : Air India complex has four elevators for visitors. To save on power only two elevator cars are available. If the traffic is heavy or if car 1 is shutdown due to technical problem, the third elevator car is switched ON. The fourth elevator car is a standby car which is powered ON if both car 1 and car 2 fails. Design a logic circuit for starting power to car 3 and car 4. **May 10, 10 Marks**

Soln. :

Step 1 : Write logical statements related to turning ON of cars 3 and 4 :

Car-3 :

Car-3 will be turned on if :

1. Traffic is heavy or
2. Car-1 is shut down due to technical problems.

Car-4 :

Car-4 will be turned on if :

1. Both Car-1 and Car-2 fail.

Step 2 : Prepare the truth table :

Let the inputs to the combinational circuits be defined as follows :

Input A : Traffic is heavy.

Input B : Car-1 is shut down.

Input C : Both Car-1 and Car-2 fail.

The outputs of the combinational circuits are as follows :

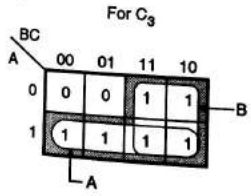
Output C_3: Car-3 is turned ON.

Output C_4 : Car-4 is turned ON.

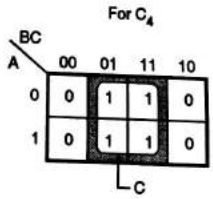
Table P. 6.1.2

Inputs			Outputs	
A	B	C	C_3	C_4
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

Step 3 : K-maps :



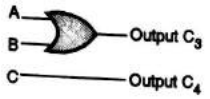
$\therefore\ C_3 = A + B$



$\therefore\ C_4 = C$

(C-3269) **Fig. P. 6.1.2(a) : K-maps**

Step 4 : Realization of logic circuit :



(C-3270) **Fig. P. 6.1.2(b) : Realization of logic circuit**

Scanned by CamScanner

Ex. 6.1.3 : Design a 4-input (A, B, C, D) digital circuit that will give at its output (X) a logic 1 only if the binary number formed at the input is between 2 and 9 (including). **May 15. 10 Marks**

Soln. :

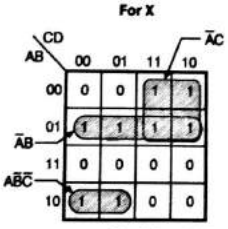
Step 1 : **Write the truth table :**

Table P. 6.1.3

Decimal	Inputs				Output X
	A	B	C	D	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

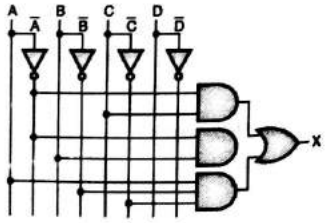
Step 2 : **K-map and simplification :**

For X



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

Step 3 : Draw the logic diagram :



(C-5195) **Fig. P. 6.1.3 : Logical diagram**

Ex. 6.1.4 : Input to a combinational circuit is a 4-bit binary number. Design the circuit with minimum hardware for the following :

1. Output P = 1 if the number is prime.
2. Output Q = 1 if the number is divisible by 3.

Dec. 15. 10 Marks

Soln. :

Step 1 : **Write the truth table :**

Prime numbers are : 1, 2, 3, 5, 7, 11 and 13.

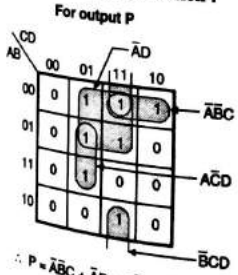
Numbers divisible by 3 are : 0, 3, 6, 9, 12, 15.

Table P. 6.1.4

Decimal	Inputs				Outputs	
	A	B	C	D	P	Q
0	0	0	0	0	0	0
1	0	0	0	1	1	0
2	0	0	1	0	1	0
3	0	0	1	1	1	1
4	0	1	0	0	0	0
5	0	1	0	1	1	0
6	0	1	1	0	0	1
7	0	1	1	1	1	0
8	1	0	0	0	0	0
9	1	0	0	1	0	1
10	1	0	1	0	0	0
11	1	0	1	1	1	0
12	1	1	0	0	0	1
13	1	1	0	1	1	0
14	1	1	1	0	0	0
15	1	1	1	1	0	1

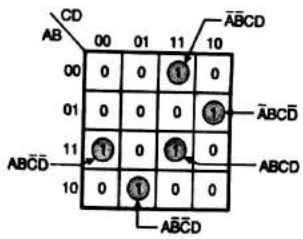
Step 2 : **K-maps and simplification :**

For output P



$\therefore P = \bar{A}\bar{B}C + \bar{A}D + A\bar{C}D + \bar{B}CD$

For output Q

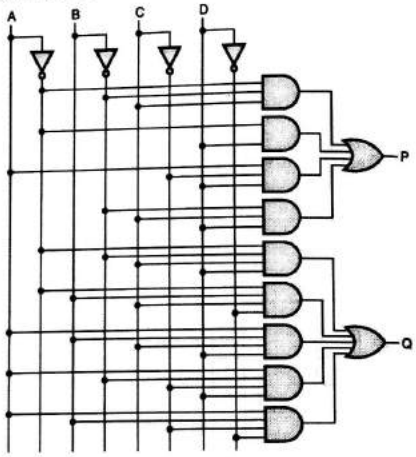


(C-5270)

$\therefore Q = \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + ABCD + A\bar{B}\bar{C}D + AB\bar{C}\bar{D}$

Scanned by CamScanner

Step 3 : Draw the logic diagram :



(C-5271) **Fig. P. 6.1.4 : Logic diagram**

6.2 Code Converters :

MU : May 08

University Questions

Q. 1 Write short note on : Code converters **(May 08, 5 Marks)**

In this section we are going to convert one type of code into another type.

- Following are the examples of code converters. We will discuss some of them.
  1. Binary to BCD converter.
  2. BCD to Binary converter.
  3. BCD to Excess 3.
  4. Excess 3 to BCD.
  5. Binary to Gray code converter.
  6. Gray code to Binary converter.
  7. BCD to Gray code converter.

6.2.1 BCD to Excess 3 Converter :

- We know that Excess-3 code can be derived from the BCD code by adding 3 to each BCD number.
- For example decimal 13 is represented as 0001 0011 in BCD.
  If we add 3 i.e. (0011 0011) then the corresponding Excess 3 code is 0100 0110.

Step 1 : Write the truth table relating BCD and Excess 3 :

Table 6.2.1 : Truth table relating BCD and Excess-3 codes

Decimal	BCD inputs				Excess-3 outputs			
	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0

Decimal	BCD inputs				Excess-3 outputs			
	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Step 2 : Write K-map for each output and obtain simplified expression :

Refer Figs. 6.2.1(a) to (d) for the K-maps and corresponding simplified equations. Don't care conditions (x) have been entered for $D_3 D_2 D_1 D_0$ above 1001.

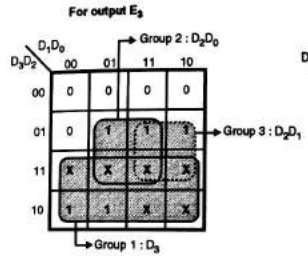
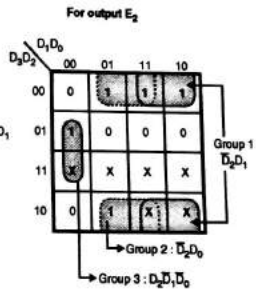


Fig. 6.2.1(a) : K-map for E_3

Simplified equation : $E_3 = D_3 + D_2D_0 + D_2D_1$

$\therefore\ E_3 = D_3 + D_2(D_0 + D_1)$



(C-365) **Fig. 6.2.1(b) : K-map for E_2**

Simplified equation : $E_2 = \overline{D}_2 D_1 + \overline{D}_2 D_0 + D_2 \overline{D}_1 \overline{D}_0$

$\therefore\ E_2 = \overline{D}_2 (D_1 + D_0) + D_2 \overline{D}_1 \overline{D}_0$

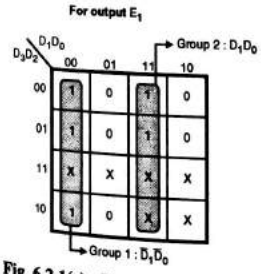
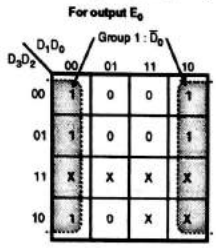


Fig. 6.2.1(c) : K-map for E_1

$\therefore E_1 = \overline{D}_1\ \overline{D}_0 + D_1D_0 = (D_1 \odot D_0)$



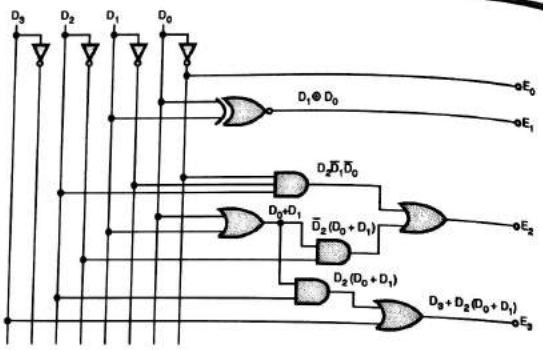
(C-366) **Fig. 6.2.1(d) : K-map for E_0**

$\therefore E_0 = \overline{D}_0$

Step 3 : Implement the BCD to Excess 3 code converter :

The circuit diagram of BCD to Excess 3 code converter is shown in Fig. 6.2.1.

Scanned by CamScanner



(C-367) **Fig. 6.2.2 : BCD to Excess – 3 code converter**

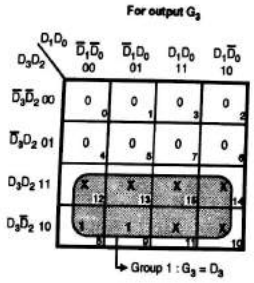
6.2.2 BCD to Gray Code Converter :

Step 1 : Write the truth table relating BCD and gray codes :

Table 6.2.2

Decimal	BCD inputs				Gray outputs			
	D_3	D_2	D_1	D_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1

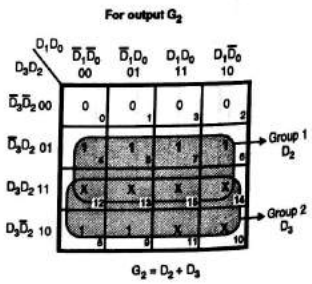
Step 2 : Write K-map for each output and get simplified equation :



(C-371) **Fig. 6.2.3(a) : K-map for G_3**

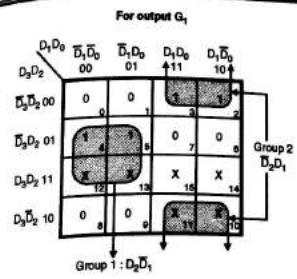
$\therefore G_3 = D_3$

and



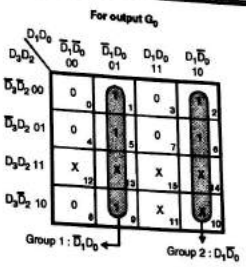
(C-371) **Fig. 6.2.3(b) : K-map for G_2**

$G_2 = D_2 + D_3$



(C-372) **Fig. 6.2.3(c) : K-map for G_1**

$\therefore G_1 = D_2\overline{D}_1 + \overline{D}_2D_1 = (D_2 \oplus D_1)$

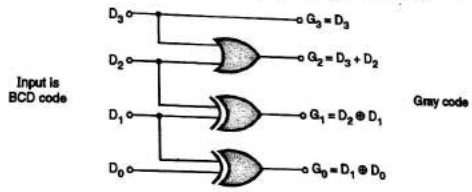


(C-372) **Fig. 6.2.3(d) : K-map for G_0**

$\therefore G_0 = (\overline{D}_1D_0 + D_1\overline{D}_0) = D_1 \oplus D_0$

Step 3 : Realization :

The BCD to gray converter is shown in Fig. 6.2.4.



(C-373) **Fig. 6.2.4 : BCD to Gray code converter**

6.2.3 Binary to Gray Code Converter :

MU : May 12

University Questions

Q. 1 Design a 4 bit binary to gray code converter. **(May 12, 10 Marks)**

In gray code only one bit changes at a time.

Step 1 : Write the truth table relating binary inputs and gray outputs :

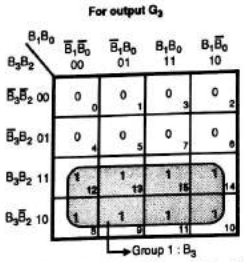
Table 6.2.3 : Truth table relating binary and gray codes

Decimal	Binary inputs				Gray outputs			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0

Scanned by CamScanner

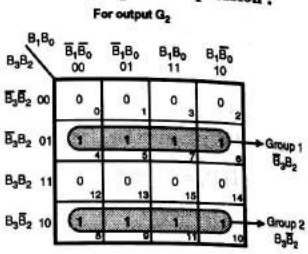
Decimal	Binary inputs				Gray outputs			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Step 2 : Write K-map for each gray output and obtain the simplified expression :



(C-374) **Fig. 6.2.5(a) : K-map for G_3**

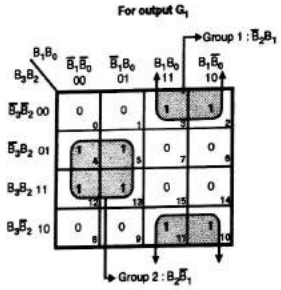
$\therefore$ Simplified equation : $G_3 = B_3$



(C-374) **Fig. 6.2.5(b) : K-map for G_2**

$\therefore$ Simplified equation : $G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2$

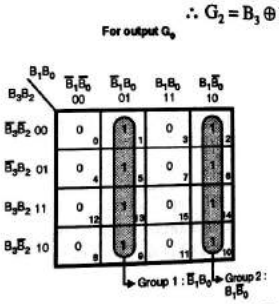
$\therefore G_2 = B_3 \oplus B_2$



(C-375) **Fig. 6.2.5(c) : K-map for G_1**

$\therefore$ Simplified equation : $G_1 = B_2 \bar{B}_1 + \bar{B}_2 B_1$

$= B_2 \oplus B_1$



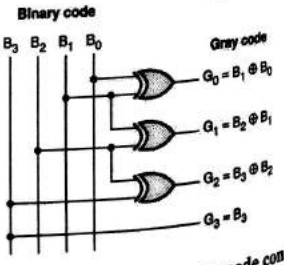
(C-375) **Fig. 6.2.5(d) : K-map for G_0**

$\therefore$ Simplified equation : $G_0 = \bar{B}_1 B_0 + B_1 \bar{B}_0$

$= B_1 \oplus B_0$

Step 3 : Realize Binary to gray code converter :

Binary to gray code converter is shown in Fig. 6.2.5(e).



(C-376) **Fig. 6.2.5(e) : Binary to gray code converter**

6.2.4 Gray to BCD Code Converter :

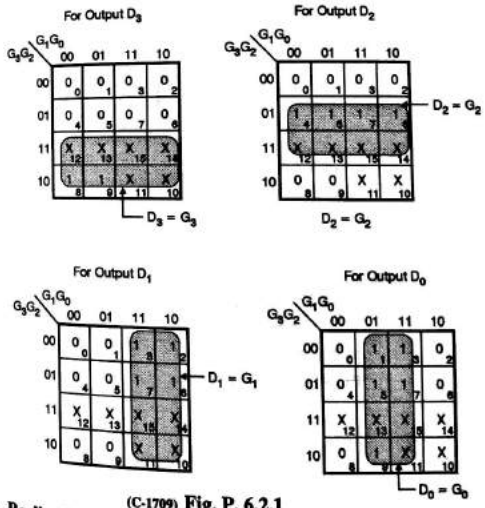
Ex. 6.2.1 : Convert 4-bit gray code into corresponding BCD code. Show truth table and implement using gates. **May 08, May 16, 10 Marks**

Soln. :

Step 1 : Write the truth table relating gray and BCD codes :

Decimal	Gray inputs				BCD outputs			
	G_3	G_2	G_1	G_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1

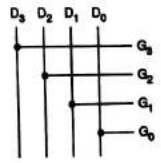
Step 2 : Write K-maps for each output and get simplified equation :



(C-1709) **Fig. P. 6.2.1**

Step 3 : Realization :

The gray to BCD converter is shown in Fig. P. 6.2.1(a).



(C-1710) **Fig. P. 6.2.1(a)**

Scanned by CamScanner

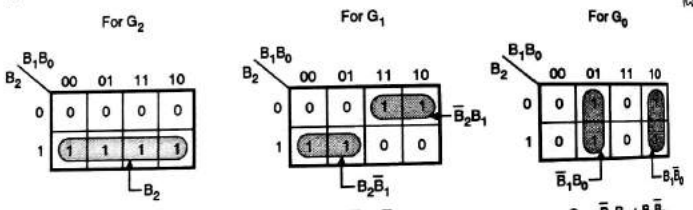
Ex. 6.2.2 : Design 3 bit Binary to gray code converter. May 10, 5 Marks

Soln. :

Step 1 : Write the truth table :

Decimal	Binary inputs			Gray outputs		
	B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

Step 2 : Write K-map for each output and get simplified equation :

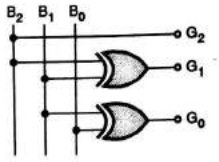


$G_2 = B_2$

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

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

Step 3 : Realization :



(C-2204)

6.3 Binary Adders and Subtractors :

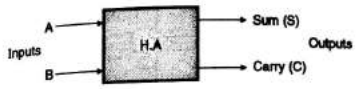
Addition of two binary digits is most basic operation performed by the digital computers.

6.3.1 Types of Binary Adders :

- In this section we are going to learn digital circuits which are used to "add" two "binary" numbers. The binary adders are of two types :
  1. Half adder and 2. Full adder

6.3.2 Half Adder :

- Half adder is a combinational logic circuit with two inputs and two outputs. It is the basic building block for addition of two "single" bit numbers. This circuit has two outputs namely "carry" and "sum". The block diagram of half adder is as shown in Fig. 6.3.1(a).
- The half adder circuit is supposed to add two single bit binary numbers A and B. Therefore the truth table of a half adder is as shown in Fig. 6.3.1(b).



(a) Block diagram

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Truth table

(C-344) **Fig. 6.3.1 : Half adder**

Karnaugh maps and simplified expressions for outputs :

- K-maps for carry and sum outputs are as shown in Figs. 6.3.2(a) and (b).

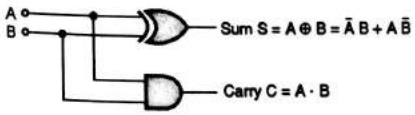
(a) K-map or sum output (b) K-map or carry out

(C-345) **Fig. 6.3.2**

- Boolean expressions for the sum (S) and carry (C) output are obtained from the K-maps as follows :

$$\left.\begin{aligned} S &= \bar{A}B + A\bar{B} = A \oplus B \\ C &= AB \end{aligned}\right\} \quad \ldots(6.3.1)$$

- The disadvantage of half adder is that addition of three bits is not possible to perform.
- Hence the half adder circuit is as shown in Fig. 6.3.3.



(C-346) **Fig. 6.3.3 : Half adder circuit**

Disadvantage of half adder :

- The principle of adding two 2-bit numbers A and B is as shown in Fig. 6.3.4(a).

Let Number A = $A_1 A_0$ and Number B = $B_1 B_0$

Then the addition should take place as shown in Fig. 6.3.4(a).

Scanned by CamScanner

- A half adder can add A_0 and B_0 to produce S_0 and C_0. But the addition of next bits requires the addition of A_1, B_1 and C_0. The addition of three bits is not possible to perform by using a half adder. Hence we cannot use a half adder in practice.

$$\begin{array}{rll} & A_1 & A_0 \leftarrow \text{Number A} \\ + & B_1 & B_0 \leftarrow \text{Number B} \\ + & \boxed{C_0} & \leftarrow \text{Carry generated from the addition } (A_0+B_0) \\ \hline \boxed{C_1} & S_1 & S_0 \end{array}$$

(C-347) **Fig. 6.3.4(a) : Two bit binary addition**

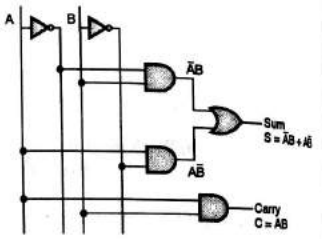
Half adder using basic gates :

- Refer Equation (6.3.1) which states that,

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

$$\text{and Carry } C = AB$$

- These equations are implemented using the basic gates as shown in Fig. 6.3.4(b).



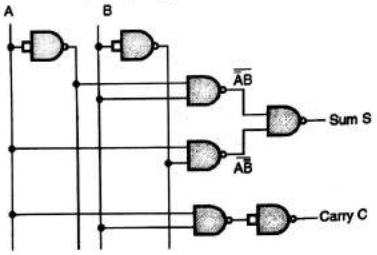
(C-348) **Fig. 6.3.4(b) : Half adder using basic gates**

Half adder using only NAND gates :

- Equation (6.3.1) states that,
- Sum, $S = \bar{A}B + A\bar{B}$ and Carry, $C = AB$
- Consider the expressions separately, $S = \bar{A}B + A\bar{B}$
- Take double inversion to get, $S = \overline{\overline{\bar{A}B + A\bar{B}}}$

Using the De Morgan's theorem, $S = \overline{\overline{\bar{A}B} \cdot \overline{A\bar{B}}}$ ...(Using only NAND) ...(6.3.2)

- Similarly $C = AB = \overline{\overline{AB}}$ .....(Using only NAND) ...(6.3.3)



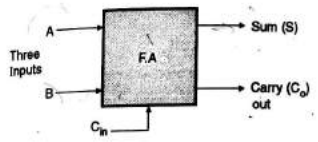
(C-349) **Fig. 6.3.4(c) : Half adder using only NAND gates**

- Equations (6.3.2) and (6.3.3) are implemented using only NAND gates as shown in Fig. 6.3.4(c).

6.3.3 Full Adder :

- To overcome the drawback of Half Adder circuit, a 3 single bit adder circuit called Full Adder is developed.

- It can add two one-bit numbers A and B, and carry C_{in}. The full adder is a three input and two output combinational circuit.
- The block diagram of a full adder is as shown in Fig. 6.3.5(a) and its truth table is given in Fig. 6.3.5(b).



(a) Block diagram

Inputs			Outputs	
A	B	C_{in}	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

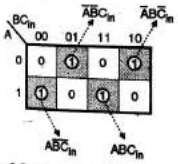
(b) Truth table

(C-350) **Fig. 6.3.5 : Full adder**

The K-maps :

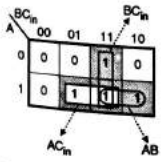
- The K-maps for the sum (S) and carry out (C_o) outputs and the corresponding Boolean expressions are as shown in Fig. 6.3.6. Note that the K-maps have been written from the truth table of Fig. 6.3.5(b).

For the sum output :



(C-351) **Fig. 6.3.6(a) : K-map for sum output**

For carry output :



(C-352) **Fig. 6.3.6(b) : K-map for carry output**

Expression for sum output :

$$S = \bar{A}\bar{B}C_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} + \bar{A}B\bar{C}_{in}$$

$$S = C_{in}\underbrace{(\bar{A}\bar{B} + AB)}_{\text{EX-NOR}} + \bar{C}_{in}\underbrace{(\bar{A}B + A\bar{B})}_{\text{EX-OR}}$$

$$\therefore S = C_{in}\,\overline{(\bar{A}B + A\bar{B})} + \bar{C}_{in}(\bar{A}B + A\bar{B})$$

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

$$\therefore S = C_{in}\bar{X} + \bar{C}_{in}X = C_{in} \oplus X$$

$$\therefore S = C_{in} \oplus (\bar{A}B + A\bar{B})$$

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

$$\therefore S = C_{in} \oplus A \oplus B \quad ...(6.3.4)$$

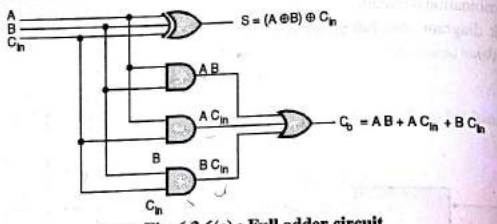
Expression for carry output :

$$C_o = AB + AC_{in} + BC_{in} \quad ...(6.3.5)$$

- We can use Equations (6.3.4) and (6.3.5) to draw the logic diagram of a full adder as shown in Fig. 6.3.6(c).

Scanned by CamScanner

Logic diagram for full adder :



(C-353) **Fig. 6.3.6(c) : Full adder circuit**

6.3.4 Full Adder using Half Adder :

MU : Dec. 08, May 11, Dec. 12, May 14, May 16

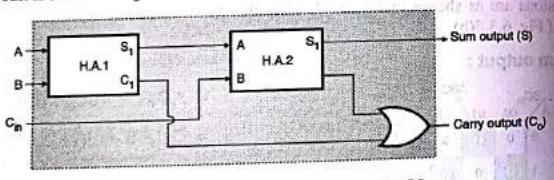
University Questions

Q. 1 Design full adder using half adders. **(Dec. 08, 4 Marks)**

Q. 2 Design a full adder circuit using half adders and some gates. **(May 11, Dec. 12, 10 Marks, May 16, 5 Marks)**

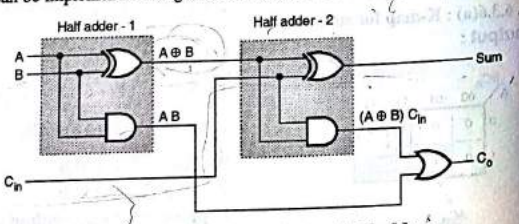
Q. 3 Design a full adder using half adder and additional gates. **(May 14, 5 Marks)**

- The full adder circuit can be constructed using two half adders as shown in Fig. 6.3.7 and the detail circuit is shown in Fig. 6.3.8.



(C-354) **Fig. 6.3.7 : Full adder using half adders**

- A full adder can be implemented using two half adders and an OR gate as shown in Fig. 6.3.8.



(C-355) **Fig. 6.3.8 : Full adder using two half adders**

- Now let us prove that this circuit acts as a full adder.

Proof :

- Refer Fig. 6.3.8 and write the expression for sum output as,

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

This expression is same as that obtained for the full adder. Thus the sum output has been successfully implemented by the circuit shown in Fig. 6.3.8.

- Now write the expression for carry output C_o as,

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

$$C_o = (\bar{A}B + A\bar{B})\, C_{in} + AB = \bar{A}BC_{in} + A\bar{B}C_{in} + AB$$

$$= \bar{A}BC_{in} + A\bar{B}C_{in} + AB\,(1 + C_{in}) = \bar{A}BC_{in} + A\bar{B}C_{in} + AB + ABC_{in}$$

$$= BC_{in}\,(\bar{A} + A) + A\bar{B}C_{in} + AB = BC_{in} + A\bar{B}C_{in} + AB$$

$$= BC_{in} + A\bar{B}C_{in} + AB\,(1 + C_{in}) = BC_{in} + A\bar{B}C_{in} + AB + ABC_{in}$$

$$= BC_{in} + AB + AC_{in}\,(\bar{B} + B)$$

$$\therefore\ C_o = BC_{in} + AB + AC_{in}$$ **...Proved.**

- This expression is same as that for a full adder. Thus we have proved that circuit shown in Fig. 6.3.8 really behaves like a full adder.

6.3.5 Applications of Full Adder :

- The full adder acts as the basic building block of the 4 bit/8 bit binary/BCD adder ICs such as 7483.

6.3.6 Binary Subtractors :

- The rules of binary subtraction are as follows :

$$0 - 0 = 0$$
$$0 - 1 = 1 \text{ with borrow } 1$$
$$1 - 0 = 1$$
$$1 - 1 = 0$$

- Note that in the second case (0 – 1) it is necessary to borrow a 1.

Types of binary subtractors :

The types of binary subtractors are :

1. Half subtractor 2. Full subtractor

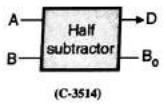
6.3.7 Half Subtractor :

- Half subtractor is a combinational circuit with two inputs and two outputs (difference and borrow).
- It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed.
- In the subtraction (A – B), A is called as **minuend bit** and B is called as **subtrahend bit.**

Truth table :

- Truth table showing the outputs of a half subtractor for all the possible combinations of input are shown in Table 6.3.1.

Scanned by CamScanner



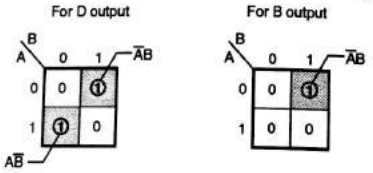
(C-3514)

Table 6.3.1 : Truth table for half subtractor

Inputs		Outputs	
A	B	Difference D (A – B)	Borrow B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-maps for difference and borrow outputs :

The K-maps for the two outputs of a half subtractor are as shown in Fig. 6.3.9.



(a) K-map and simplification for difference output **(b) K-map and simplification for the borrow output**

(C-356) **Fig. 6.3.9**

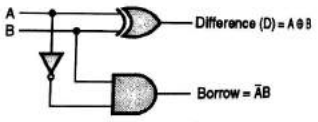
$\therefore$ Difference $D = \bar{A}B + A\bar{B} = A \oplus B$ $\therefore$ Borrow $B_o = \bar{A}B$

Logic diagram :

The logic diagram using these two Boolean expressions is as shown in Fig. 6.3.9(c).

Disadvantage of the half subtractor :

Half subtractor can only perform the subtraction of two binary bits. But while performing the subtraction, it does not take into account the borrow of the lower significant stage.



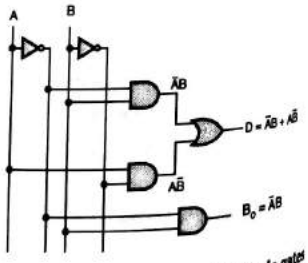
(C-357) **Fig. 6.3.9(c) : Half subtractor circuit**

Half subtractor using basic gates :

The expressions for the difference and borrow outputs are :

$$D = \bar{A}B + A\bar{B} \quad \text{and} \quad B_o = \bar{A}B$$

The half subtractor using basic gates is as shown in Fig. 6.3.10.



(C-358) **Fig. 6.3.10 : Half subtractor using basic gates**

Half subtractor using NAND gates :

- The expression for difference output is,

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

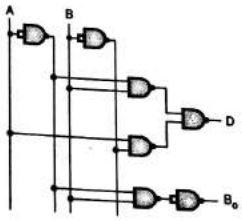
$$D = \overline{\overline{\bar{A}B + A\bar{B}}} \quad \text{...Double inversion}$$

$$D = \overline{\overline{\bar{A}B} \cdot \overline{A\bar{B}}} \quad \text{...(6.3.6)}$$

Similarly the other output B_o is given by,

$$B_o = \bar{A}B = \overline{\overline{\bar{A}B}} \quad \text{...(6.3.7)}$$

- Equations (6.3.6) and (6.3.7) can be implemented as shown in Fig. 6.3.11.



(C-359) **Fig. 6.3.11 : Half subtractor using NAND gates**

6.3.8 Full Subtractor :

MU : May 12

University Questions

Q. 1 Design a full subtractor and implement using basic logic gates. **(May 12, 10 Marks)**

- The disadvantage of a half subtractor is overcome if we use the full subtractor.
- The full subtractor is a combinational circuit with three inputs A, B and B_{in} and two outputs D and B_o.
- A is the minuend, B is subtrahend, B_{in} is the borrow produced by the previous stage, D is the difference output and B_o is the borrow output.

Truth table :

The truth table for full subtractor is shown in Table 6.3.2.

(C-3515)

Table 6.3.2 : Truth table for a full subtractor

Inputs			Outputs	
A (Minuend)	B (Subtrahend)	B_{in} Previous borrow	$(A - B - B_{in})$	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-maps and simplifications :

K-maps for D and B_o outputs are shown in Figs. 6.3.12(a) and (b).

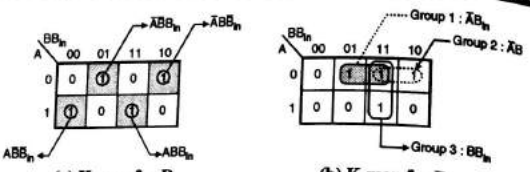
For difference output

$$\therefore D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$$

For borrow output

$$\therefore B_o = \bar{A}B_{in} + \bar{A}B + BB_{in}$$

Scanned by CamScanner



(a) K-map for D (b) K-map for B_o

(C-360) **Fig. 6.3.12**

Simplification for difference output :

From Fig. 6.3.12(a),

$$D = \bar{A}\,\bar{B}\,B_{in} + \bar{A}\,B\,\bar{B}_{in} + A\,\bar{B}\,\bar{B}_{in} + ABB_{in}$$

$$= B_{in}\underbrace{(\bar{A}\,\bar{B} + AB)}_{\text{EX-NOR}} + \bar{B}_{in}\underbrace{(\bar{A}B + A\bar{B})}_{\text{EX-OR}}$$

$$\therefore\ D = B_{in}\,\overline{(A \oplus B)} + \bar{B}_{in}\,(A \oplus B)$$

Let $A \oplus B = C$,

$$\therefore\ D = B_{in}\,\bar{C} + \bar{B}_{in}C = B_{in} \oplus C$$

$$\therefore\ D = B_{in} \oplus A \oplus B \qquad \text{...(6.3.8)}$$

Simplification for borrow output :

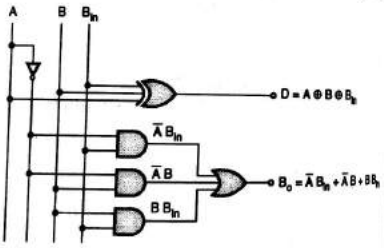
From Fig. 6.3.12(b),

$$B_o = \bar{A}B_{in} + \bar{A}B + BB_{in} \qquad \text{...(6.3.9)}$$

No further simplification is possible.

Logic diagram for full subtractor :

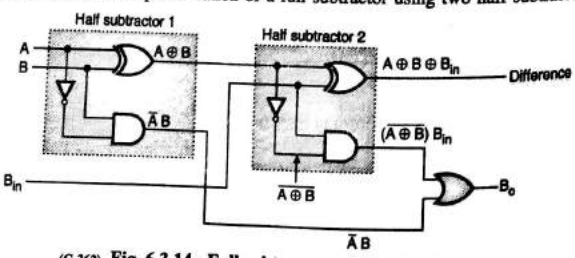
Logic diagram for the full subtractor is shown in Fig. 6.3.13. This has been drawn by using the Boolean equations of (6.3.8) and (6.3.9).



(C-361) **Fig. 6.3.13 : Logic diagram for a full subtractor**

6.3.9 Full Subtractor using Half Subtractors :

- Fig. 6.3.14 shows the implementation of a full subtractor using two half subtractors and an OR gate.



(C-362) **Fig. 6.3.14 : Full subtractor using half subtractors**

- Let us prove that the circuit shown in Fig. 6.3.14 really operates as a full subtractor.

Proof :

- Refer Fig. 6.3.13 to write the expression for difference output D as,

$$D = (A \oplus B) \oplus B_{in} = A \oplus B \oplus B_{in}$$

This is same as the expression for D output of a full subtractor.

- Now write the expression for Borrow output B_{out}.

$$B_{out} = (\overline{A \oplus B})\,B_{in} + \bar{A}B = (\overline{\bar{A}B + A\bar{B}})\,B_{in} + \bar{A}B$$

$$= (\bar{A}\,\bar{B} + AB)\,B_{in} + \bar{A}B = \bar{A}\,\bar{B}\,B_{in} + ABB_{in} + \bar{A}B$$

$$= \bar{A}\,\bar{B}\,B_{in} + ABB_{in} + \bar{A}B\,(1 + B_{in}) \qquad \text{...since } (1 + B_{in}) = 1$$

$$= \bar{A}\,\bar{B}\,B_{in} + ABB_{in} + \bar{A}B + \bar{A}B\,B_{in} = \bar{A}\,\bar{B}\,B_{in} + BB_{in}\,(A + \bar{A}) + \bar{A}B$$

$$= \bar{A}\,\bar{B}\,B_{in} + B\,B_{in} + \bar{A}B \qquad \text{...since } (A + \bar{A}) = 1$$

$$= \bar{A}\,\bar{B}\,B_{in} + B\,B_{in} + \bar{A}B\,(1 + B_{in}) = \bar{A}\,\bar{B}\,B_{in} + B\,B_{in} + \bar{A}B + \bar{A}B\,B_{in}$$

$$= \bar{A}B_{in}\,(\bar{B} + B) + BB_{in} + \bar{A}B = \bar{A}B_{in} + BB_{in} + \bar{A}B \qquad \textbf{...Proved.}$$

- Note that this expression is exactly same as that for B_o of the full subtractor.
- Thus the circuit shown in Fig. 6.3.14 acts as a full subtractor.

Ex. 6.3.1 : Design a full subtractor using only NAND gates. **May 09. 4 Marks**

Soln. :

Step 1 : Write the truth table for full subtractor :

The truth table for full subtractor is shown in Table P. 6.3.1.

Table P. 6.3.1 : Truth table for a full subtractor

Inputs			Outputs	
A (Minuend)	B (Subtrahend)	B_{in} Previous borrow	$D = A - B - B_{in}$	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$\therefore D = \bar{A}\,\bar{B}\,B_{in} + \bar{A}B\,\bar{B}_{in} + A\,\bar{B}\,\bar{B}_{in} + A\,B\,B_{in} \qquad \text{...(1)}$$

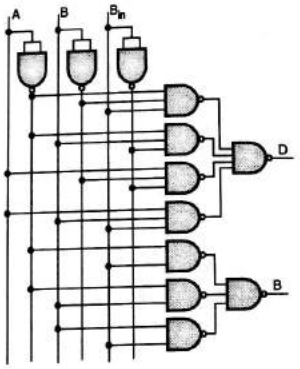
$$B = \bar{A}B_{in} + \bar{A}B + B\,B_{in} \qquad \text{...(2)}$$

Scanned by CamScanner

$$D = \overline{(\overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B\,\overline{B}_{in} + A\,\overline{B}\,\overline{B}_{in} + A\,B\,B_{in})} = \overline{(\overline{A}\,\overline{B}\,B_{in})} \cdot \overline{(\overline{A}B\,\overline{B}_{in})} \cdot \overline{(A\,\overline{B}\,\overline{B}_{in})} \cdot \overline{A\,B\,B_{in}}$$

$$B = \overline{\overline{AB_{in} + \overline{A}\,B + BB_{in}}} = \overline{\overline{A}\,B_{in} \cdot \overline{AB} \cdot B\,B_{in}}$$

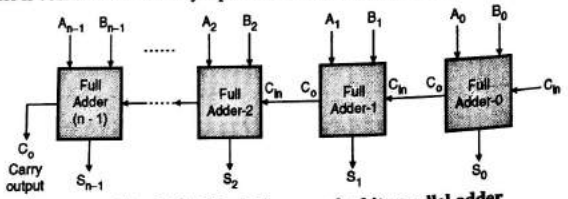
Step 2 : Implementation using NAND gates :



(C-2200) **Fig. P. 6.3.1**

6.4 The n-Bit Parallel Adder :

- The full adder is capable of adding only two single digit binary numbers along with a carry input.
- But in practice we need to add binary numbers which are much larger in size than just one bit. The two binary numbers to be added could be 4 bit, 8 bit, 16 bit long. In general we assume that both the numbers are n bit long.
- To add two n-bit binary numbers we need to use the n-bit parallel adder shown in Fig. 6.4.1.
- It uses a number of full adders which are connected in cascade. The carry output of the previous full adder is connected to the carry input of the next full adder as shown in Fig. 6.4.1.



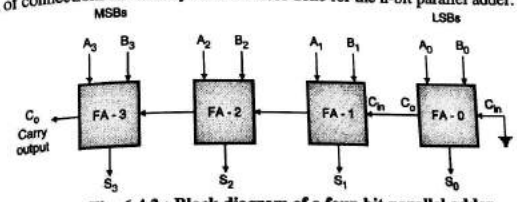
(C-385) **Fig. 6.4.1 : Block diagram of n-bit parallel adder**

6.4.1 A Four Bit Parallel Adder Using Full Adders :

- The block diagram of a four bit parallel adder using full adders is shown in Fig. 6.4.2. Let the two four bit words that are to be added be A and B and be denoted as follows :

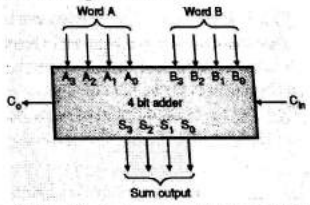
$$A = A_3\,A_2\,A_1\,A_0, \qquad B = B_3\,B_2\,B_1\,B_0.$$

- A_0 and B_0 represent the LSBs of the four bit words A and B. Hence Full Adder – 0 is the lowest stage. Hence its C_{in} has been connected to 0 permanently.
- The rest of connections are exactly same as those done for the n-bit parallel adder.



(C-386) **Fig. 6.4.2 : Block diagram of a four-bit parallel adder**

- The four-bit parallel adder is a very common logic circuit. It is normally shown by a block diagram as shown in Fig. 6.4.3. It has two 4 bit inputs A_3....A_0 and B_3 .....B_0, a carry input and carry output and 4-bit sum output $S_3\,S_2\,S_1\,S_0$.



(C-387) **Fig. 6.4.3 : Block diagram of 4-bit parallel adder**

6.4.2 Propagation Delay in Parallel Adder :

- In parallel adder carry out of the previous stage is connected to carry in of the next stage. Therefore the carry is said to be propagated like ripple from the LSB stage to the MSB stage. This phenomenon is called **ripple carry propagation.**
- Due to this ripple carry propagation **time delay** is introduced in the addition process. This time delay is called as the propagation delay.
- Consider the addition of the two 4-bit numbers.

$$\begin{array}{rcccc} & 0 & 1 & 1 & 0 \\ + & 0 & 0 & 1 & 1 \\ \text{carry} & 1 & 1 & & \\ \hline & 1 & 0 & 0 & 1 \end{array}$$

- From this addition it is evident that the addition of 1's in the second position produces a carry which is added to the bits in the third position and the carry produced in the third position is added to the bits in the fourth position.
- Therefore the sum bit produced in the MSB position of the result depends on the carry generated due to additions in the preceding stages. The real problem is created due to this.
- If the propagation delay of each full adder is say 20 nS, then sum S_3 will reach its correct value after $3 \times 20 = 60$ nS from the instant when LSB carry is generated.
- Hence the total time required to perform the addition will be $4 \times 20 = 80$ nS.

Scanned by CamScanner

- The problem of propagation delay becomes severe as the number of bits increases. For example if n = 16 then the total time required for performing the addition is $16 \times 20 = 320$ nS.

6.4.3 Look Ahead – Carry Adder :

MU : Dec. 02, Dec. 03, Dec. 05, Dec. 06

University Questions

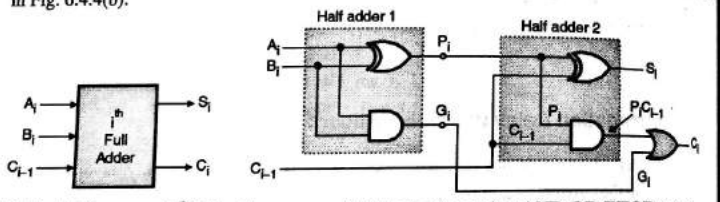
Q. 1 Write short note on : Carry look ahead carry generator. **(Dec. 02, 4 Marks)**

Q. 2 What is a carry look ahead adder ? Design a 4-bit carry look ahead adder using gates. **(Dec. 03, 5 Marks)**

Q. 3 What is a carry look ahead adder ? Design a 4-bit carry look ahead adder using gates. **(Dec. 05, 6 Marks)**

Q. 4 What is a carry look ahead adder design 4 bit carry look ahead adder using gates ? **(Dec. 06, 5 Marks)**

- The problem of propagation delay can be eliminated by using this addition technique.
- The look-ahead-carry addition will therefore speed up the addition process.
- The adder with look ahead carry needs additional hardware but the speed of that adder is independent of the number of bits.
- Consider the block diagram of full adder Fig. 6.4.4(a) and its AND-OR-EXOR realization shown in Fig. 6.4.4(b).



(a) Block diagram of ith full adder **(b) Realization using AND-OR-EXOR gates**

(C-389) **Fig. 6.4.4**

Refer Fig. 6.4.4(b) to write,

$$P_i = A_i \oplus B_i \quad ...(6.4.1)$$

$$\text{and } G_i = A_i B_i \quad ...(6.4.2)$$

$$\text{Also } S_i = P_i \oplus C_{i-1} = A_i \oplus B_i \oplus C_{i-1} \quad ...(6.4.3)$$

$$\text{and } C_i = G_i + P_i C_{i-1} \quad ...(6.4.4)$$

The carry output G_i of the first half adder is equal to 1 if $A_i = B_i = 1$ and a carry is generated at the ith stage of the parallel adder. That means $C_i = 1$.

This variable G_i is known as **Carry Generate** and its value does not depend on the input carry i.e. C_{i-1}.

The variable P_i is called as **Carry Propagate** because this term is associated with the propagation of carry from C_{i-1} to C_i. Now consider Equation (6.4.4) i.e. $C_i = G_i + P_i C_{i-1}$.

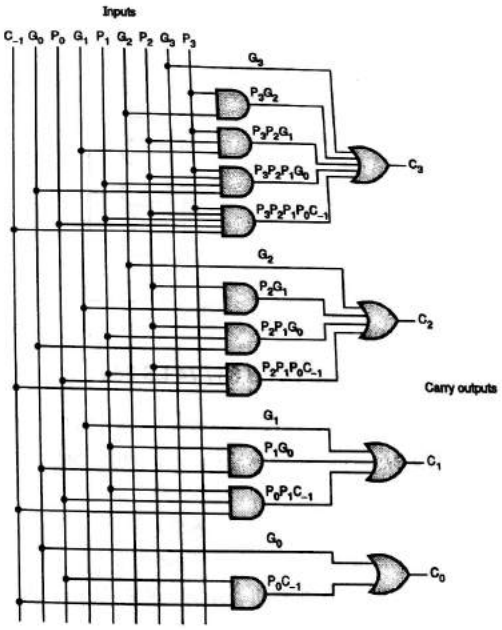
Using this equation, we can write the expression for the carry output of each stage in a 4 bit parallel adder as follows :

$$P_i = A_i \oplus B_i$$

Stage	Expression for carry output	
0	$C_0 = G_0 + P_0 C_{-1}$	...(6.4.5)
1	$C_1 = G_1 + P_1 C_0$	
	$= G_1 + P_1 (G_0 + P_0 C_{-1})$	
	$\therefore C_1 = G_1 + P_1 G_0 + P_0 P_1 C_{-1}$	...(6.4.6)

Stage	Expression for carry output	
2	$C_2 = G_2 + P_2 C_1$	
	$= G_2 + P_2 (G_1 + P_1 G_0 + P_0 P_1 C_{-1})$	
	$\therefore C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1}$	...(6.4.7)
3	$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}$	...(6.4.8)

- In the expressions stated above, the variable involved are $G_0, G_1, G_2, G_3, P_0, P_1, P_2, P_3$ and C_{-1}.
- Out of them the G variables are generated from the A and B inputs using AND gates (as illustrated in Equation (6.4.2).
- And the P variables are obtained again directly from A and B inputs using EX-OR gates (as illustrated in Equation (6.4.1).
- If the G, P and C_{-1} are at a time available, then it is possible to produce the carry outputs C_0, C_1, C_2 and C_3 by using 2-level realization (AND-OR or NAND-NAND) etc.
- The advantage of generating the carry outputs using this method is that the propagation delay in this process corresponds to the propagation delay of only two gates.



(C-390) **Fig. 6.4.5 : Logic diagram of the look ahead carry generator**

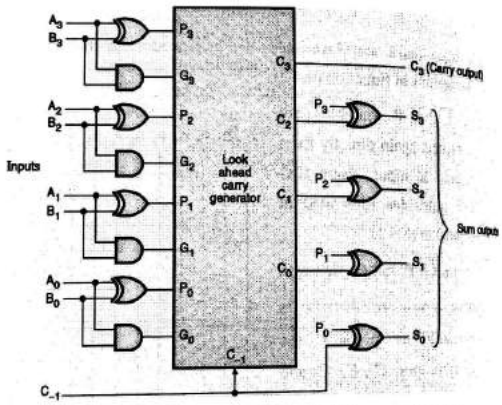
- These carry outputs are then connected to the carry inputs of the succeeding stages. This eliminates the problem of carry getting propagated like ripples.

Scanned by CamScanner

- The logic circuit of a look ahead carry generator is shown in Fig. 6.4.5 and it is based on Equations (6.4.1) through (6.4.8).

6.4.4 Four Bit Fast Adder with Look-Ahead Carry :

- The block diagram of a four bit parallel adder using the look-ahead carry generator is shown in Fig. 6.4.6.
- The P_i and G_i variables (i.e. $P_0, P_1, \ldots, P_3$ and $G_0, G_1 \ldots G_3$) are obtained from the inputs A_i and B_i (i.e. $A_3\ A_2\ A_1\ A_0$ and $B_3\ B_2\ B_1\ B_0$) by using the EX-OR and AND gates respectively.
- The carry outputs C_0 through C_3 are produced simultaneously by the look ahead carry generator, as explained earlier.
- These carry outputs and P_i variables are EX-ORed to produce the sum outputs S_0 through S_3. For example S_3 is produced by Ex-ORing P_3 and C_2, then S_2 is produced by Ex-ORing P_2 and C_1 and so on.



(C-391) **Fig. 6.4.6 : A 4-bit parallel adder**

6.4.5 Fast Adder IC 74 LS 83 / 74 LS 283 :

- The most common binary parallel adder in the integrated circuit form is IC 74 LS 83 / 74 LS 283.
- It is a 4-Bit parallel adder, which consists of four interconnected full adders alongwith the look-ahead carry circuit.
- The IC 7483 and 74283 are TTL MSI (medium scale integration) for 4-bit parallel adders and both of them have the same pin configuration.
- Fig. 6.4.7 shows the functional symbol of IC 74 LS 283. $A_3\ A_2\ A_1\ A_0$ is a four bit word A and $B_3\ B_2\ B_1\ B_0$ is another word B. Both these words are applied at the inputs of the I.C. 7483/74 LS 283.
- $C_{in\ 0}$ is the input carry and $C_{out\ 3}$ represents the output carry. S_3, S_2, S_1, S_0 represent the sum outputs with S_3 MSB.

Pin diagram :

- The pin diagram of IC 7483 is shown in Fig. 6.4.8.
- This IC adds the two four bit words A and B and the bit at $C_{in\ 0}$ and produces a four bit sum output alongwith carry output at $C_{out\ 3}$.

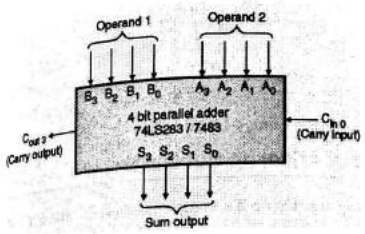
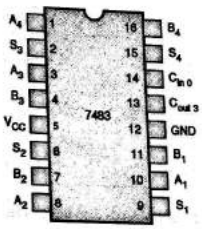


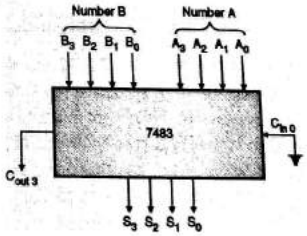
Fig. 6.4.7 : **Functional symbol for 74 LS 283**



(C-392) **Fig. 6.4.8 : Pin diagram of IC – 7483**

6.4.6 Four Bit Binary Adder using IC 7483 :

- A four bit binary adder is shown in Fig. 6.4.9. Note that the carry input C_{in0} has been connected to ground. So at the outputs we get the addition of two four bit numbers A and B.

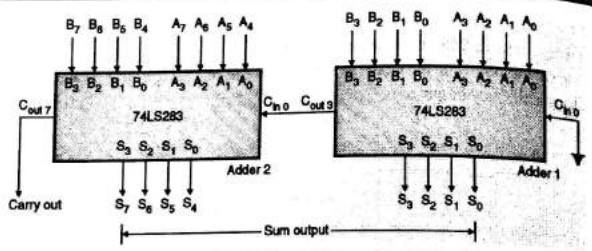


(C-393) **Fig. 6.4.9 : 4 bit binary adder using IC 7483**

6.4.7 Cascading of Adders :

- If we want to add two 8-bit numbers using the 4-bit parallel adder 74283, then we have to cascade two such four bit adders.
- Fig. 6.4.10 shows an 8-bit adder using two 4-bit adders. Similarly it is possible to connect a number of adders to make an n-bit adder.
- $A_7 - A_0$ and $B_7 - B_0$ are the two eight bit numbers to be added.
- Adder-1 in Fig. 6.4.10 adds the four LSB bits of the two numbers i.e. $A_3 - A_0$ and $B_3 - B_0$. The carry input of first adder has been connected to ground (logic 0).
- The $C_{out\ 3}$ i.e. the carry output of adder-1 is connected to $C_{in\ 0}$ input of adder-2. The second adder adds this carry and the four MSB bits of the two numbers.
- $C_{out\ 7}$ of adder-2 acts as the final output carry and the sum output is from S_7 through S_0.

Scanned by CamScanner



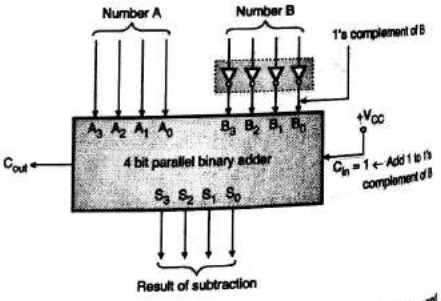
(C-394) **Fig. 6.4.10 : 8-bit addition using 4-bit adders**

6.5 n-bit Parallel Subtractor (Use of Adder as Subtractor) :

- The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted.
- For example we can perform the subtraction (A – B) by adding either 1's complement or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

6.5.1 4 Bit Parallel Subtractor using 2's Complement :

- A 4-bit parallel subtractor using a 4-bit parallel adder is shown in Fig. 6.5.1.
- The number to be subtracted (B) is first passed through inverters to obtain its 1's complement. One inverter per bit of word B is used so that all the bits of B get inverted.
- Then 1 is added to 1's complement of B, by making $C_{in} = 1$. Thus we obtain the 2's complement of B.
- The 4-bit adder then adds A and 2's complement of B to produce the subtraction at its sum outputs $S_3\ S_2\ S_1\ S_0$.
- The word $S_3\ S_2\ S_1\ S_0$ represent the result of binary subtraction (A – B) and carry output C_{out} represents the polarity of the result.
- If A > B then $C_{out} = 0$ and the result is in true binary form but if A < B then $C_{out} = 1$ and the result is negative and in the 2's complement form.



(C-395) **Fig. 6.5.1 : 4-bit parallel binary subtractor using 2's complement**

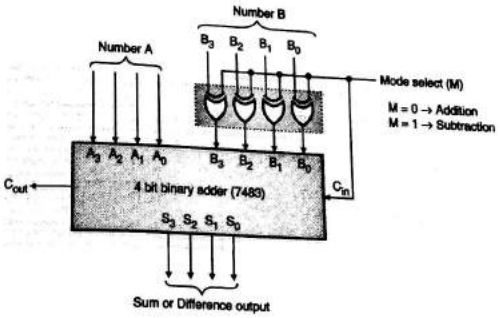
6.5.2 4-Bit Binary Parallel Adder/Subtractor (Using 2's Complement) :

MU - May 03

University Questions

Q. 1 Implement a mode controlled full adder/full subtractor using IC 7483's and required gates. **(May 03, 12 Marks)**

- The addition or subtraction of two 4-bit binary numbers can be obtained using the same circuit shown in Fig. 6.5.2. Therefore it is called as an adder/subtractor circuit.
- The operation performed by this circuit (Addition or subtraction) depends on the state of the mode of select input i.e. M in Fig. 6.5.2.



(C-396) **Fig. 6.5.2 : 4-bit binary parallel adder/subtractor**

- Number B is applied to the adder through four EX-OR gates. One input of each EX-OR gate is connected to the mode-select input (M).

Operation as Adder (M = 0) :

- For operation as an adder, the mode select (M) input is connected to ground. ∴ M = 0.
- Since M = 0, the output of the EX-OR gates will be the same number B which was applied at their inputs. This is because $0 \oplus 0 = 0$ and $0 \oplus 1 = 1$. Hence B_3, B_2, B_1 and B_0 will pass unchanged through the EX-OR gates. The carry input C_{in} is connected to M.

$$\therefore C_{in} = 0$$

- Therefore the adder adds $A + B + C_{in} = A + B$ since $C_{in} = 0$. Thus with M = 0 addition of A and B will take place.

Operation as Subtractor (M = 1) :

- For operation as a subtractor, the mode select (M) input is connected to V_{CC}. ∴ M = 1.
- Since M = 1, one input of each EX-OR gate is now 1. Hence each EX-OR gate acts as an inverter. This is because $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$.
- Thus each bit of word B is inverted by the EX-OR inverters. Thus we get the 1's complement of number B at the output of EX-OR gates. The carry input terminal C_{in} is connected to M.

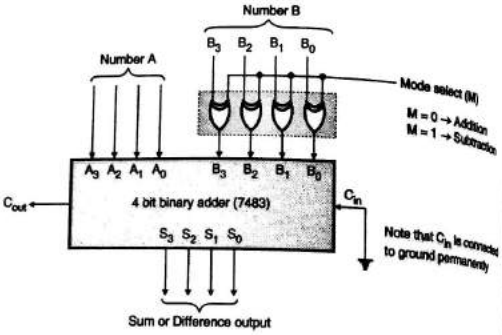
$$\therefore C_{in} = 1$$

- The inverted number B adds with $C_{in} = 1$ to give the 2's complement of B. Hence the adder will add A with the 2's complement of B and the result is actually the subtraction A – B. If $C_{out} = 0$ then the subtraction (A – B) is positive and in true form. But if $C_{out} = 1$ then the subtraction (A – B) is negative and in the 2's complement form.
- Thus with M = 1, this circuit works as a 2's complement subtractor.

Scanned by CamScanner

6.5.3 4-Bit Binary Parallel Adder/Subtractor (Using 1's Complement) :

- The addition or subtraction of two 4-bit binary numbers can be obtained using the same circuit shown in Fig. 6.5.3.
- The operation performed by this circuit (Addition or subtraction) depends on the state of the mode of select input.



(C-396(a)) **Fig. 6.5.3 : 4-bit binary parallel adder/subtractor (1's complement)**

- Number B is applied to the adder through a set of EX-OR gates. One input of each EX-OR gate is connected to the mode-select input (M). Also C_{in} is connected to GND permanently. (In previous circuit C_{in} was connected to 0 for addition and 1 for subtraction.)

Operation as Adder (M = 0) :

- The mode select (M) input is connected to ground. $\therefore$ M = 0.
- Since M = 0, the output of the EX-OR gates will be the same number B applied at their inputs. This is because $0 \oplus 0 = 0$ and $0 \oplus 1 = 1$. Hence B_3, B_2, B_1 and B_0 will pass unchanged through the EX-OR gates. C_{in} is connected to ground.

$$\therefore \; C_{in} = 0$$

- The adder then adds $A + B + C_{in} = A + B$ since $C_{in} = 0$. Thus with M = 0 addition will take place.

Operation as Subtractor (M = 1) :

- The mode select (M) input is connected to V_{CC}. $\therefore$ M = 1.
- Since M = 1, one input of each EX-OR gate is now 1. Hence each EX-OR gate acts as an inverter. This is because $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$.
- Hence each bit of word B is inverted by the EX-OR inverters. Thus we get the 1's complement of number B at the output of EX-OR gates. The carry in bit $C_{in} = 0$.
- The inverted number B corresponds to the 1's complement of B. Hence the adder will add A with the 1's complement of B and the result is actually the subtraction A – B.
- Thus with M = 1, this circuit works as a 1's complement subtractor.

6.6 BCD Adder :

MU : Dec. 03, May 04, May 07, May 08, Dec. 11, Dec. 14

University Questions

Q. 1 Design an adder circuit to add two BCD numbers using IC 7483 and necessary gates. **(Dec. 03, 7 Marks)**

Q. 2 Design a 4 bit BCD adder using IC 7483. **(May 04, 10 Marks)**

Q. 3 Design one bit BCD adder using IC 7483. **(May 07, May 08, 10 Marks)**

Q. 4 Design a BCD adder using 4-bit binary adders and explain. **(Dec. 11, 10 Marks)**

Q. 5 Explain with example 4 bit BCD adder using IC-7483. **(Dec. 14, 8 Marks)**

- BCD adder adds two BCD digits and produces a BCD digit. But remember that a BCD digit cannot be greater than 9.
- The two given 4-bit BCD numbers are to be added using the rules of binary addition.
- If sum is less than or equal to 9 and carry = 0, then no correction is necessary. The sum obtained is correct and in the true BCD form.
- But if sum is invalid BCD or carry = 1, then the result is wrong and needs correction.
- The wrong result can be corrected by adding six i.e. (0110) to it.

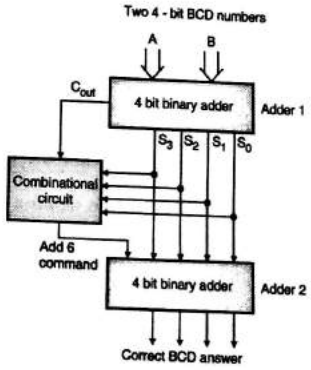
6.6.1 Block Diagram of BCD Adder :

MU : Dec. 15

University Questions

Q. 1 Implement single digit BCD adder using 4-bit binary adder IC 7483. Show the design procedure and explain its operation. **(Dec. 15, 10 Marks)**

- From the points stated above, we understand that the 4-bit, BCD adder should consist of the following blocks :
  1. A 4-bit binary adder to add the given two 4-bit BCD numbers A and B.
  2. A combinational circuit to check if sum is greater than 9 or carry = 1.
  3. Another 4-bit binary adder to add six (0110) to the incorrect sum if sum > 9 or carry = 1.
- The block diagram of such a BCD adder is shown in Fig. 6.6.1.
- So we have to design the combinational circuit that finds out whether the sum is greater than 9 or Carry = 1.



(C-397) **Fig. 6.6.1 : Block diagram of BCD adder**

6.6.2 Design of Combinational Circuit :

MU : May 09, Dec. 15

University Questions

Q. 1 Design a BCD adder using 4-bit binary adders and explain. **(May 09, 10 Marks)**

Q. 2 Implement single digit BCD adder using 4-bit binary adder IC 7483. Show the design procedure and explain its operation. **(Dec. 15, 10 Marks)**

- The output of combinational circuit should be 1 if the sum produced by adder 1 is greater than 9 i.e. 1001. The truth table is as follows :

Scanned by CamScanner

Table 6.6.1 : Truth table for combinational circuit design

Sum bits of adder-1 →

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Sum is valid BCD number. ∴ Y = 0

Sum is invalid BCD number. ∴ Y = 1

Write K-map :

For Y output

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Group 2 : S_3S_1

Group 1 : S_3S_2

(C-398) **Fig. 6.6.2 : K-map for Y output**

- The Boolean expression is,

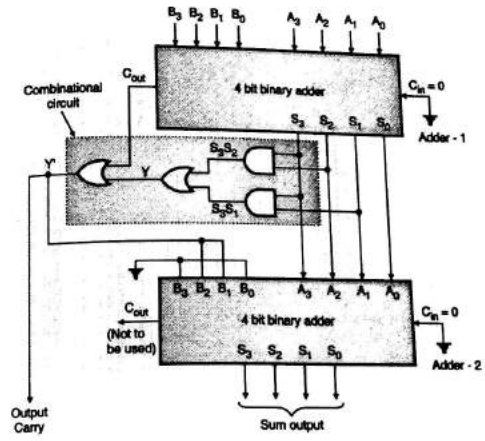
$$Y = S_3S_2 + S_3S_1$$

- The complete BCD adder is shown in Fig. 6.6.3.
- The output of the combinational circuit should be 1 if C_{out} of adder-1 is high or if the output of adder-1 is greater than 9. Therefore Y is ORed with C_{out} of adder 1 as shown in Fig. 6.6.3.
- The output of combinational circuit is connected to B_1B_2 inputs of adder-2 and $B_3 = B_1 = 0$ as they are connected to ground permanently. This makes $B_3\,B_2\,B_1\,B_0 = 0\,1\,1\,0$ if $Y' = 1$.
- The sum outputs of adder-1 are applied to $A_3A_2A_1A_0$ of adder-2. The output of combinational circuit is to be used as final output carry and the carry output of adder-2 is to be ignored.

Operation :

Case I : Sum ≤ 9 and carry = 0

- The output of combinational circuit $Y' = 0$. Hence $B_3\,B_2\,B_1\,B_0 = 0000$ for adder-2.
- Hence output of adder-2 is same as that of adder-1.



(C-399) **Fig. 6.6.3 : 4-bit BCD adder**

Case II : Sum > 9 and carry = 0

- If $S_3\,S_2\,S_1\,S_0$ of adder-1 is greater than 9, then output Y′ of combinational circuit becomes 1.
- $\therefore B_3\,B_2\,B_1\,B_0 = 0110$ (of adder-2)
- Hence six (0110) will be added to the sum output of adder-1.
- We get the corrected BCD result at the sum output of adder-2.

Case III : Sum ≤ 9 but carry = 1

- As carry output of adder-1 is high, $Y' = 1$.
- $\therefore B_3\,B_2\,B_1\,B_0 = 0\,1\,1\,0$ (of adder-2)
- $\therefore$ 0 1 1 0 will be added to the sum output of adder-1.
- We get the corrected BCD result at the sum output of adder-2.

Thus the four bit BCD addition can be carried out using the binary adder.

6.6.3 8-Bit BCD Adder using IC 74283 :

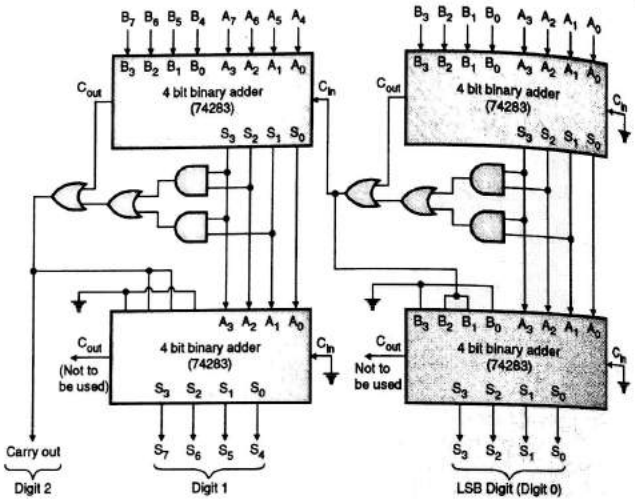
MU : May 15, May 16

University Questions

Q.1 Design 8 bit BCD adder. **(May 15, 8 Marks, May 16, 10 Marks)**

- The 8-bit BCD adder using the 4-bit BCD adder 74283 is shown in Fig. 6.6.4.

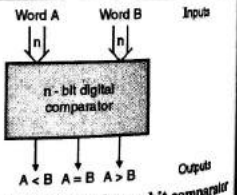
Scanned by CamScanner



(C-400) **Fig. 6.6.4 : 8-bit BCD adder using 74283**

6.7 Magnitude Comparators :

- The block diagram of an n bit digital comparator is shown in Fig. 6.7.1.
- Digital comparator is a combinational circuit, which compares the two n-bit binary words applied at its input.
- The comparator has three outputs namely A > B, A = B and A < B. Depending on the result of comparison, one of these outputs will go high.



(C-403) **Fig. 6.7.1 : Block diagram of an n-bit comparator**

6.7.1 1-Bit Binary Comparator :

MU : May 15

University Questions

Q. 1 Design 1 bit comparator using logic gates. **(May 15, 5 Marks)**

Truth table of one-bit comparator :

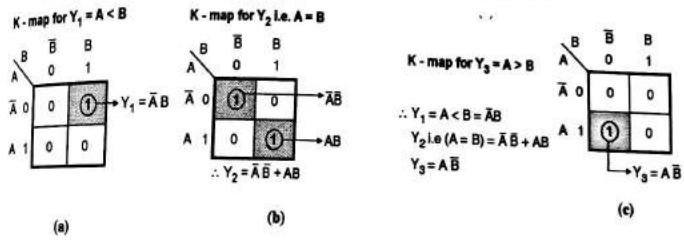
- The one-bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A < B, A = B and A > B.
- A one bit comparator compares the two single bit numbers A and B and produces an output that indicates the result of the comparison. This is clear from the truth table as given in Table 6.7.1.

Table 6.7.1 : Truth table of a one-bit comparator

Inputs		Outputs		
A	B	$Y_1 = A < B$	$Y_2 = A = B$	$Y_3 = A > B$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

K-maps for each output :

- The K-maps for the three outputs Y_1, Y_2 and Y_3 are as shown in Fig. 6.7.2.



(C-404) **Fig. 6.7.2 : K-maps for the three outputs of a one-bit comparator**

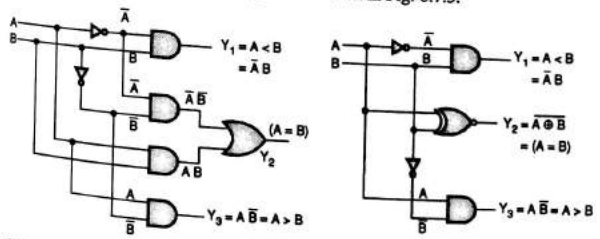
- From Fig. 6.7.2 we can write the expressions for the three outputs as,

$$Y_1 = (A < B) = \bar{A}B,$$

$$Y_2 = (A = B) = \bar{A}\bar{B} + AB = \overline{A \oplus B}$$

$$Y_3 = A\bar{B}$$

- The expression for Y_2 is nothing but the expression for an EX-NOR gate. Hence the single bit comparator can be realized using basic gates as shown in Fig. 6.7.3.



(a) Using basic gates (b) Using AND and EX-NOR gates

(C-405) **Fig. 6.7.3 : Realization of one bit comparator**

Scanned by CamScanner

7.2 A 2-Bit Comparator :

MU : May 03, May 04, Dec. 09, May 12, May 13

niversity Questions

1 Write short note on : 2 - bit magnitude comparator **(May 03, 8 Marks)**

2 Design a 2 bit magnitude comparator. Which is the IC used for same ? **(May 04, 10 Marks)**

3 Design a 2-bit digital comparator that accepts inputs A and B and gives three outputs G, E and L.

1. Output G is active, when A > B.
2. Output E is active, when A = B.
3. Output L is active, when A < B. **(Dec. 09, May 14, 10 Marks)**

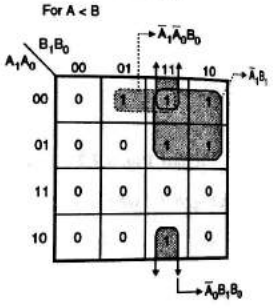
4 Design a 2 bit digital magnitude comparator and implement using NAND gates only. **(May 12, 10 Marks)**

For a 2-bit comparator, each input word A and B is 2 bit long.
The truth table of a 2-bit comparator is shown in Table 6.7.2.

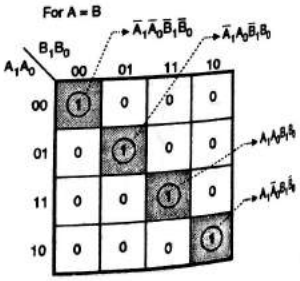
Table 6.7.2 : Truth table for a 2-bit comparator

Inputs				Outputs		
A_1	A_0	B_1	B_0	A < B	A = B	A > B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

K-maps :



(a) K-map for output A < B



(b) K-map for output A = B

(C-406) **Fig. 6.7.4**

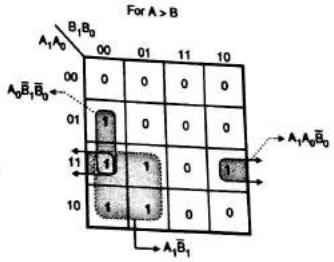
- The K-maps for the three outputs and corresponding simplified expressions are shown in Figs. 6.7.4(a), (b) and (c). From these K-maps we get the simplified expressions for the three outputs of comparator are as follows :

$$A < B = \bar{A}_1\bar{A}_0B_0 + \bar{A}_1B_1 + \bar{A}_0B_1B_0$$

For A > B

Simplified expression :

$$A > B = A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0 + A_1\bar{B}_1$$



(C-407) **Fig. 6.7.4(c) : K-map for output A > B**

Simplification for output A = B :

Refer the K-map for output A = B shown in Fig. 6.7.4(b).
The expression for A = B is given by,

$$(A = B) = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$$

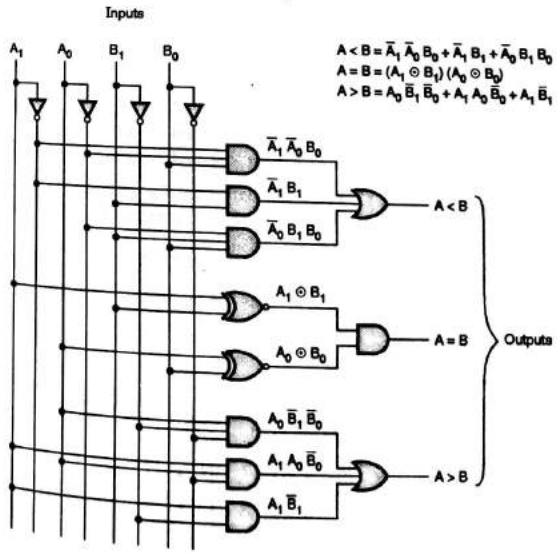
$$= \bar{A}_0\bar{B}_0(\bar{A}_1\bar{B}_1 + A_1B_1) + A_0B_0(\bar{A}_1\bar{B}_1 + A_1B_1)$$

$$= (\bar{A}_1\bar{B}_1 + A_1B_1)(\bar{A}_0\bar{B}_0 + A_0B_0)$$

$$\therefore (A = B) = (A_1 \odot B_1)(A_0 \odot B_0) \quad \text{where } \odot = \text{EX-NOR}$$

Logic diagram for a two bit comparator :

The logic diagram for the 2-bit digital comparator is shown in Fig. 6.7.5. This diagram is drawn by referring to the simplified expressions of the outputs.



(C-408) **Fig. 6.7.5 : Logic diagram for 2-bit comparator**

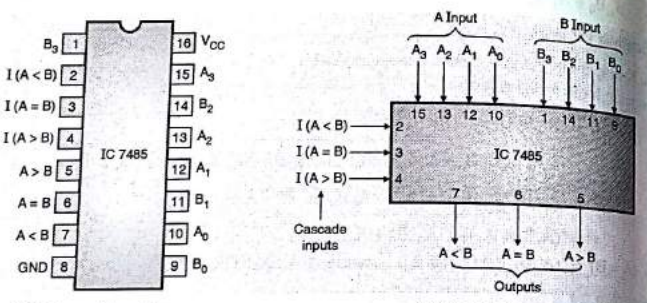
Scanned by CamScanner

6.7.3 A 4-Bit Magnitude Comparator (IC 7485) :

University Questions MU : May 16

Q. 1 Write short note on :4-bit magnitude comparator **(May 16, 5 Marks)**

- IC 7485 is a four bit comparator in the integrated circuit form. It compares two 4-bit words A ($A_3 - A_0$) and B ($B_3 - B_0$).
- It is possible to cascade more than one IC 7485 to compare words of almost any length.
- Fig. 6.7.6 shows the pin configuration and logic symbol of IC 7485.



(a) Pin configuration (b) Logic diagram of IC 7485

(C-409) **Fig. 6.7.6**

Pin names and their functions :

The pin names and their functions are as listed in Table 6.7.3.

Table 6.7.3.

Pin name	Pin number	Function
A_0 to A_3	10, 12, 13, 15	Binary input (operand 1) Active high
B_0 to B_3	9, 11, 14, 1	Binary input (operand 2) Active high
I (A < B) I (A = B) I (A > B)	2 3 4	These lines are used for cascading a number of IC 7485 outputs of the previous stage are fed as inputs to this stage.
A < B A = B A > B	7 6 5	These are the outputs. When ICs 7485 are cascaded, these outputs are applied to cascading inputs of the next stage.

Truth table of IC 7485 :

Table 6.7.4

Comparing inputs				Cascading inputs			Outputs		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$Y_{A>B}$	$Y_{A<B}$	$Y_{A=B}$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L

Comparing inputs				Cascading inputs			Outputs		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$Y_{A>B}$	$Y_{A<B}$	$Y_{A=B}$
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	L	H	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	X	X	H	L	L	H
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	H	L	L	L	L

Function table of IC 7485 : Table 6.7.5 shows the function table for IC 7485.

Table 6.7.5

Comparing inputs	Cascading inputs			Outputs		
	$I_{A<B}$	$I_{A=B}$	$I_{A>B}$	$Y_{A<B}$	$Y_{A=B}$	$Y_{A>B}$
A < B	X	X	X	1	0	0
A = B	0	0	0	1	0	1
	0	0	1	0	0	1
	1	0	0	1	0	0
	1	0	1	0	0	0
	X	1	X	0	1	0
A > B	X	X	X	0	0	1

- The function table shown in Table 6.7.5 tells us that if I (A < B) or I (A > B) is equal to 1 and if the other two cascading inputs are at logic 0 then the corresponding output (A < B or A > B) will be active.
- But if A = B, then the corresponding output will be active if and only if I (A = B) is at logic 1 irrespective of the other two inputs.

General description :

- As shown in the truth table (Table 6.7.4); the MSB are compared first i.e. A_3 is compared with B_3.
- Depending on whether $A_3 > B_3$ or $A_3 < B_3$ the outputs Y (A > B) or Y (A < B) are activated.
- But if $A_3 = B_3$ then the next MSB bits B_2 and A_2 are compared. If $A_2 = B_2$ then comparison of A_1 and B_1 is performed and so on.
- If $A_3\ A_2\ A_1\ A_0 = B_3\ B_2\ B_1\ B_0$ then the IC 7485 will check the cascading inputs. The decision making will then take place as follows :

Case 1 : If I (A > B) = 1 and I (A = B) = I (A < B) = 0

This indicates that at the previous stage LSB of A is greater than LSB of B.

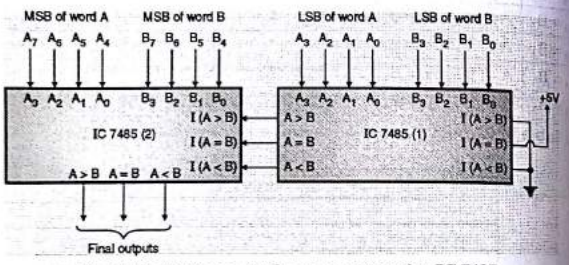
Case 2 : If I (A = B) = 1

If I (A = B) = 1 then the chip will not check the status of I (A < B) and I (A > B) inputs and will understand that the LSB of A is equal to LSB of B.

Scanned by CamScanner

6.7.4 Cascading of IC 7485 :

- IC 7485 is a four bit magnitude comparator. If we want to compare the words that are longer than 4 bit then the cascading is required to be done.
- An 8-bit comparator using IC 7485 is shown in Fig. 6.7.7.
- The outputs of IC 7485-1 are applied to the cascading inputs of IC 7485 – 2.
- The two 8-bit words to be compared are A ($A_7 - A_0$) and B ($B_7 - B_0$). These are divided in two 4-bit words each and then applied to the two comparator ICs.



(C-410) **Fig. 6.7.7 : 8-bit magnitude comparator using IC 7485**

Operation :

- The MSB bits A_7 and B_7 are compared first. If $A_7 = B_7$ then the next bits A_6 and B_6 are compared. This will continue upto A_0 and B_0.
- If $A_7 A_6 A_5 A_4 = B_7 B_6 B_5 B_4$ then IC 7485 (2) will check the cascading inputs.
- IC 7485(1) will continue the comparison from $A_3 - B_3$ to $A_0 - B_0$ as explained earlier. If $A_3 A_2 A_1 A_0 = B_3 B_2 B_1 B_0$ then IC 7485(1) will check its cascading inputs.

Important Note : Since IC 7485(1) itself is the LSB chip, we have to adjust the status of its cascade inputs so that they indicate that A = B. Hence I (A = B) should be connected to + 5 V i.e. logic 1.

Ex. 6.7.1 : Design 5 bit magnitude comparator using magnitude comparator chip (7485).

Soln. :

We know by now that output of a comparator is based on the data inputs as well as cascading inputs. Also, comparison always begins from MSB bit. As only four bits are available per variable we accommodate the fifth bit for A_0 and B_0 i.e. LSB's on $I_{A<B}$ and $I_{A>B}$ inputs. This leaves with $I_{A=B}$ whose connection can be found out based on function Table P. 6.7.1.

Table P. 6.7.1 : Function table of IC 7485

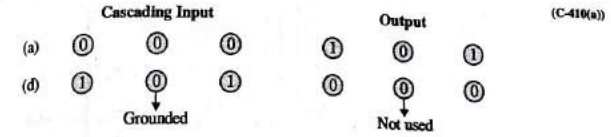
Comparing inputs		Cascading inputs			Outputs		
		$I_{A<B}$	$I_{A=B}$	$I_{A>B}$	$Y_{A<B}$	$Y_{A=B}$	$Y_{A>B}$
A = B	(a)	0	0	0	1	0	1
	(b)	0	0	1	0	0	1
	(c)	1	0	0	1	0	0
	(d)	1	0	1	0	0	0
	(e)	X	1	X	0	1	0

Condition (e) is ruled out as it always indicate $Y_{A=B} = 1$.

The other cases imposes a condition that $I_{A=B}$ must be ground.

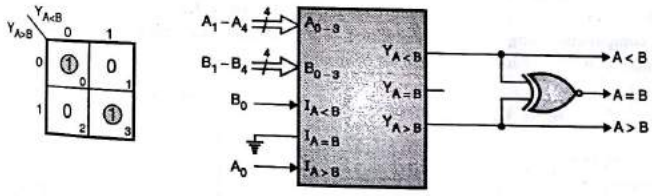
Cases (b) and (c) gives true output. However for case (a) and (d) output $Y_{A=B}$ must be at logic 1.

This can be implemented by considering that,



Since output $Y_{A=B}$ cannot be used, we design a new output for A = B as a function of $Y_{A=B}$ and $Y_{A<B}$ i.e. when $Y_{A>B} = Y_{A<B}$ '0' or '1'. Output $Y_{A=B} = 1$.

Input		Output
$Y_{A>B}$	$Y_{A<B}$	$Y_{A=B}$ (Output)
0	0	1
0	1	0
1	0	0
1	1	1



(C-411) **Fig. P. 6.7.1**

$$Y_{A=B} = \overline{Y}_{A>B}\,\overline{Y}_{A<B} + Y_{A<B}\,Y_{A>B} = Y_{A>B} \odot Y_{A<B}$$

Final circuit diagram is as shown in Fig. P. 6.7.1.

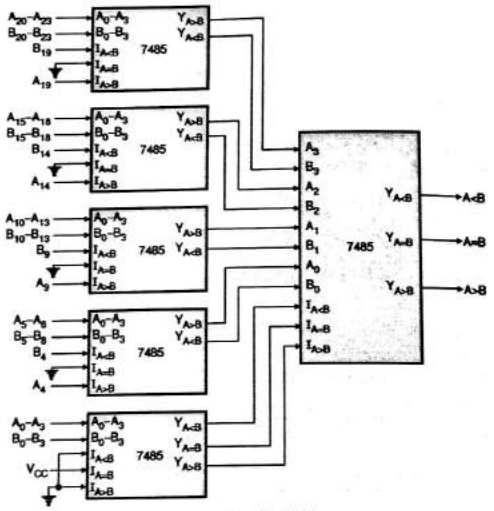
Ex. 6.7.2 : Design 24 bit magnitude comparator using minimum magnitude comparator chips (7485). **May 11. 10 Marks**

Soln. : Refer Fig. P. 6.7.2.

Here we can cascade six chips. But the problem is of propagation delay, therefore circuit is bit slower.

The 24 bit input is fed to five comparators. Since there are only 20 inputs per variable the remaining four are accommodated as explained in previous problem. This connection is not done for LSB nibble comparator.

Scanned by CamScanner



(C-412) **Fig. P. 6.7.2**

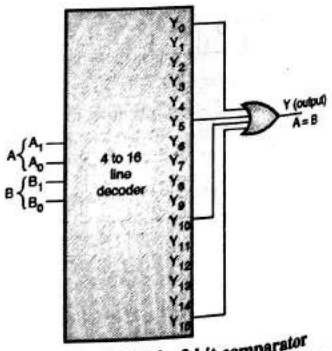
Ex. 6.7.3 : Explain the concept of comparator. Develop the truth table for 2-bit binary comparator and design it using a suitable decoder and additional gates. Dec. 15. 10 Marks

Soln. : For the concept of comparator refer sections 6.7 and 6.7.2.

2 bit comparator using 4 : 16 line decoder : The truth table of a 2 bit comparator is as follows :

Table P. 6.7.3 : Truth table of a 2 bit comparator

A		B		Y = equality output	
A_1	A_0	B_1	B_0		
0	0	0	0	1	y_0
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	y_5
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	y_{10}
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	0	
1	1	1	1	1	y_{15}



(C-1329) **Fig. P. 6.7.3 : 2 bit comparator using 4 line to 16 line decoder**

- From the truth table the output can be expressed in the SOP form as follows :

$$Y = f(A_1, A_0, B_1, B_0) = \Sigma m (0, 5, 10, 15)$$

- The implementation is shown in Fig. P. 6.7.3.

6.8 Arithmetic Logic Unit (ALU) :

MU : Dec. 02. May 03. Dec. 04. May 05. Dec. 05. Dec. 06. May 07. May 08. Dec. 08. May 09. Dec. 10. May 11. Dec. 11. May 12

University Questions

Q.1 Write short notes on : Arithmetic Logic Unit (ALU).
(Dec. 02, May 03, Dec. 04, 10 Marks, May 05, Dec. 05, Dec. 06, May 07, May 08, Dec. 08, May 09, Dec. 10, May 11, Dec. 11, May 12, 4 Marks)

- ALU is a very widely used and popular combinational circuit.
- It is capable of performing the arithmetic as well as the logic operations.
- ALU is the heart of any microprocessor.
- Fig. 6.8.1 shows the block diagram of ALU IC 74181, Table 6.8.1 gives the pin description and Fig. 6.8.2 gives its pin configuration.

Table 6.8.1 : Pin description of 74181

Pin name	Description
$A_0 - A_3$	Operand inputs
$B_0 - B_3$	Operand inputs
$S_0 - S_3$	Function select inputs
M	Mode control input
$\overline{C}_n$	Carry input (active low)
$F_0 - F_3$	Function output
A = B	Comparator output (equality output)
G	Carry generate output
P	Carry propagate output
$\overline{C}_{n+4}$	Carry output (active low)

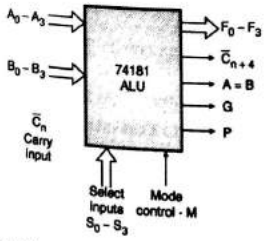
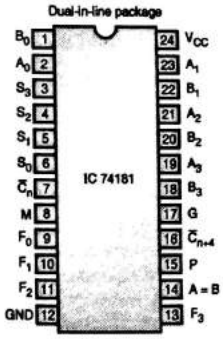


Fig. 6.8.1 : Block diagram of ALU IC 74181



(C-1079)

Fig. 6.8.2 : Pin configuration of the ALU IC 74181

Scanned by CamScanner

- 74181 is a 24-pin IC and it uses the Dual In line (DIP) Package.
- A ($A_0 - A_3$) and B ($B_0 - B_3$) are the two 4 bit words applied at its input.
- It can perform a total of 16 different arithmetic operations which includes addition, subtraction, comparison and doubling operations.
- It provides many logic operations such as AND, OR, NOR, NAND, EX-OR, compare as well on the two four bit variables.
- 74181 is a high speed 4 bit parallel ALU.
- The function to be performed is controlled by the four function select inputs $S_0 - S_3$. These lines can select 16 different operations for one mode (arithmetic) and 16 another operations for the other mode (logic).
- M is the mode control input. Its value decides the mode of operation to be either arithmetic or logic as follows :

  M = 0 For arithmetic operations.

  M = 1 For logic operations.
- G and P outputs are used when more than one IC 74181 circuits are to be used in cascade along with 74182, which is the look ahead carry generator circuit to make the arithmetic operations faster.
- When mode control input is high (M = 1), then the logic operations are performed on the individual bits and all the internal carries are inhibited.
- When mode control input is low (M = 0), then the arithmetic operations are performed on the two 4-bit words and all the internal carries are enabled.
- IC 74181 incorporates full internal carry lookahead. Due to this its speed of operation gets increased to a great extent.
- It provides a ripple carry between the devices using the C_{n+4} output.
  (see cascading of two 74181 s).
- Or for exploiting the option of carry lookahead between the packages, we have to use the P (carry propagate) and G (carry generate) outputs. This option should be used only when the speed requirements are stringent.
- If low speed of operation is acceptable, the ripple carry operation using C_{n+4} and C_n should be exercised.

A = B output :

1. A = B output is used to indicate that both the operands are equal. This output goes HIGH when the unit is in the subtract mode and A = B.
2. This output also goes high when all the four "Function outputs" are HIGH.
3. It is possible to wire AND the A = B outputs when more than one 74181s are being used. The wire ANDing becomes possible because A = B is an open collector output. This enables us to compare words which are longer than 4-bits.

Function tables :

- Table 6.8.2(a) shows the function table for IC 74181. It is valid for the active high operands and active high outputs, and with $\bar{C}_n = 1$ i.e. no carry.

Table 6.8.2(a) : Function table for IC 74181 with active high data and $\bar{C}_n = 1$ (no carry)

Function Select Inputs				Active high data and $\bar{C}_n = 1$	
S_3	S_2	S_1	S_0	Logic operations (M = 1)	Arithmetic operations (M = 0)
0	0	0	0	$F = \bar{A}$ (Inversion)	F = A
0	0	0	1	$F = \overline{A+B}$ (NOR)	F = A + B
0	0	1	0	$F = \bar{A} \cdot B$	$F = A + \bar{B}$
0	0	1	1	F = 0	F = minus 1 (2's comp)
0	1	0	0	$F = \overline{AB}$ (NAND)	$F = A$ plus $A\bar{B}$
0	1	0	1	$F = \bar{B}$ (Invert)	$F = (A + B)$ plus $A\bar{B}$
0	1	1	0	$F = A \oplus B$ (EXOR)	F = A minus B minus 1
0	1	1	1	$F = A\bar{B}$	$F = A\bar{B}$ minus 1
1	0	0	0	$F = \bar{A} + B$	F = A plus AB
1	0	0	1	$F = \overline{A \oplus B}$ (EXNOR)	F = A plus B
1	0	1	0	F = B	$F = (A + \bar{B})$ plus AB
1	0	1	1	F = AB (AND)	F = AB minus 1
1	1	0	0	F = logic 1	F = A plus A *
1	1	0	1	$F = A + \bar{B}$	$F = (A + B)$ plus A
1	1	1	0	F = A + B (OR)	$F = (A + \bar{B})$ plus A
1	1	1	1	F = A	F = A Minus 1

Each bit is shifted to the next more significant position.

- The arithmetic operations mentioned in function Table 6.8.2(a), correspond to no carry input. They will get modified if the carry input is present as shown in Table 6.8.2(b).
- It is possible to use IC 74181 with either active high inputs or with active low inputs. With active low inputs the device produces active low outputs and with active high inputs it produces active high outputs.
- The function table for active low inputs and outputs has been given in Table 6.8.2(b).

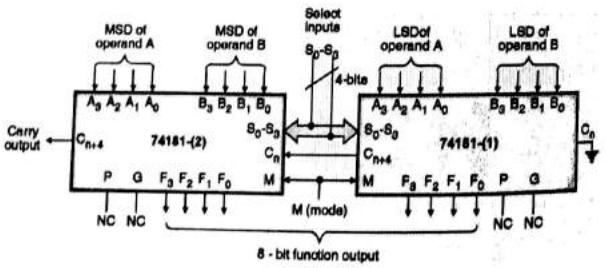
Scanned by CamScanner

Table 6.8.2(b) : Function table for IC 74181 with active low data and $\bar{C}_n = 0$ (with carry)

Function Select Inputs				Active low data and $\bar{C}_n = 0$	
S_3	S_2	S_1	S_0	Logic operations M = 0	Arithmetic operations M = 1
0	0	0	0	$F = \bar{A}$ (inversion)	F = A minus 1
0	0	0	1	$F = \overline{AB}$ (NAND)	F = AB minus 1
0	0	1	0	$F = \bar{A} + \bar{B}$	$F = A\bar{B}$ minus 1
0	0	1	1	F = 1	F = minus 1
0	1	0	0	$F = \bar{A} + \bar{B}$	F = A plus $(A + \bar{B})$
0	1	0	1	$F = \bar{B}$ (inversion)	F = AB plus $(A + \bar{B})$

6.8.1 Cascading of two 74181 ALUs :

- When an 8-bit ALU is to be designed, two 74181 ICs are to be cascaded as shown in Fig. 6.8.3.
- The select input lines ($S_3 - S_0$) and the mode select (M) inputs of both the IC are connected to each other parallelly and brought out.
- This ensures that both the IC's operate in the same mode and select the same function.
- C_n (Carry input) of 74181-(1) is connected to ground whereas C_{n+4} (Carry output) of 74181-(1) is connected to carry input C_n of 74181 – (2). This connection will propagate the carry of one stage propagated to the next stage.
- The final carry output C_{n+4} is produced by 74181 – (2).
- The LSDs of the two input operands are connected to the four bit inputs of 74181-(1) and the MSDs are connected to the four bit inputs of 74181-(2).



(C-1253(c)) **Fig. 6.8.3 : 8 bit ALU using 74181**

6.9 Multiplexer (Data Selector) :

MU : May 04, Dec. 07, Dec. 08, Dec. 10, Dec. 16

University Questions

Q. 1 What is multiplexer ? **(Dec. 07, Dec. 16, 4 Marks)**

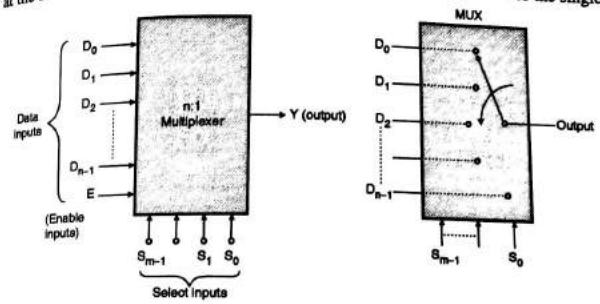
Q. 2 Write short note on multiplexer. **(May 04, Dec. 08, Dec. 10, 3 Marks)**

- Multiplexer is a special type of combinational circuit. The block diagram of an n-to-1 multiplexer is shown in Fig. 6.9.1(a) and its equivalent circuit is shown in Fig. 6.9.1(b).

- As shown, there are n-data inputs (D_0, D_1, ...D_{n-1}), one output Y and m select inputs, S_0, S_1 .....S_{m-1}. The relation between the number of data inputs (n) and number of select inputs (m) is as follows :

$$n = 2^m$$

- A multiplexer is a digital circuit which selects one of the n data inputs and routes (connects) it to the output. The selection of one of the n inputs is done with the help of the select inputs.
- To select n inputs we need m select lines such that $2^m = n$. Depending on the digital code applied at the select inputs, one out of n data sources is selected and transmitted to the single output Y.



(a) Block diagram of an n : 1 multiplexer **(b) Equivalent circuit**

(C-413) **Fig. 6.9.1**

- E is called as a strobe or enable input which is useful for cascading. It is generally an active low terminal, that means it will perform the required operation when it is low.
- As shown in Fig. 6.9.1(b) the multiplexer acts like a digitally controlled single pole, multiple way switch. The output gets connected to only one of the n data inputs at given instant of time.

6.9.1 Necessity of Multiplexers :

- In most of the electronic systems, the digital data is available from more than one sources. It is necessary to route this data over a single line.
- Under such circumstances we require a circuit which selects one of the many sources at a time.
- This circuit is nothing else but a multiplexer, which has many inputs, one output and some select inputs.
- Multiplexer improves the reliability of the digital system because it reduces the number of external wired connections.

6.10 Advantages of Multiplexers :

1. It reduces the number of wires, required to be used.
2. A multiplexer reduces the circuit complexity and cost.
3. We can implement many combinational circuits using MUX.
4. It simplifies the logic design.
5. It does not need the K maps for simplification.

6.11 Types of Multiplexers :

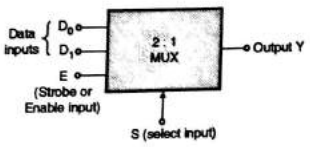
The types of multiplexers are :

1. 2 : 1 multiplexer
2. 4 : 1 multiplexer
3. 8 : 1 multiplexer
4. 16 : 1 multiplexer
5. 32 : 1 multiplexer

Scanned by CamScanner

6.11.1 2 : 1 Multiplexer :

- The block schematic of a 2 : 1 multiplexer is shown in Fig. 6.11.1(a). It has two data inputs D_0 and D_1, one select input S, an enable input and one output.
- The truth table of this MUX is shown in Fig. 6.11.1(b).



(a) Block diagram

Enable	Select input S	Output Y
0	X	0
1	0	D_0
1	1	D_1

X = Don't care

(b) Truth table

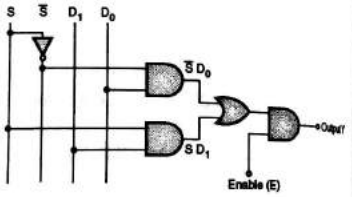
(C-414) **Fig. 6.11.1 : 2 : 1 multiplexer**

Realization of a 2 : 1 MUX using gates :

- Write a more elaborate truth table for 2 : 1 MUX as shown in Table 6.11.1.

Table 6.11.1 : Truth table of a 2 : 1 MUX

Enable E	Select S	D_1	D_0	Output Y
0	X	X	X	0
1	0	X	0	0
1	0	X	1	1
1	1	0	X	0
1	1	1	X	1



(C-415) **Fig. 6.11.2 : Realization of 2 : 1 MUX using gates**

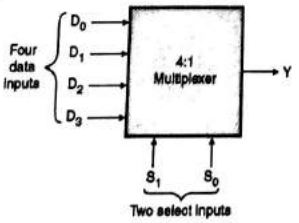
- From the truth table it is clear that Y = 1 for only two conditions shown by the shaded boxes. We can write down the Boolean expression by taking into consideration these two conditions as follows :

$$\therefore \quad Y = E\overline{S}D_0 + ESD_1 = E(\overline{S}D_0 + SD_1)$$

- The realization using gates is shown in Fig. 6.11.2.

6.11.2 A 4 : 1 Multiplexer :

- Fig. 6.11.3 shows the block diagram of a 4 : 1 multiplexer and Table 6.11.2 gives its truth table.



(C-416) **Fig. 6.11.3 : 4 : 1 multiplexer**

Table 6.11.2 : Truth table

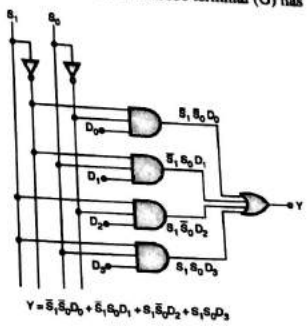
Select inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- Note that n = 4 hence number of select lines i.e. m = 2 so that $2^m = n$. The strobe terminal (G) has not been included to reduce the complexity.
- The truth table tells us that if $S_1 S_0 = 00$, the data bit D_0 is selected and routed to output

  $\therefore \quad Y = D_0$ .... when $S_1 S_0 = 00$
- Similarly if $S_1 S_0 = 01$, then D_1 is selected and routed to the output

  $\therefore \quad Y = D_1$ .... when $S_1 S_0 = 01$
- $Y = D_2$ for $S_1 S_0 = 10$ and

  $Y = D_3$ for $S_1 S_0 = 11$
- The output will be high when the selected input (D_0, D_1 etc.) is 1. Hence the logical expression for output in the SOP form is as follows :

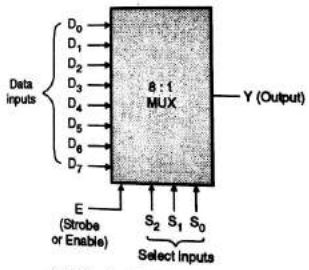


(C-417) **Fig. 6.11.4 : Realization of 4 : 1 multiplexer using basic gates**

$$Y = \overline{S}_1\overline{S}_0D_0 + \overline{S}_1S_0D_1 + S_1\overline{S}_0D_2 + S_1S_0D_3$$

- This expression can be realized using basic gates as shown in Fig. 6.11.4.

6.11.3 8 : 1 Multiplexer :



(a) Block diagram

Enable	Select inputs			Output
E	S_2	S_1	S_0	Y
0	X	X	X	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

(b) Truth table

(C-419(a)) **Fig. 6.11.5 : 8 : 1 multiplexer**

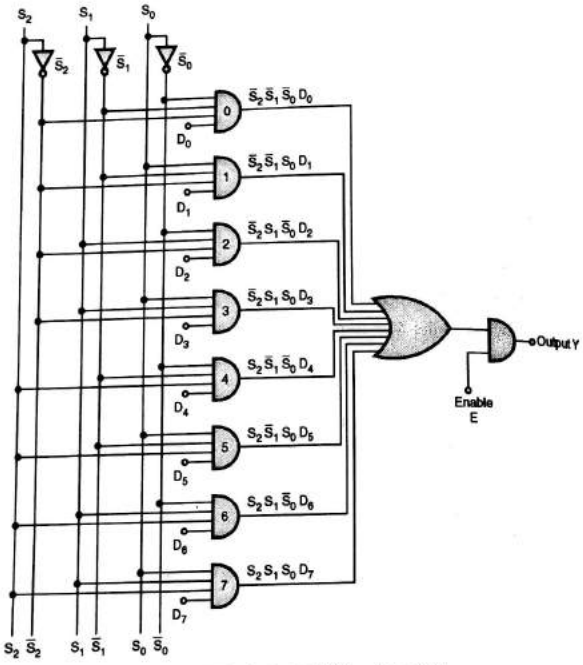
- The block diagram of an 8 : 1 MUX is shown in Fig. 6.11.5(a) and its truth table is shown in Fig. 6.11.5(b).
- It has eight data inputs, one enable input, three select inputs and one output.

Operating principle :

- When the strobe or enable input is 0, the output of the multiplexer will be 0 irrespective of any input.
- With E = 1, we can select any one of the eight data inputs and connect it to the output.
- For example if $S_2 S_1 S_0 = 0\ 1\ 1$ then the data input D_3 is selected and output Y will follow the selected input D_3.

Scanned by CamScanner

Realization using gates :



(C-420) **Fig. 6.11.6 : 8 : 1 MUX using gates**

- The expression for output Y can be obtained from the truth table of Fig. 6.11.5(b) as :

$$Y = E \cdot [\bar{S}_2\,\bar{S}_1\,\bar{S}_0\,D_0 + \bar{S}_2\,\bar{S}_1\,S_0D_1 + \bar{S}_2\,S_1\bar{S}_0\,D_2 + \bar{S}_2\,S_1S_0D_3 + S_2\bar{S}_1\bar{S}_0\,D_4 + S_2\bar{S}_1S_0D_5 + S_2S_1\bar{S}_0D_6 + S_2S_1S_0D_7]$$

- The realization of 8 : 1 MUX using gates is shown in Fig. 6.11.6.

6.11.4 16 : 1 MUX :

- The block diagram of a 16 : 1 MUX is shown in Fig. 6.11.7.
- It has 16 data inputs, 4 select lines, one enable or strobe input and one output.

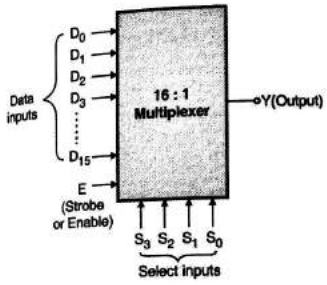
Operating principle :

- When the strobe or enable input E = 0, the multiplexer output Y = 0 irrespective of the status of all other inputs.

- With E = 1, we can select any one of the 16 inputs with the help of select input lines and connect the selected input to the output as shown in the truth table of Table 6.11.3.
- The truth table of a 16 : 1 multiplexer is as follows :

Table 6.11.3 : Truth table of 16 : 1 multiplexer

Enable	Select inputs				Output
E	S_3	S_2	S_1	S_0	Y
0	0	0	0	0	D_0
1	0	0	0	1	D_1
1	0	0	1	0	D_2
1	0	0	1	1	D_3
1	0	1	0	0	D_4
1	0	1	0	1	D_5
1	0	1	1	0	D_6
1	0	1	1	1	D_7
1	1	0	0	0	D_8
1	1	0	0	1	D_9
1	1	0	1	0	D_{10}
1	1	0	1	1	D_{11}
1	1	1	0	0	D_{12}
1	1	1	0	1	D_{13}
1	1	1	1	0	D_{14}
1	1	1	1	1	D_{15}



(C-421) **Fig. 6.11.7 : Block schematic of a 16 : 1 multiplexer**

6.11.5 Applications of a Multiplexer :

Some of the important applications of a multiplexer are as follows :

1. It is used as a data selector to select one out of many data inputs.
2. It is used for simplification of logic design.
3. In the data acquisition system.
4. In designing the combinational circuits.
5. In the D/A converters.
6. To minimize the number of connections in a logic circuit.

6.12 Study of Different Multiplexer ICs :

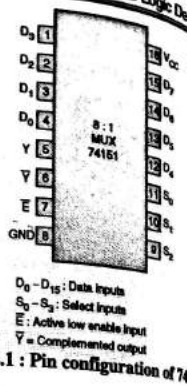
- The multiplexer ICs available in market are given in Table 6.12.1.
- Out of these the two most commonly used multiplexer ICs are : 74150 (16 : 1 multiplexer) and 74151 (8 : 1 multiplexer), and 74153 (Dual 4:1 multiplexer).

Scanned by CamScanner

- The pin configuration for these ICs is shown in Fig. 6.12.1.

Table 6.12.1

IC Number	Description	Output
74157	Quad 2 : 1 Mux	Same as input (No inversion)
74158	Quad 2 : 1 Mux	Inverted output
74153	Dual 4 : 1 Mux	Same as input (No inversion)
74352	Dual 4 : 1 Mux	Inverted output
74151A	8 : 1 Mux	Complementary outputs
74152	8 : 1 Mux	Inverted output
74150	16 : 1 Mux	Inverted output



(C-422) **Fig. 6.12.1 : Pin configuration of 74151**

6.13 Multiplexer Tree :

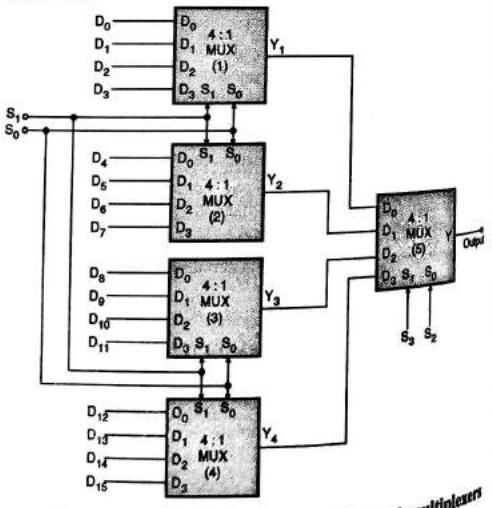
- The multiplexers having more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs.
- This is called as a multiplexer tree. This concept will be clear after solving the following examples.

Ex. 6.13.1 : Implement a 16 : 1 multiplexer using 4 : 1 multiplexers. **May 09, Dec. 11, 10 Marks**

Soln. : Refer Fig. P. 6.13.1 for implementation.

Description :

- The select inputs S_1 and S_0 of the multiplexers 1, 2, 3 and 4 are connected together.
- The select inputs S_3 and S_2 are applied to the select inputs S_1 and S_0 of MUX-5.
- The outputs Y_1, Y_2, Y_3, Y_4 are applied to the data inputs D_0, D_1, D_2 and D_3 of MUX-5 as shown in Fig. P. 6.13.1.



(C-425) **Fig. P. 6.13.1 : 16 : 1 multiplexer using 4 : 1 multiplexers**

- The operation can be summarized using Table P. 6.13.1.

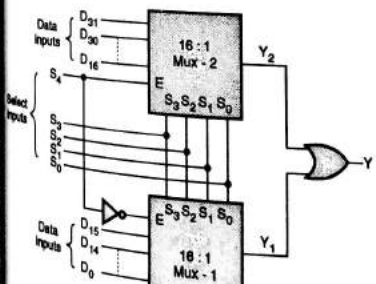
Table P. 6.13.1 : Summary of operation

Select inputs				Mux. Outputs				Final output	
S_3	S_2	S_1	S_0	Y_1	Y_2	Y_3	Y_4	Y	
0	0	0	0	D_0	D_4	D_8	D_{12}	D_0	$S_3 S_2 = 00$ ∴ MUX-5 selects Y_1
0	0	0	1	D_1	D_5	D_9	D_{13}	D_1	
0	0	1	0	D_2	D_6	D_{10}	D_{14}	D_2	
0	0	1	1	D_3	D_7	D_{11}	D_{15}	D_3	
0	1	0	0	D_0	D_4	D_8	D_{12}	D_4	$S_3 S_2 = 01$ ∴ MUX-5 selects Y_2
0	1	0	1	D_1	D_5	D_9	D_{13}	D_5	
0	1	1	0	D_2	D_6	D_{10}	D_{14}	D_6	
0	1	1	1	D_3	D_7	D_{11}	D_{15}	D_7	
1	0	0	0	D_0	D_4	D_8	D_{12}	D_8	$S_3 S_2 = 10$ ∴ MUX-5 selects Y_3
1	0	0	1	D_1	D_5	D_9	D_{13}	D_9	
1	0	1	0	D_2	D_6	D_{10}	D_{14}	D_{10}	
1	0	1	1	D_3	D_7	D_{11}	D_{15}	D_{11}	
1	1	0	0	D_0	D_4	D_8	D_{12}	D_{12}	$S_3 S_2 = 11$ ∴ MUX-5 selects Y_4
1	1	0	1	D_1	D_5	D_9	D_{13}	D_{13}	
1	1	1	0	D_2	D_6	D_{10}	D_{14}	D_{14}	
1	1	1	1	D_3	D_7	D_{11}	D_{15}	D_{15}	

Ex. 6.13.2 : Implement a 32 : 1 multiplexer using 16 : 1 multiplexers. **May 08, 10 Marks**

Soln. :

- The construction of a 32 : 1 multiplexer using 16 : 1 multiplexer is shown in Fig. P. 6.13.2.



(C-428) **Fig. P. 6.13.2**

Table P. 6.13.2

Select inputs					MUX enabled	Output Y
S_4	S_3	S_2	S_1	S_0		
0	0	0	0	0	MUX – 1	D_0
0	0	0	0	1	MUX – 1	D_1
		:			:	:
		:			:	:
0	1	1	1	1	MUX – 1	D_{15}
1	0	0	0	0	MUX – 2	D_{16}
1	0	0	0	1	MUX – 2	D_{17}
:		:			:	:
:		:			:	:
1	1	1	1	1	MUX – 2	D_{31}

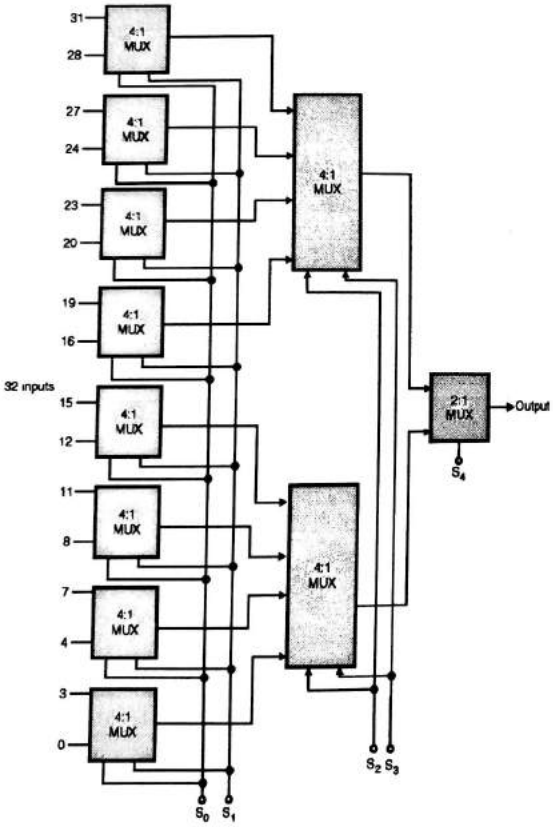
- The four select lines S_0, S_1, S_2 and S_3 of both the 16 : 1 multiplexers are made common and brought out.

Scanned by CamScanner

- Whereas the most significant select line S_4 is used for selection of multiplexers. For $S_4 = 0$, the MUX – 1 will be selected and $Y = Y_1$ = one of the inputs from D_0 to D_{15}.
- With $S_4 = 1$, MUX–2 will be enabled. So $Y = Y_2$ = one of the inputs from D_{16} to D_{31}.
- Table P. 6.13.2 shows the truth table.

Ex. 6.13.3 : Design a 32 : 1 multiplexer using 4 : 1 multiplexers. **May 12, 10 Marks**

Soln. : 32 : 1 MUX using 4 : 1 multiplexers is as shown in Fig. P. 6.13.3.



(C-3296) **Fig. P. 6.13.3 : 32 : 1 MUX using 4 : 1 MUX**

.14 Use of Multiplexers in Combinational Logic Design :

Multiplexers ICs for 2 : 1 , 4 : 1, 8 : 1 and 16 : 1 multiplexers are available. We can use them to implement the given Boolean expressions representing the combinational circuits.

- Thus it is possible to design and implement many combinational circuits by using multiplexer and a few logic gates.
- Advantages of using multiplexers for logic design are as follows :
  1. Logic design is simplified.
  2. It is not necessary to simplify the logic expression.
  3. It minimizes the number of ICs required to be used.

Use of MUX for combinational circuit design :

1. A truth table or logic expression in standard SOP or POS form is given to us.
2. We have to follow the design procedure given below to use MUX for implementing the given logical expression.

Design procedure :

Step 1 : Identify the decimal number corresponding to each minterm in the given expression as illustrated below.

$$\text{If } Y = \underbrace{\bar{A}\,B\,\bar{C}}_{0} + \underbrace{A\,\bar{B}\,C}_{5} + \underbrace{A\,B\,C}_{7}$$

Step 2 : The input lines of the multiplexer, corresponding to these numbers (0, 5 and 7) are connected to logic 1 level.

Step 3 : All the other input lines of multiplexer are connected to logic 0 level.

Step 4 : The inputs (A, B, C) are to be connected to the select inputs.

The following example will make this concept clear.

Ex. 6.14.1 : Realize the logic function of the truth table given in Table P. 6.14.1 using a multiplexer.

Soln. :

Step 1 : Express the output Y in the standard SOP form $Y = \sum m(1, 3, 4, 6)$

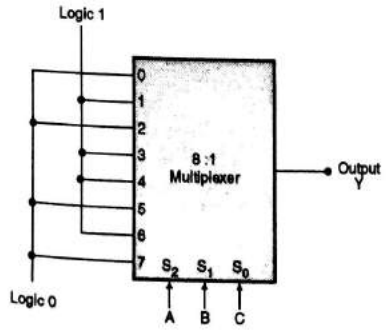
Step 2 : Connect the data inputs 1, 3, 4 and 6 to logic 1 and the remaining to logic 0.

Step 3 : Connect the inputs A, B, C to the select lines S_2 S_1 and S_0 respectively.

Step 4 : Draw the logic diagram as shown in Fig. P. 6.14.1.

Table P. 6.14.1

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



(C-429) **Fig. P. 6.14.1 : Logic diagram for Ex. 6.14.1**

Scanned by CamScanner

Ex. 6.14.2 : Implement the following expression using a multiplexer.

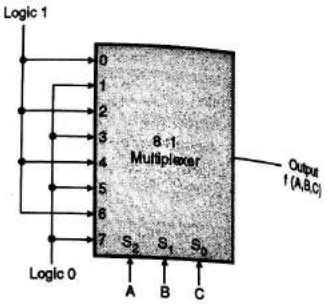
$$f(A, B, C) = \sum m(0, 2, 4, 6)$$

Soln. : Since there are three variables, the multiplexer having three select inputs should be used. Hence an 8 : 1 multiplexer as shown in Fig. P. 6.14.2 should be used.

Step 1 : Identify the decimal number corresponding to each minterm.
The decimal numbers corresponding the minterms are 0, 2, 4 and 6.

Step 2 : Connect the data input lines 0, 2, 4 and 6 to logic 1 and the remaining lines (1, 3, 5, 7) to logic 0 as shown in Fig. P. 6.14.2.

Step 3 : Connect the variables A, B and C to the select inputs.



(C-430) **Fig. P. 6.14.2 : Implementation of a logic expression using a multiplexer**

Note : We can use IC 74151 which is an 8 : 1 Multiplexer to implement Boolean equations using 8 : 1 MUX.

Ex. 6.14.3 : Implement the logic function f (A, B, C) = $\sum$ m (1, 3, 4, 6) using a 4 : 1 multiplexer.

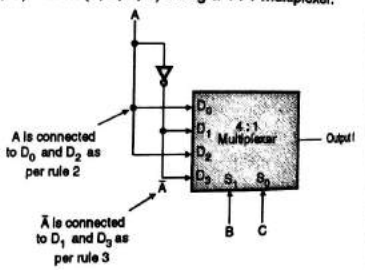
Soln. :

The logic function to be implemented is,

$$f(A, B, C) = \sum m(1, 3, 4, 6)$$

Step 1 : Apply two variables B and C to the select inputs :

As shown in Fig. P. 6.14.3(a), the two input variables B and C are applied to the select lines S_1 and S_0 respectively.

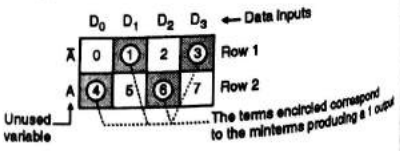


(C-431) **Fig. P. 6.14.3(a) : Logic diagram**

Step 2 : Write the design table :

The design table is shown in Fig. P. 6.14.3(b).

- The data inputs D_0 to D_3 have been written at the top of the table and the two possible values A and $\bar{A}$ of the unused variable A have been written.
- In the eight boxes we enter the decimal numbers corresponding to the minterms (0 to 7) serially.



(C-432) **Fig. P. 6.14.3(b) : Design table**

- Encircle those minterms corresponding to which the output is 1 (minterms 1, 3, 4, 6).

Step 3 : Check each column in the design table :

The columns of design table are inspected using the following rules :

Rule 1 : If both the minterms in a column are not circled, then apply logic 0 to the corresponding data input. Note that there is no such column in our implementation table.

Rule 2 : If only the minterm in the second row is encircled (see columns 1 and 3) then "A" should be applied to that data input. **Hence we should apply A to the D_0 and D_2 inputs.**

Rule 3 : If only the minterm in the first row is encircled, (see columns 2 and 4), then $\bar{A}$ should be connected to that data input. **Hence we should apply $\bar{A}$ to the D_1 and D_3 inputs.**

Rule 4 : If both the minterms in a column are encircled, then apply a logic 1 to the corresponding data input. Note that there is no such column in our implementation table.

Step 4 : Draw the logic diagram :

The logic diagram is as shown in Fig. P. 6.14.3(a).

Note : We can use IC 74153 which is a dual 4 : 1 Multiplexer in order to implement the Boolean expressions using 4 : 1 MUX.

Ex. 6.14.4 : Implement a full adder using 8 : 1 multiplexer.

Soln. :

Step 1 : Write the truth table of a full adder :

The truth table of a full adder is shown in Table P. 6.14.4.

Table P. 6.14.4 : Truth table for a full adder

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

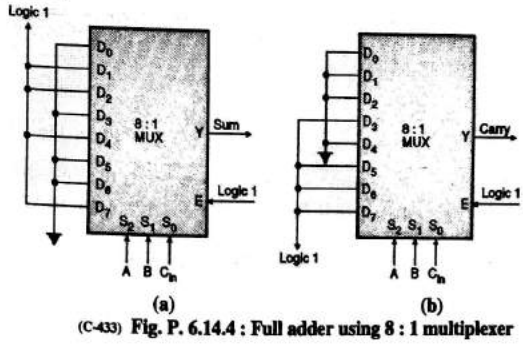
Step 2 : Identify the decimal number corresponding to each minterm :

The sum and carry outputs can be expressed in the standard SOP form as follows :

$$S = \sum m(1, 2, 4, 7) \text{ and } C = \sum m(3, 5, 6, 7)$$

Step 3 : Realize the circuit using two 8 : 1 multiplexers :

We have to use two 8 : 1 multiplexers, one each for the sum and carry outputs. The realization is shown in Fig. P. 6.14.4.



(C-433) **Fig. P. 6.14.4 : Full adder using 8 : 1 multiplexer**

Scanned by CamScanner

Ex. 6.14.5 : Implement the following Boolean function using 8 : 1 multiplexer.

$f(A, B, C, D) = \sum m(2, 4, 5, 7, 10, 14)$

Soln. :

Step 1 : Apply the variables B, C and D to the select inputs :

As shown in Fig. P. 6.14.5(b), the three variables B, C and D are connected to the select lines S_2, S_1 and S_0 respectively.

Step 2 : Write the design table :

The design table is as shown in Fig. P. 6.14.5(a).

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
$\bar{A}$	0	1	2	3	4	5	6	7	Row 1
A	8	9	10	11	12	13	14	15	Row 2
Input to Mux :	0	0	1	0	$\bar{A}$	$\bar{A}$	A	$\bar{A}$	

The terms encircled correspond to the minterms producing a 1 output

(C-434) **Fig. P. 6.14.5(a) : Design table**

- In the sixteen boxes we have entered the decimal numbers corresponding to the minterms 0 to 15 serially.
- Encircle those minterms which correspond to an output = 1 (minterms 2, 4, 5, 7, 10, 14).

Step 3 : Inspect each column in the design table :

Rule 1 : If both the minterms in a column are not circled, then apply logic 0 to the corresponding data input. Note that there is no such column in our implementation table.

Rule 2 : If only the minterm in the second row is encircled then "A" should be applied to that data input. **Hence we should apply A to the D_6 input.**

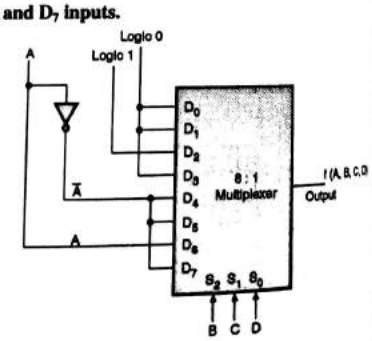
Rule 3 : If only the minterm in the first row is encircled, then $\bar{A}$ should be connected to that data input.

Hence we should apply $\bar{A}$ to the D_4, D_5 and D_7 inputs.

Rule 4 : If both the minterms in a column are encircled, then apply a logic 1 to the corresponding data 1 input. Note that there is no such column in our implementation table.

Step 4 : Draw the logic diagram :

The logic diagram is as shown in Fig. P. 6.14.5(b).



(C-435) **Fig. P. 6.14.5(b) : Logic diagram**

Ex. 6.14.6 : Implement the following function using 16 : 1 multiplexer

$f(A, B, C, D, E) = \sum m(2, 4, 5, 7, 10, 14, 15, 16, 17, 25, 26, 30, 31)$

Soln. :

Step 1 : Apply the variables B, C, D, E to the select inputs :

As shown in Fig. P. 6.14.6 the variables B, C, D, E are applied to the select lines S_3, S_2, S_1, S_0 respectively.

Step 2 : Write the design table :

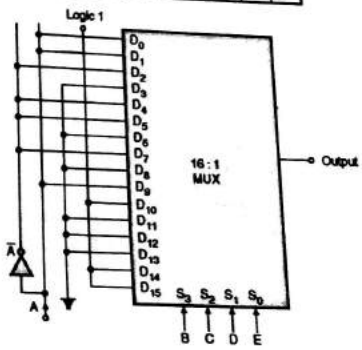
The design table is as shown in Table P. 6.14.6.

(C-435(a)) **Table P. 6.14.6 : Design table**

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
$\bar{A}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Input to MUX	A	A	$\bar{A}$	0	$\bar{A}$	$\bar{A}$	0	$\bar{A}$	0	A	1	0	0	0	1	1

Step 3 : Realize the logic circuit :

The logic circuit is as shown in Fig. P. 6.14.6.



(C-436) **Fig. P. 6.14.6 : Logic diagram**

Ex. 6.14.7 : Implement the following function using 4 : 1 multiplexers with active low strobe input :

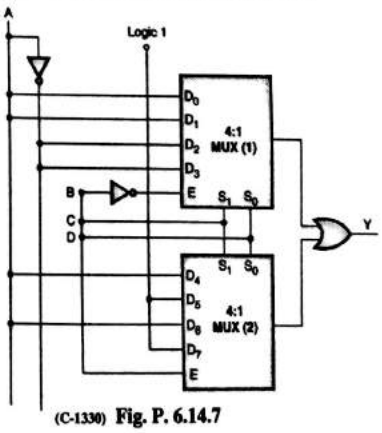
$f(A, B, C, D) = \sum m(2, 3, 5, 7, 8, 9, 12, 13, 14, 15)$.

Soln. :

Step 1 : Write the design table : (C-1329(a))

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX :	A	A	$\bar{A}$	$\bar{A}$	A	1	A	1
	MUX - 1				MUX - 2			

Step 2 : Implementation of logic circuit :



(C-1330) **Fig. P. 6.14.7**

Scanned by CamScanner

Ex. 6.14.8 : Implement the following function using single 4 : 1 MUX and logic gates :
$F(A, B, C, D) = \Sigma m(0, 1, 5, 9, 10, 15)$

Soln. :

$$F(A, B, C, D) = \Sigma m(0, 1, 5, 9, 10, 15)$$

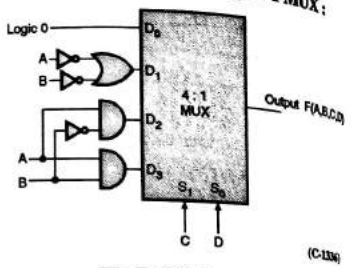
Step 1 : Write the design table :

	D_0	D_1	D_2	D_3
$\bar{A}\bar{B}$	0	1	2	3
$\bar{A}B$	4	5	6	7
$A\bar{B}$	8	9	10	11
AB	12	13	14	15
	0		$A\bar{B}$	AB

(C-1335)

$$D_1 = \bar{A}\bar{B} + \bar{A}B + A\bar{B} = \bar{A}(\bar{B} + B) + A\bar{B}$$
$$= \bar{A} + A\bar{B} = \bar{A} + \bar{B}$$

Step 2 : Implementation using 4 : 1 MUX :



(C-1336)

Fig. P. 6.14.8

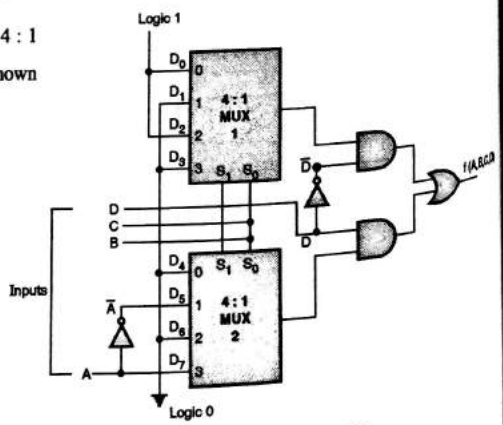
Ex. 6.14.9 : Implement the following function using 4 : 1 MUX and logic gates :
$F(A, B, C, D) = \Sigma(0, 2, 5, 8\ 10, 15)$.

Soln. :

- For the given function, we need to use an 8 : 1 MUX. We will first write the design table for 8 : 1 MUX and then will implement the circuit by 4 : 1 multiplexers and logic gates.
- The implementation using 4 : 1 multiplexers and gates is shown in Fig. P. 6.14.9.

(C-3584) **Table P. 6.14.9 : Design table**

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX :	1	0	1	0	0	$\bar{A}$	0	A
	MUX - 1				MUX - 2			



6.14.1 Implementation of a Logical Expression in the Non-standard SOP Form :

Till now we have seen the implementation of expressions in the standard SOP form. But if the given expression is not in the standard form, then we have to first bring it to the standard form and then proceed. Ex. 6.14.10 illustrates this concept.

Ex. 6.14.10 : Implement the following Boolean expression using 8 : 1 multiplexer.
$f(A, B, C, D) = \bar{A}\bar{B}\bar{D} + ABC + \bar{B}CD + \bar{A}CD$ OR
Implement the following Boolean expression using 8 : 1 MUX.
$Y = \Sigma m(0, 2, 3, 7, 11, 14, 15)$

Soln. :

Step 1 : Convert the given expression to standard SOP form :

$$f(A, B, C, D) = \bar{A}\bar{B}\bar{D}(C + \bar{C}) + ABC(D + \bar{D}) + \bar{B}CD(A + \bar{A}) + \bar{A}CD(B + \bar{B})$$
$$= \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + ABCD + ABC\bar{D} + A\bar{B}CD + \bar{A}\bar{B}CD$$
$$+ \bar{A}BCD + \bar{A}\bar{B}CD$$

This is the expression in the standard SOP form.

$$\therefore f(A, B, C, D) = \sum m(2, 0, 15, 14, 11, 3, 7, 3)$$

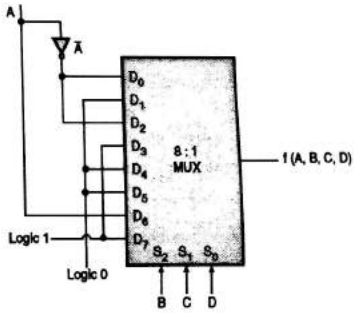
As $X + X = X$ $\quad \therefore \bar{A}\bar{B}CD + \bar{A}\bar{B}CD = \bar{A}\bar{B}CD$

$$\therefore f(A, B, C, D) = \sum m(0, 2, 3, 7, 11, 14, 15)$$

Step 2 : Write the design table :

(C-437(a)) **Table P. 6.14.10 : Design table**

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Inputs to MUX	$\bar{A}$	0	$\bar{A}$	1	0	0	A	0



(C-438) **Fig. P. 6.14.10 : Implementation**

Step 3 : Implementation using 8 : 1 multiplexer :

The implementation is as shown in Fig. P. 6.14.10.

6.14.2 Implementing a Standard POS Expression using Multiplexer :

Let us consider the Boolean expression in the standard POS form and its implementation using a multiplexer.

Scanned by CamScanner

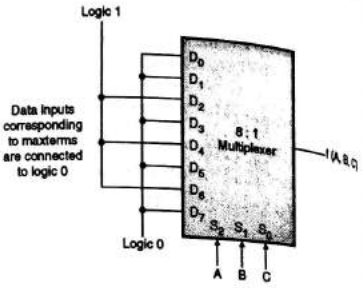
Ex. 6.14.11 : Implement the following logic function using the 8 : 1 multiplexer.

$$f(A, B, C) = \Pi M (0, 1, 3, 5, 7)$$

Soln. :

In the given expression, the maxterms have been specified.

- Hence we should connect the corresponding data inputs (D_0, D_1, D_3, D_5 and D_7) to logic 0.
- And connect the remaining data inputs to logic 1.
- Connect the input variables A, B, C to the select inputs S_2, S_1 and S_0 respectively. The logic diagram is as shown in Fig. P. 6.14.11.



(C-439) **Fig. P. 6.14.11 : Implementation of standard POS expression**

Ex. 6.14.12 : Implement the following logic function using 4 : 1 multiplexer.

$$f(A, B, C) = \Pi M (0, 1, 3, 5, 7)$$

Soln. :

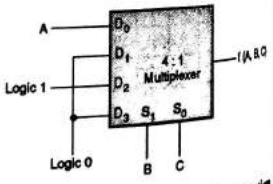
- Connect B and C to the select inputs S_1 and S_0 respectively.
- For the connections of data inputs (D_0 to D_3) and A write down the design table as shown in Fig. P. 6.14.12(a).

	D_0	D_1	D_2	D_3	
$\bar{A}$	0	1	2	3	Row 1
A	4	5	6	7	Row 2
Input to MUX:	A	0	1	0	

Encircle the maxterms which are not included in the given Boolean expression.

(C-440) **Fig. P. 6.14.12(a) : Design table**

- Note that as the given expression is in the POS form, we should encircle those maxterms which are not included in the given Boolean expression.
- The rules for deciding the input to the data lines (D_0 to D_3) are however the same as those stated earlier for the SOP form.
- The logic diagram is as shown in Fig. P. 6.14.12(b).



(C-441) **Fig. P. 6.14.12(b) : Implementation using 4 : 1 multiplexer**

Ex. 6.14.13 : Implement the following function using an 8 : 1 multiplexer.

$$f(A, B, C, D) = \Pi M (0, 2, 4, 6, 8, 10, 12, 14)$$

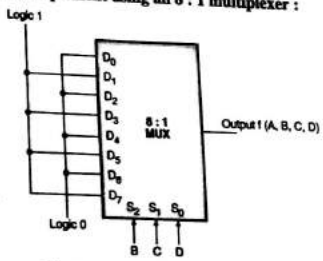
Soln. :

Step 1 : Write the design table :

Table P. 6.14.13 : Design table

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Inputs to MUX	0	1	0	1	0	1	0	1

Step 2 : Implement using an 8 : 1 multiplexer :



(C-442) **Fig. P. 6.14.13 : Implementation using 8 : 1 multiplexer**

The implementation is as shown in Fig. P. 6.14.13.

Ex. 6.14.14 : Realise FULL subtractor circuit using 4:1 Multiplexer and 3 L : 8 L Decoder [Active high input and Active low output decoder] **Dec. 07, May 08, Dec. 09, 10 Marks**

Soln. :

Step 1 : Write the truth table of full subtractor :

Step 2 : Write the design tables for sum and carry outputs :

Table P. 6.14.14 : Truth table for a full subtractor

Inputs			Outputs	
A	B	B_{in}	D_0	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Difference D_0

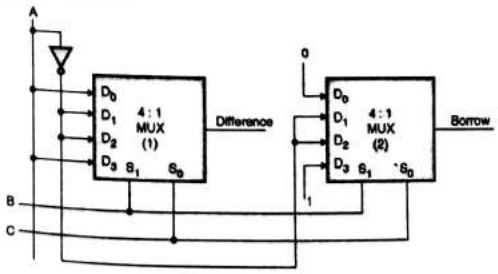
	D_0	D_1	D_2	D_3
$\bar{A}$	0	1	2	3
A	4	5	6	7
Input to Mux	A	$\bar{A}$	$\bar{A}$	A

Borrow B_0

	D_0	D_1	D_2	D_3
$\bar{A}$	0	1	2	3
A	4	5	6	7
Input to Mux	0	$\bar{A}$	$\bar{A}$	1

(C-3759)

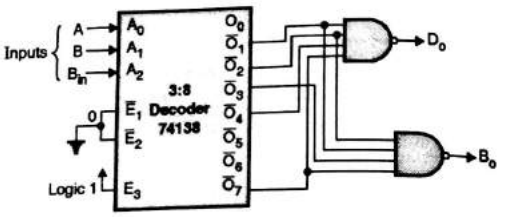
Implementation using 4 : 1 MUX :



(C-3760) **Fig. P. 6.14.14**

Scanned by CamScanner

Implementation using 3 : 8 Decoder :



(C-3761) **Fig. P. 6.14.14(a)**

6.14.3 Implementation of Boolean SOP Expression with Don't Care Conditions :

- Sometimes the Boolean expressions are given with don't care conditions.
- We know that the don't care conditions can be treated as logic 1's or 0's.
- Ex. 6.14.15 illustrates the use of multiplexer to implement such equations.

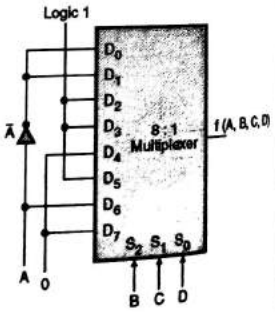
Ex. 6.14.15 : Implement the following Boolean expression using an 8 : 1 multiplexer.
$f(A, B, C, D) = \Sigma m(1, 3, 5, 10, 11, 13, 14) + d(0, 2)$.

Soln. :

The don't care conditions are assumed to be logic 1's. The design table is shown in Fig. P. 6.14.15(a) and the corresponding logic diagram is shown in Fig. P. 6.14.15(b).

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to Mux	$\overline{A}$	$\overline{A}$	1	1	0	1	A	0

(a) Design table



(b) Implementation

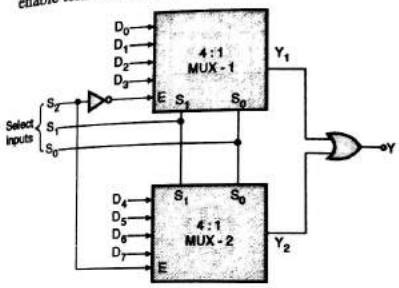
(C-445) **Fig. P. 6.14.15**

Ex. 6.14.16 : Design an 8 : 1 multiplexer using two 4 : 1 multiplexers. Explain with the help of the truth table. Implement the function $f(A, B, C) = \Sigma m(1, 3, 7)$ using the same.

Soln. :

For the design of 8 : 1 multiplexer using two 4 : 1 multiplexers,

- The cascading of two 4 : 1 multiplexer results in 8 : 1 multiplexer as shown in Fig. P. 6.14.16.
- There are in all eight data inputs (D_0 through D_7).
- The select lines S_1 and S_0 of both 4 : 1 multiplexers are connected in parallel whereas a third select input S_2 is used for enabling one multiplexer at a time.
- S_2 is connected directly to the enable (E) terminal of MUX-1 whereas $\overline{S}_2$ is connected to the enable terminal of MUX-2.



(C-424) **Fig. P. 6.14.16 : 8 : 1 multiplexer by cascading two 4 : 1 multiplexers**

Table P. 6.14.16 : Truth table

Select inputs			Output	
S_2	S_1	S_0	Y	
0	0	0	D_0	MUX 1 is enable
0	0	1	D_1	
0	1	0	D_2	
0	1	1	D_3	
1	0	0	D_4	MUX 2 is enable
1	0	1	D_5	
1	1	0	D_6	
1	1	1	D_7	

- The outputs of the two multiplexers are ORed to obtain the final output Y.

$$f(A, B, C) = \Sigma m(1, 3, 7)$$

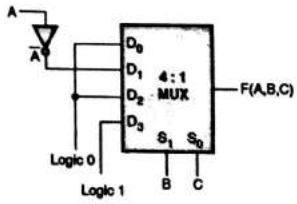
Step 1 : Write the design table :

The design table is as shown in Fig. 6.14.16(a).

	D_0	D_1	D_2	D_3	
$\overline{A}$	0	1	2	3	Row 1
A	4	5	6	7	Row 2
Input to MUX	0	$\overline{A}$	0	1	

(C-3597) **Fig. P. 6.14.16(a)**

Step 2 : Implementation using 4 : 1 MUX :



(C-3598) **Fig. P. 6.14.16(b) : Implementation using 4 : 1 MUX**

Ex. 6.14.17 : Implement the following expression using 8 : 1 MUX.
$f(A, B, C, D) = \Sigma m(0, 1, 5, 7, 9, 10, 15)$.

May 09, 10 Marks

Scanned by CamScanner

Soln. :

$$f(A, B, C, D) = \sum m(0, 1, 5, 7, 9, 10, 15)$$

Step 1 : Apply the variables B, C, D to select inputs :

The variables B, C, D are applied to the select lines S_2, S_1 and S_0 respectively.

Step 2 : Write the design table :

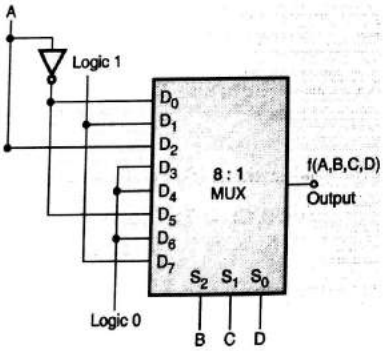
The design table is as shown in Table P. 6.14.17.

(C-3258) **Table P. 6.14.17**

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	$\bar{A}$	1	A	0	0	$\bar{A}$	0	1

The logic circuit is as shown in Fig. P. 6.14.17.

Step 3 : Realize the logic circuit



(C-2201) **Fig. P. 6.14.17**

Ex. 6.14.18 : Implement the following expression using 8 : 1 MUX.

$f(A, B, C, D) = \Sigma m(0, 1, 3, 5, 7, 10, 11, 13, 14, 15)$.

Dec. 09. 10 Marks. Dec. 14. 4 Marks

Soln. :

$$f(A, B, C, D) = \sum m(0, 1, 3, 5, 7, 10, 11, 13, 14, 15)$$

Step 1 : Apply the variables B, C and D to the select inputs :

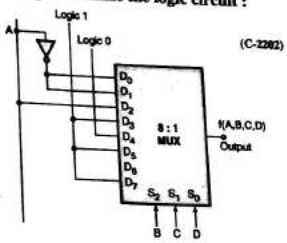
The three variables B, C and D are connected to the select lines S_2, S_1, and S_0 respectively.

Step 2 : Write the design table :

(C-3259)

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX :	$\bar{A}$	$\bar{A}$	A	1	0	1	A	1

Step 3 : Realize the logic circuit :



(C-2202)

Ex. 6.14.19 : Implement the following using only one 8:1 MUX and few gates.

$F(A, B, C, D) = \Sigma m(0, 3, 5, 7, 9, 13, 15)$

May 10. May 16. 5 Marks

Soln. :

$$F(A, B, C, D) = \sum m(0, 3, 5, 7, 9, 13, 15)$$

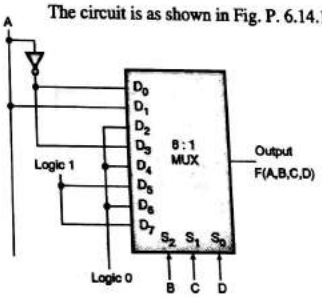
Step 1 : Write the design table :

(C-3260)

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	$\bar{A}$	A	0	$\bar{A}$	0	1	0	1

Step 2 : Implement using an 8 : 1 multiplexer :

The circuit is as shown in Fig. P. 6.14.19.



(C-3261) **Fig. P. 6.14.19**

Ex. 6.14.20 : Implement the following Boolean function using 4:1 MUX

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

May 10. 10 Marks. May 15. 8 Marks

Soln. :

$$F(A, B, C, D) = \sum m(0, 1, 2, 4, 6, 9, 12, 14)$$

Step 1 : Write the design table :

(C-3267)

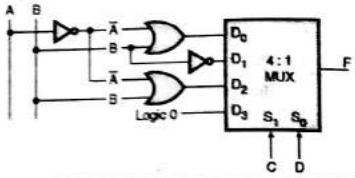
Inputs	D_0	D_1	D_2	D_3
$\bar{A}\bar{B}$	0	1	2	3
$\bar{A}B$	4	5	6	7
$A\bar{B}$	8	9	10	11
AB	12	13	14	15
				↓ 0

$$D_0 = \bar{A}\bar{B} + \bar{A}B + AB$$
$$= \bar{A}(\bar{B} + B) + AB$$
$$= \bar{A} + AB = \bar{A} + B$$
$$D_1 = \bar{A}\bar{B} + A\bar{B} = \bar{B}(\bar{A} + A) = \bar{B}$$
$$D_2 = \bar{A}\bar{B} + \bar{A}B + AB$$

Scanned by CamScanner

$$= \bar{A}(\bar{B} + B) + AB = \bar{A} + AB = \bar{A} + B = D_3 = 0$$

Step 2 : Implementation using 4 : 1 MUX : (C-3268)



Ex. 6.14.21 : Implement the following expression using single 4 : 1 MUX.

$f(A, B, C, D) = \sum m(2, 6, 8, 12, 13, 14)$. May 11, 10 Marks

Soln. :

Step 1 : Write the design table :

Inputs	D_0	D_1	D_2	D_3
$\bar{A}\bar{B}$	0	1	2	3
$\bar{A}B$	4	5	6	7
$A\bar{B}$	8	9	10	11
AB	12	13	14	15
MUX input	A	AB	$\bar{A}$+B	0

(C-3272) **Fig. P. 6.14.21**

$$D_0 = A\bar{B} + AB = A(B + \bar{B}) \quad \ldots(\because B + \bar{B} = 1)$$

$$D_0 = A$$

$$D_1 = AB$$

$$D_2 = \bar{A}\bar{B} + \bar{A}B + AB$$

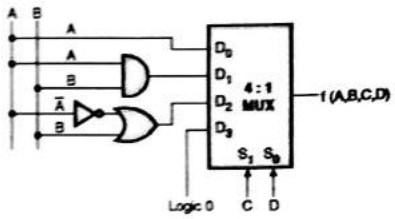
$$= \bar{A}(B + \bar{B}) + AB \quad \ldots(\because B + \bar{B} = 1)$$

$$= \bar{A} + AB \quad \ldots(\because \bar{A} + AB = \bar{A} + B)$$

$$D_2 = \bar{A} + B$$

and $D_3 = 0$

Step 2 : Implementation using 4 : 1 MUX :



(C-3273) **Fig. P. 6.14.21(a)**

Ex. 6.14.22 : Implement the following expression using 8 : 1 MUX.

$f(A, B, C, D) = \sum m(0, 1, 3, 6, 9, 11, 12, 13, 15)$. Dec. 11, 10 Marks

Soln. :

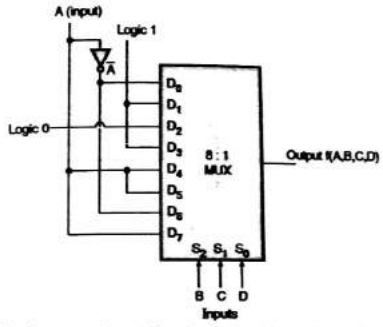
$$f(A, B, C, D) = \sum m(0, 1, 3, 6, 9, 11, 12, 13, 15)$$

Step 1 : Write the design table :

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	$\bar{A}$	1	0	1	A	A	$\bar{A}$	A

(C-3275)

Step 2 : Implement using an 8 : 1 multiplexer :



(C-3276) **Fig. P. 6.14.22 : Implementation of the given function using only one 8:1 multiplexer**

Ex. 6.14.23 : Implement the following using 8 : 1 MUX :

$F(A, B, C, D) = \sum m(0, 1, 2, 4, 6, 7, 8, 10, 14, 15)$. May 12, Dec. 12, 10 Marks

Soln. :

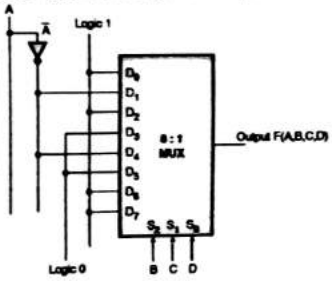
$$F(A, B, C, D) = \sum m(0, 1, 2, 4, 6, 7, 8, 10, 14, 15)$$

Step 1 : Write the design table :

(C-3783) **Table P. 6.14.23**

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	1	$\bar{A}$	1	0	$\bar{A}$	0	1	1

Step 2 : Implement using 8 : 1 multiplexer :



(C-3783) **Fig. P. 6.14.23 : Implementation**

Scanned by CamScanner

Ex. 6.14.24 : Implement the following logic function using all 4 : 1 multiplexers with select inputs as 'B', 'C', 'D', 'E' only

$$f(A, B, C, D, E) = \sum m(0, 1, 2, 3, 6, 8, 9, 10, 13, 15, 17, 20, 24, 30)$$

Dec. 13. 10 Marks

Soln. :

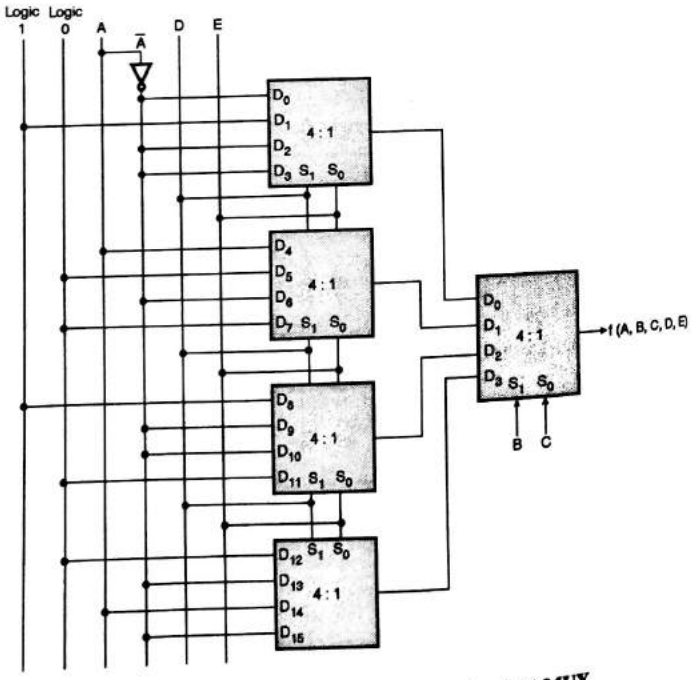
$$f(A, B, C, D, E) = \sum m(0, 1, 2, 3, 6, 8, 9, 10, 13, 15, 17, 20, 24, 30)$$

Step 1 : Write the design table :

Table P. 6.14.24 (C-4209)

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
$\bar{A}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Input to mux	$\bar{A}$	1	$\bar{A}$	$\bar{A}$	A	0	$\bar{A}$	0	1	$\bar{A}$	$\bar{A}$	0	0	$\bar{A}$	A	$\bar{A}$

Step 2 : Realize the logic circuit :



(C-4210) **Fig. P. 6.14.24 : Logic circuit using 4 : 1 MUX**

Ex. 6.14.25 : Implement the following using 8 : 1 Mux.

$$F(A, B, C, D) = \pi M(1, 3, 5, 9, 11, 12, 13).$$

May 14. 10 Marks

Soln. :

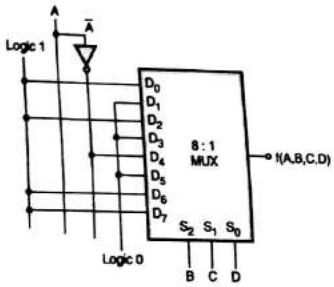
$$F(A, B, C, D) = \pi M(1, 3, 5, 9, 11, 12, 13)$$

Step 1 : Write the design table :

(C-4965) **Table P. 6.14.25 : Design table**

Inputs	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	1	0	1	0	$\bar{A}$	0	1	1

Step 2 : Implementation using 8 : 1 MUX :



(C-4966) **Fig. P. 6.14.25 : Implementation using 8 : 1 MUX**

Ex. 6.14.26 : What is multiplexer tree ? Construct 32 : 1 multiplexer using 8 : 1 multiplexers only. Explain how the logic on particular data line is steered to the output in this design with example ?

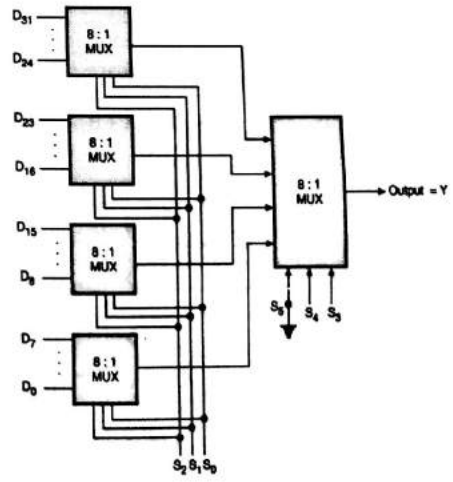
Dec. 15. 10 Marks

Soln. : For multiplexer tree refer section 6.13.

The truth table is as follows :

Table P. 6.14.26

Select inputs					Output
S_4	S_3	S_2	S_1	S_0	Y
0	0	0	0	0	D_0
⋮			⋮		⋮
0	0	1	1	1	D_7
0	1	0	0	0	D_8
⋮			⋮		⋮
0	1	1	1	1	D_{15}
1	0	0	0	0	D_{16}
⋮			⋮		⋮
1	0	1	1	1	D_{23}
1	1	0	0	0	D_{24}
⋮			⋮		⋮
1	1	1	1	1	D_{31}



(C-5272) **Fig. P. 6.14.26 : 32 : 1 MUX using only 8 : 1 multiplexer**

Scanned by CamScanner

Ex. 6.14.27 : Implement the following function using 4 : 1 multiplexer and few gates.

$F(A, B, C, D) = \Sigma m(0, 1, 2, 3, 6, 7, 9, 10, 13, 15)$ Dec. 16, 8 Marks

Soln. :

Step 1 : Write the design table :

	D_0	D_1	D_2	D_3
$\bar{A}\bar{B}$	0	1	2	3
$\bar{A}B$	4	5	6	7
$A\bar{B}$	8	9	10	11
AB	12	13	14	15
	$\bar{A}\bar{B}$			

(C-5721)

$$D_0 = \bar{A}\,\bar{B}$$

$$D_1 = \bar{A}\,\bar{B} + A\bar{B} + AB = \bar{B}(A + \bar{A}) + AB$$

$$= \bar{B} + AB \ldots (\because A + \bar{A} = 1)$$

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

$$D_2 = \bar{A}\bar{B} + \bar{A}B + A\bar{B}$$

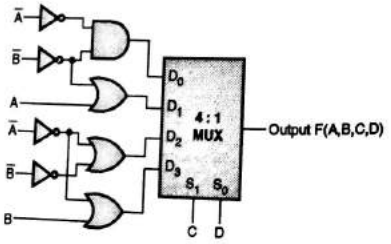
$$= \bar{A}(B + \bar{B}) + A\bar{B} \quad (\because B + \bar{B} = 1)$$

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

$$D_3 = \bar{A}\bar{B} + \bar{A}B + AB = \bar{A}(B + \bar{B}) + AB$$

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

Step 2 : Implementation using 4 : 1 MUX :



(C-5722) **Fig. P. 6.14.27 : Implementation using 4 : 1 MUX**

.15 Demultiplexers :

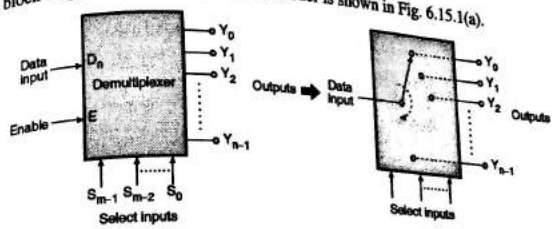
MU : May 04, Dec. 08, Dec. 10

niversity Questions

1 Write short note on : Demultiplexer. **(May 04, Dec. 08, Dec. 10, 2 Marks)**

6.15.1 Demultiplexer Principle :

- The block diagram of a demultiplexer or decoder is shown in Fig. 6.15.1(a).



(a) 1 : n demultiplexer **(b) Equivalent circuit**

(C-447) **Fig. 6.15.1**

- It has only one input, "n" outputs, and "m" select inputs.
- A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- At a time only one output line is selected by the select lines and the input is routed to the selected output line.
- Hence a demultiplexer is equivalent to a single pole multiple way switch as shown in Fig. 6.15.1(b). The enable input will enable the demultiplexer. If the enable (E) input is not active, then the demultiplexer does not work.
- The relation between the n output lines and m select lines is as follows :

$$n = 2^m$$

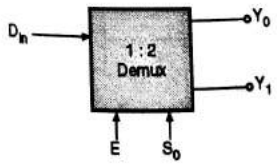
6.16 Types of Demultiplexers :

Similar to the multiplexers, the demultiplexers are classified as follows :

1. 1 : 2 demultiplexer
2. 1 : 4 demultiplexer
3. 1 : 8 demultiplexer
4. 1 : 16 demultiplexer

6.16.1 1 : 2 Demultiplexer :

- The block diagram of 1 : 2 demultiplexer is shown in Fig. 6.16.1. It has one data input D_{in}, one select input S_0, one enable (E) input and two outputs Y_0 and Y_1.
- D_{in} is connected to Y_0 if $S_0 = 0$ and $E = 1$. Similarly D_{in} is connected to Y_1 if $S_0 = 1$ and $E = 1$.
- If $E = 0$, then both the outputs will be 0 irrespective of the inputs, because the DEMUX is disabled.
- The truth table of the 1 : 2 demultiplexer is as follows :



(C-448) **Fig. 6.16.1**

Table 6.16.1 : Truth table of demux 1 : 2

Enable	Select	Outputs	
E	S_0	Y_1	Y_0
0	X	0	0
1	0	0	D_{in}
1	1	D_{in}	0

Scanned by CamScanner

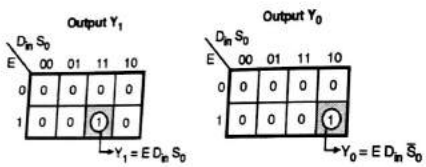
Realization of 1 : 2 demux :

Step 1 : To realize the 1 : 2 demultiplexer using gates we have to write the truth table as given in Table 6.16.2 :

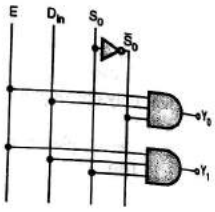
Table 6.16.2 : Detail truth table of demux 1 : 2

Enable	Data	Select	Outputs		
E	D_{in}	S_0	Y_1	Y_0	
0	X	X	0	0	
1	0	0	0	0	Y_0 is connected to D_{in}
1	1	0	0	1	
1	0	1	0	0	Y_1 is connected to D_{in}
1	1	1	1	0	

Step 2 : Write the K maps : The K maps for the two outputs are as follows :



(C-449) **Fig. 6.16.1(a) : K maps**



(C-450) **Fig. 6.16.2 : 1 : 2 demux using gates**

- Thus the expressions for Y_0 and Y_1 are as follows :

$$Y_0 = ED_{in}\overline{S_0} \quad \text{and} \quad Y_1 = E\,D_{in}\,S_0$$

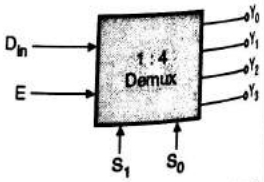
- The 1 : 2 demux using gates is shown in Fig. 6.16.2.

6.16.2 1 : 4 Demultiplexer :

- The 1 : 4 demultiplexer is shown in Fig. 6.16.3 and its truth table is given in Table 6.16.3.

Table 6.16.3 : Truth table for 1 : 4 demultiplexer

		Inputs		Outputs				
E	D_{in}	S_1	S_0	Y_0	Y_1	Y_2	Y_3	
1	0	0	0	0	0	0	0	Y_0 is connected to D_{in}
1	1	0	0	1	0	0	0	
1	0	0	1	0	0	0	0	Y_1 is connected to D_{in}
1	1	0	1	0	1	0	0	
1	0	1	0	0	0	0	0	Y_2 is connected to D_{in}
1	1	1	0	0	0	1	0	
1	0	1	1	0	0	0	0	Y_3 is connected to D_{in}
1	1	1	1	0	0	0	1	

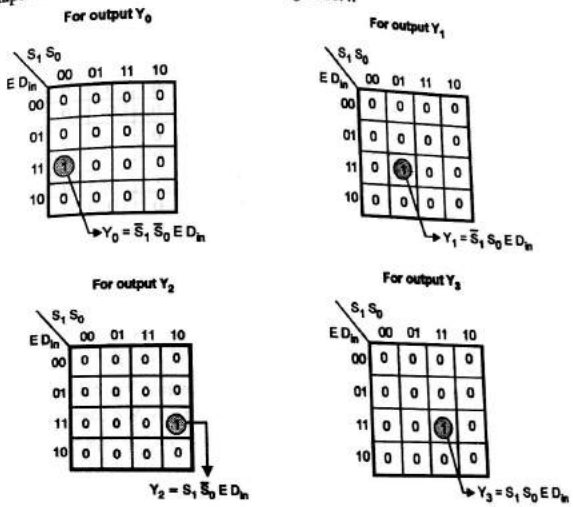


(C-451) **Fig. 6.16.3 : Block diagram of 1 : 4 demultiplexer**

- D_{in} is connected to Y_0 when $S_1S_0 = 00$, it is connected to Y_1 when $S_1S_0 = 01$ and so on. The other outputs will remain zero.
- The enable input needs to be high in order to enable the demux. If E = 0 then all the outputs will be low irrespective of everything.

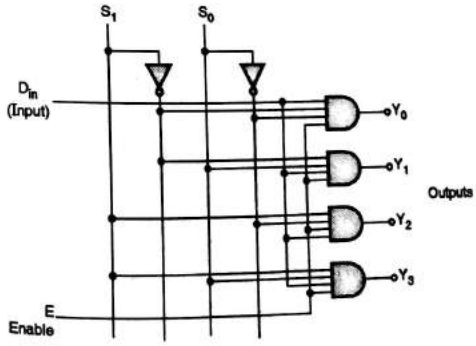
K-maps :

The K-maps for the four outputs are shown in Fig. 6.16.4.



(C-452) **Fig. 6.16.4 : K-maps for various outputs**

Implementation : The 1 : 4 demux is implemented as shown in Fig. 6.16.5.

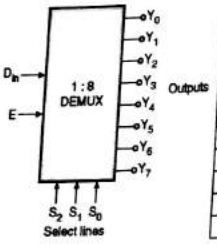


(C-453) **Fig. 6.16.5 : 1:4 demultiplexer**

Scanned by CamScanner

6.16.3 1 : 8 Demultiplexer :

- The block diagram of 1 : 8 demux is shown in Fig. 6.16.6(a).
- It has one data input, eight outputs, three select inputs and an enable input E.
- The truth table is shown in Fig. 6.16.6(b).



(C-454) **(a) 1 : 8 demux**

Enable	Select			Outputs							
E	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	D_{in}
1	0	0	1	0	0	0	0	0	0	D_{in}	0
1	0	1	0	0	0	0	0	0	D_{in}	0	0
1	0	1	1	0	0	0	0	D_{in}	0	0	0
1	1	0	0	0	0	0	D_{in}	0	0	0	0
1	1	0	1	0	0	D_{in}	0	0	0	0	0
1	1	1	0	0	D_{in}	0	0	0	0	0	0
1	1	1	1	D_{in}	0	0	0	0	0	0	0

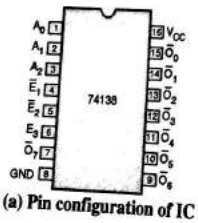
(b) Truth table for 1 : 8 demux

Fig. 6.16.6

- Depending on the combination of the select inputs S_2 S_1 S_0, the data input D_{in} is connected to one of the eight outputs.
- For example if S_2 S_1 S_0 = 1 1 0 then D_{in} is connected to output Y_6.

6.16.4 IC 74138 as 1 : 8 DE-MUX :

- The pin configuration of the 3 : 8 decoder IC 74138 is shown in Fig. 6.16.7.
- A_0 , A_1 , A_2 are the three address lines or the three input lines. We have to apply the 3-bit binary data to these inputs. But when this IC is to be used as a 1 : 8 DEMUX we have to use these lines as the three select lines.
- $\bar{O}_0$ to $\bar{O}_7$ are the 8-output active low lines.
- There are three enable inputs out of which $\bar{E}_1$ and $\bar{E}_2$ are the active low enable inputs whereas E_3 is an active high enable input.
- We have to make $\bar{E}_1 = \bar{E}_2 = 0$ and $E_3 = 1$ in order to enable the IC.
- The truth table for IC 74138 is shown in Table 6.16.4.



(a) Pin configuration of IC 74138

Pin names	Description
$A_0 - A_2$	Address inputs (Select lines)
$\bar{E}_1 - \bar{E}_2$	Enable inputs (Active Low)
E_3	Enable Input (Active HIGH)
$\bar{O}_0 - \bar{O}_7$	Outputs (Active Low)

(b) Pin names and description

(C-489) **Fig. 6.16.7**

Features of IC 74138 :

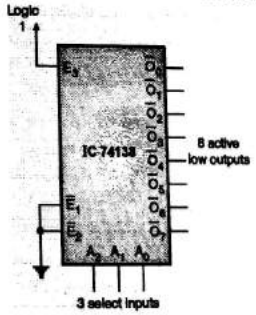
- Schottky process for achieving a high speed.
- Multiple enable inputs ensure easy expansion.
- It can work as a decoder or as a demultiplexer.
- Active low, mutually exclusive outputs.

Table 6.16.4 : Truth table of 74138

Inputs						Outputs								
$\bar{E}_1$	$\bar{E}_2$	E_3	A_0	A_1	A_2	$\bar{O}_0$	$\bar{O}_1$	$\bar{O}_2$	$\bar{O}_3$	$\bar{O}_4$	$\bar{O}_5$	$\bar{O}_6$	$\bar{O}_7$	
H	×	×	×	×	×	H	H	H	H	H	H	H	H	Decoder is disabled
×	H	×	×	×	×	H	H	H	H	H	H	H	H	
×	×	L	×	×	×	H	H	H	H	H	H	H	H	
L	L	H	L	L	L	L	H	H	H	H	H	H	H	
L	L	H	H	L	L	H	L	H	H	H	H	H	H	Decoder is enabled
L	L	H	L	H	L	H	H	L	H	H	H	H	H	
L	L	H	H	H	L	H	H	H	L	H	H	H	H	
L	L	H	L	L	H	H	H	H	H	L	H	H	H	
L	L	H	H	L	H	H	H	H	H	H	L	H	H	
L	L	H	L	H	H	H	H	H	H	H	H	L	H	
L	L	H	H	H	H	H	H	H	H	H	H	H	L	

A_0 = LSB, A_2 = MSB, H = HIGH Voltage Level, L = LOW Voltage Level × = Don't care condition.

- The connection diagram of IC 74138 as 1:8 DEMUX is as shown in Fig. 6.16.8.



(C-3580) **Fig. 6.16.8 : IC74138 as 1 : 8 DEMUX**

6.17 Demultiplexer Tree :

- Similar to multiplexer we can construct the demultiplexer having more number of lines using demultiplexers having less number lines.
- This is called as demultiplexer tree. It is also called as cascading of demultiplexers.
- This concept will be clear by solving the following examples.

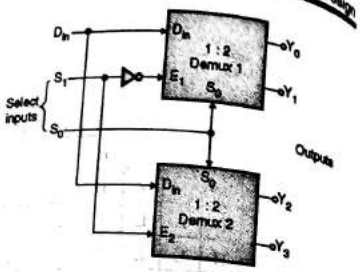
Ex. 6.17.1 : Obtain a 1 : 4 line demux using 1 : 2 demultiplexers.

Scanned by CamScanner

Soln. :

Refer Fig. P. 6.17.1 for the solution.

- The select lines S_0 of both the 1 : 2 demultiplexers are connected together and brought out as S_0 line of the 1 : 4 demux.
- The second select line S_1 is connected directly to the enable (E_2) input of demux 2 whereas inverted S_1 is connected to the enable input of demux – 1.



(C-459) **Fig. P. 6.17.1 : 1 : 4 demultiplexer using 1 : 2 demultiplexers**

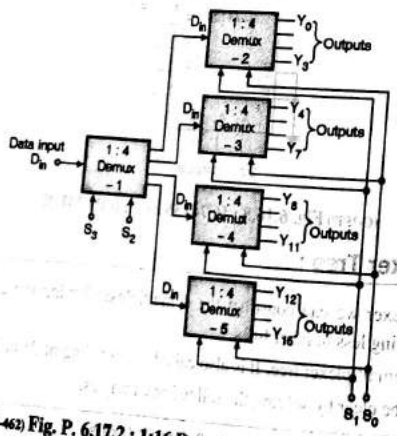
- The truth table of the circuit is shown in the Table P. 6.17.1.

Table P. 6.17.1 : Truth table of 1 : 4 demux using two 1 : 2 demux

Select inputs		Outputs				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	
0	0	0	0	0	D_{in}	Demux – 1 enabled
0	1	0	0	D_{in}	0	
1	0	0	D_{in}	0	0	Demux – 2 enabled
1	1	D_{in}	0	0	0	

Ex. 6.17.2 : Draw the 1:16 demux using 1:4 demultiplexers. The 1:16 demux using 1:4 demultiplexer is shown in Fig. P. 6.17.2.

Dec. 09, Dec. 10, 4 Marks



(C-462) **Fig. P. 6.17.2 : 1:16 Demux using 1:4 demux**

Soln. : In all five 1:4 Demux ICs are used. $S_0 S_1 S_2 S_3$ are the four select lines. D_{in} is the data input and Y_0 .... Y_{15} are the 16 outputs of the 1:16 demux. The truth table of 1:16 Demux is shown in Table P. 6.17.2.

Table P. 6.17.2

Select inputs				Output	
S_3	S_2	S_1	S_0		
0	0	0	0	$Y_0 = D_{in}$	$S_3 S_2 = 00$
⋮		⋮		⋮	∴ Demux 2 is selected
0	0	1	1	$Y_3 = D_{in}$	
0	1	0	0	$Y_4 = D_{in}$	$S_3 S_2 = 01$
⋮		⋮		⋮	∴ Demux 3 is selected
0	1	1	1	$Y_7 = D_{in}$	
1	0	0	0	$Y_8 = D_{in}$	$S_3 S_2 = 10$
⋮		⋮		⋮	∴ Demux 4 is selected
1	0	1	1	$Y_{11} = D_{in}$	
1	1	0	0	$Y_{12} = D_{in}$	$S_3 S_2 = 11$
⋮		⋮		⋮	∴ Demux 5 is selected
1	1	1	1	$Y_{15} = D_{in}$	

Ex. 6.17.3 : Design 1 : 28 line Demux using 1 : 8 Demux.

May 11, 10 Marks

Soln. :

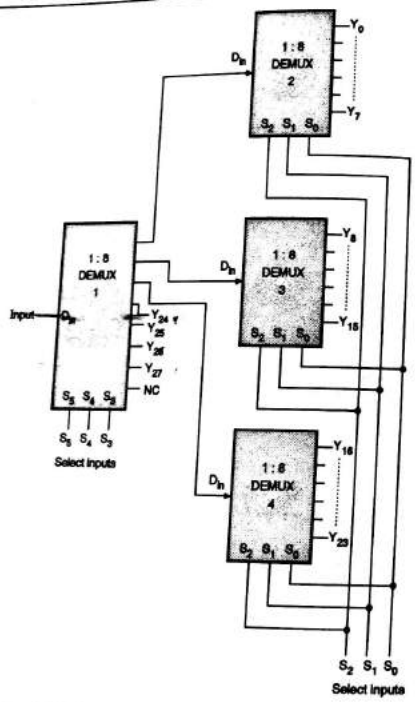
The circuit diagram of a 1:28 DEMUX using 1:8 DEMUX is shown in Fig. P. 6.17.3. We have used four DEMUX ICs. $S_0 S_1 S_2 S_3 S_4 S_5$ are the six select lines. D_{in} is the input and Y_0 to Y_{27} are the 28 output lines. The truth table of 1:28 DEMUX is as shown in Table P. 6.17.3.

Table P. 6.17.3

Select Inputs						Output	Comment
S_5	S_4	S_3	S_2	S_1	S_0		
0	0	0	0	0	0	$Y_0 = D_{in}$	$S_5 S_4 S_3 = 000$
0	0	0	0	0	1	$Y_1 = D_{in}$	Hence DEMUX-2 is selected.
		⋮				⋮	
0	0	0	1	1	1	$Y_7 = D_{in}$	
0	0	1	0	0	0	$Y_8 = D_{in}$	$S_5 S_4 S_3 = 001$
0	0	1	0	0	1	$Y_9 = D_{in}$	Hence DEMUX-3 is selected.
		⋮				⋮	
0	0	1	1	1	1	$Y_{15} = D_{in}$	
0	1	0	0	0	0	$Y_{16} = D_{in}$	$S_5 S_4 S_3 = 010$
0	1	0	0	0	1	$Y_{17} = D_{in}$	Hence DEMUX-4 is selected.
		⋮				⋮	
0	1	0	1	1	1	$Y_{23} = D_{in}$	

Scanned by CamScanner

Select Inputs						Output	Comment
S_5	S_4	S_3	S_2	S_1	S_0		
0	1	1	X	X	X	$Y_{24} = D_{in}$	$S_5\ S_4\ S_3 = 011$ to 111 DEMUX-1 is selected
1	0	0	X	X	X	$Y_{25} = D_{in}$	
1	0	1	X	X	X	$Y_{26} = D_{in}$	
1	1	0	X	X	X	$Y_{27} = D_{in}$	
1	1	1	X	X	X	Y_{28} NOT USED	



(C-3274) **Fig. P. 6.17.3 : 1:28 DEMUX using 1:8 MUX**

6.17.1 Comparison of Multiplexer and Demultiplexer :

Table 6.17.1 : Comparison of MUX and DEMUX

Sr. No.	Parameter	Multiplexer	Demultiplexer
1.	Type of logic circuit	Combinational	Combinational
2.	Number of data inputs	n	1
3.	Number of select inputs	m	m
4.	Number of data output	1	n

Sr. No.	Parameter	Multiplexer	Demultiplexer
5.	Relation between input / output lines and select lines	$n = 2^m$	$n = 2^m$
6.	Operation principle	Many to 1 or as data selector	1 to many or data distributor
7.	Application	• As a universal logic circuit we can implement any combinational circuit using a MUX. • In time division multiplexing at the sending end.	• We can implement some combinational circuit. • In TDM system at receiving end.

6.17.2 Use of DEMUX in Combinational Logic Design :

Like multiplexers, we can use the demultiplexers for designing the combinational circuits. The following examples will demonstrate this concept.

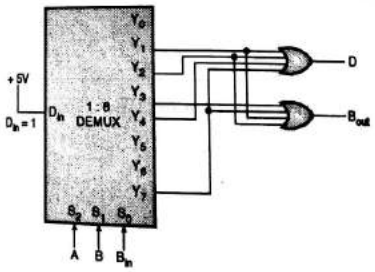
Ex. 6.17.4 : Implement the full subtractor using a 1 : 8 demultiplexer. May 16, 5 Marks

Soln. :

- The full subtractor has three inputs A, B and B_{in} and two outputs namely difference D and borrow out (B_{out}).
- The truth table of a full subtractor is as follows :

Table P. 6.17.4 : Truth table of a full subtractor

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



(C-464) **Fig. P. 6.17.4 : Full subtractor using 1 : 8 demux**

- From the truth table we can express the difference and borrow out outputs in the standard SOP forms as,

$$D = f(A, B, B_{in}) = \Sigma m(1, 2, 4, 7)$$

$$\text{and} \quad B_{out} = f(A, B, B_{in}) = \Sigma m(1, 2, 3, 7)$$

- Connect D_{in} to logic 1 permanently and connect A, B and B_{in} to the select inputs S_2, S_1, S_0 respectively as shown in Fig. P. 6.17.4.
- As D_{in} is connected to 1, we get the required minterms at the Demux outputs. We have to OR the required minterms to obtain the D and B_{out} outputs as shown in Fig. P. 6.17.4.

Note : We can use IC 74138 as 1 : 8 DEMUX to implement the Boolean expression with 1 : 8 DEMUX.

Scanned by CamScanner

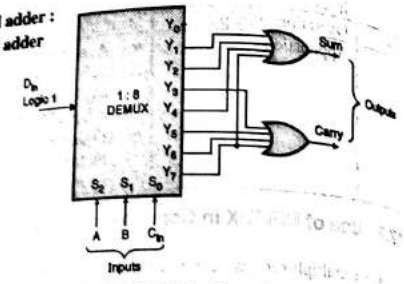
Ex. 6.17.5 : Implement a full adder using demultiplexer.

Soln. :

Step 1 : Write the truth table of the full adder :

Table P. 6.17.5 : Truth table of a full adder

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



(C-465) **Fig. P. 6.17.5 : Full adder using a demultiplexer**

Step 2 : Write expressions for sum and carry :

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

$\therefore$ Outputs Y_1, Y_2, Y_4 and Y_7 should be ORed to get sum output.

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

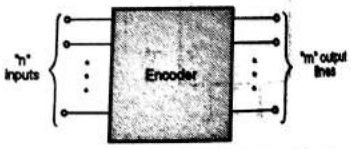
$\therefore$ Outputs Y_3, Y_5, Y_6 and Y_7 should be ORed to get the carry output.

Step 3 : Implementation using a 1 : 8 demultiplexer :

Fig. P. 6.17.5 shows the implementation.

6.18 Encoders :

- Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder.
- An encoder has "n" number of input lines and "m" number of output lines.
- An encoder produces an m bit binary code corresponding to the n bit digital number, applied at its input.
- Block diagram of an encoder is shown in Fig. 6.18.1.



(C-466) **Fig. 6.18.1 : Block diagram of an encoder**

- The encoder accepts an n input digital word and converts it into an m bit another digital word. For example a BCD number applied at the input can be converted into a binary number at the output.
- The internal combinational circuit of the encoder is designed accordingly.

6.18.1 Types of Encoders :

The types of encoders which we are going to discuss are as follows :

1. Priority encoders.
2. Decimal to BCD encoder.
3. Octal to binary encoder.
4. Hexadecimal to binary encoder.

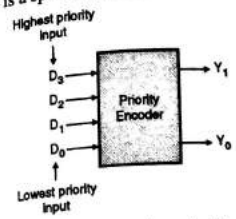
6.19 Priority Encoder :

MU : Dec. 05, May 07, May 08, Dec. 09, May 11

University Questions

Q. 1 Write short notes on priority encoder. **(Dec. 05, May 07, May 08, Dec. 09, May 11, 5 Marks)**

- This is a special type of encoder.



(a) **Block diagram of a priority encoder**

Highest Inputs			Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_1	Y_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

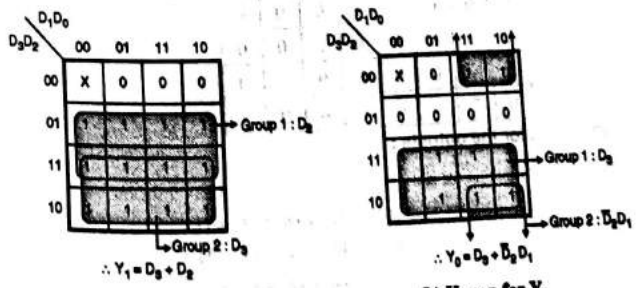
(b) **Truth table of a priority encoder**

(C-467) **Fig. 6.19.1**

- Priorities are given to the input lines. If two or more input lines are "1" at the same time, then the input line with highest priority will be considered.
- The block diagram of a priority encoder is shown in Fig. 6.19.1(a) and its truth table is shown in Fig. 6.19.1(b).
- There are four inputs, D_0 through D_3 and two outputs Y_1 and Y_0. Out of the four inputs D_3 has the highest priority and D_0 has the lowest priority.
- That means if $D_3 = 1$ then $Y_1 Y_0 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then $Y_1 Y_0 = 10$ irrespective of the other inputs.
- Carefully go through the truth table shown in Fig. 6.19.1(b) to get the feel of priority encoder operation.

K-maps for the outputs :

The K-maps for the two outputs are as shown in Fig. 6.19.2.



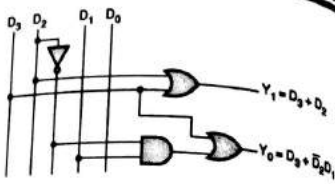
(a) **K map for Y_1** (b) **K map for Y_0**

(C-468) **Fig. 6.19.2**

Scanned by CamScanner

Implementation :

The logic diagram for the priority encoder is shown in Fig. 6.19.3.



(C-469) **Fig. 6.19.3 : Logic diagram of a priority encoder**

6.19.1 Octal to Binary Encoder :

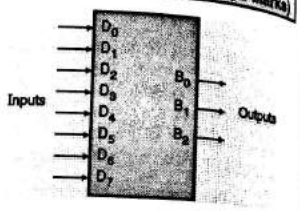
MU : Dec. 08

University Questions

2. 1 Write short note on : Octal to binary encoder. **(Dec. 08, 5 Marks)**

- The octal to binary encoder has 8 – input lines and 3 – output lines. Corresponding to the eight input octal numbers we get three bit binary output.
- Note that in encoders only one input will have a one value at any given time.

Fig. 6.19.4 shows the block diagram of octal to binary encoder and Table 6.19.1 gives its truth table.



(C-474) **Fig. 6.19.4 : Block diagram of octal to binary encoder**

Table 6.19.1 : Truth table of octal to binary encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	B_0	B_1	B_2
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

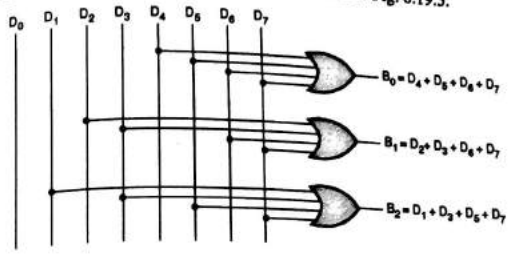
gical expressions for the outputs :

Referring to the truth table we can write the logical expressions for the outputs as follows :

$$B_0 = D_4 + D_5 + D_6 + D_7$$
$$B_1 = D_2 + D_3 + D_6 + D_7$$
$$B_2 = D_1 + D_3 + D_5 + D_7$$

Implementation using basic gates :

The octal to binary encoder using the basic gates is shown in Fig. 6.19.5.



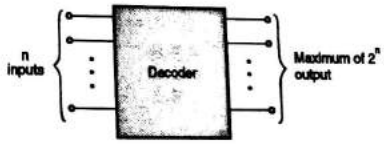
(C-475) **Fig. 6.19.5 : Octal to binary encoder**

6.20 Decoder :

- A decoder is a combinational circuit.
- Fig. 6.20.1 shows the block diagram of a decoder. It has "n" inputs and to a maximum 2^n outputs.
- Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Typical applications :

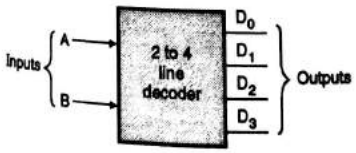
1. Code converters
2. BCD to seven segment decoders.
3. Nixie tube decoders.
4. Relay actuators.



(C-479) **Fig. 6.20.1 : Block diagram of a decoder**

6.20.1 2 to 4 Line Decoder :

- The block diagram of a 2 to 4 line decoder is shown in Fig. 6.20.2(a). A and B are the two inputs whereas D_0 through D_3 are the four outputs.
- Table 6.20.1 shows the truth table which explains the operation of the decoder. It shows that each output is "1" for only a specific combination of inputs.

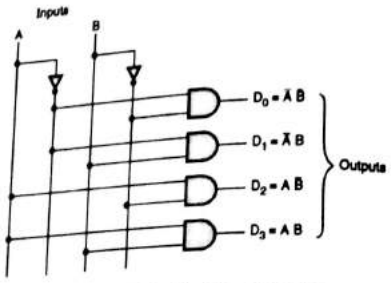


(C-480) **Fig. 6.20.2(a) : Block diagram of a 2 line to 4 line decoder**

Table 6.20.1 : Truth table of a 2 line to 4 line decoder

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

Scanned by CamScanner



(C-481) **Fig. 6.20.2(b) : 2 to 4 line decoder**

- The Boolean expressions for the four outputs are,

$$D_0 = \bar{A}\,\bar{B},\quad D_1 = \bar{A}\,B$$

$$D_2 = A\,\bar{B}, \text{ and } D_3 = A\,B$$

- The logic diagram for a 2 to 4 line decoder is shown in Fig. 6.20.2(b).

6.20.2 Difference Between Decoder and Demultiplexer :

Sr. No.	Parameter	Demux	Decoder
1.	Block schematic	See Fig. A	See Fig. B.
2.	Number of data inputs	One	More than 1.
3.	Select inputs	Present	Absent
4.	Applications	As a distributor switch, to implement Boolean expressions.	BCD to seven segment decoder, Decimal to BCD, Binary to Hex etc.

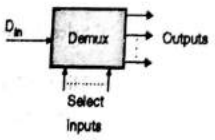
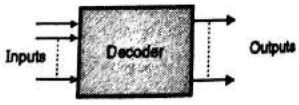


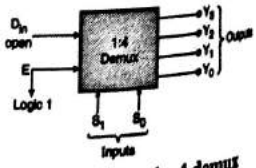
Fig. A



(C-482) **Fig. B**

6.20.3 Demultiplexer as Decoder :

- We can use a demultiplexer as a decoder.
- Let us see how to operate a 1 : 4 demux as 2 : 4 decoder.
- Consider Fig. 6.20.3 which shows a 1 : 4 demultiplexer.
- D_{in} is the data input, S_1 S_0 are the select lines and Y_3 through Y_0 are the outputs.



(C-483) **Fig. 6.20.3 : 1 : 4 demux as 2 : 4 decoder**

- In order to operate it as 2 : 4 decoder, we have to use S_1 S_0 as inputs, keep D_{in} open and use Y_3 to Y_0 as outputs as shown in Fig. 6.20.3.

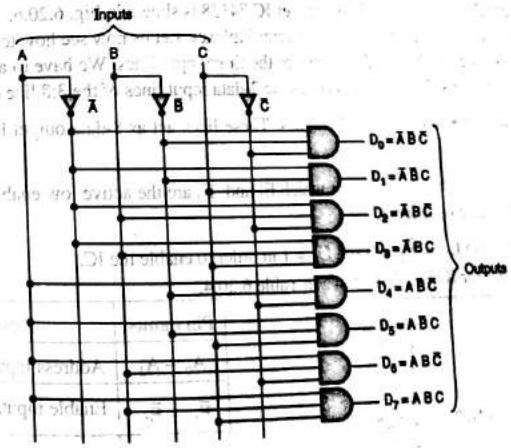
6.20.4 3 to 8 Line Decoder :

- The truth table of 3 to 8 line decoder is shown in Table 6.20.2.

Table 6.20.2 : Truth table for 3 to 8 line decoder

Inputs			Outputs								
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
0	0	0	1	0	0	0	0	0	0	0	$D_0 = \bar{A}\,\bar{B}\,\bar{C}$
0	0	1	0	1	0	0	0	0	0	0	$D_1 = \bar{A}\,\bar{B}\,C$
0	1	0	0	0	1	0	0	0	0	0	$D_2 = \bar{A}\,B\,\bar{C}$
0	1	1	0	0	0	1	0	0	0	0	$D_3 = \bar{A}\,B\,C$
1	0	0	0	0	0	0	1	0	0	0	$D_4 = A\,\bar{B}\,\bar{C}$
1	0	1	0	0	0	0	0	1	0	0	$D_5 = A\,\bar{B}\,C$
1	1	0	0	0	0	0	0	0	1	0	$D_6 = A\,B\,\bar{C}$
1	1	1	0	0	0	0	0	0	0	1	$D_7 = A\,B\,C$

- The logic diagram for 3 to 8 line decoder is shown in Fig. 6.20.4.



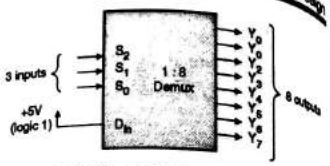
(C-484) **Fig. 6.20.4 : 3 to 8 line decoder**

6.20.5 1 : 8 DEMUX Operated as 3:8 Decoder :

- In order to operate 1 : 8 Demux as a 3:8 line decoder, the connections are to be made as sh Fig. 6.20.5.

Scanned by CamScanner

- The data input D_{in} is connected to logic 1 permanently. The three select inputs S_0, S_1 and S_2 will act as three input lines of the decoder and Y_0 to Y_7 are the 8-output lines.
- The truth table of this circuit is as given in Table 6.20.3 which shows that the circuit works as a 3 : 8 decoder.



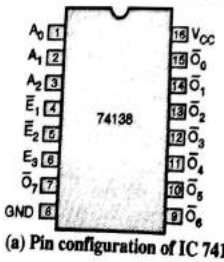
(C-485) **Fig. 6.20.5 : Use of 1:8 demux as 3 : 8 decoder**

Table 6.20.3 : Truth table

D_{in}	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

6.20.6 IC 74138 as 3 : 8 Decoder :

- The pin configuration of the 3 : 8 decoder IC 74138 is shown in Fig. 6.20.6.
- We have already discussed this IC as a demultiplexer. Let us now see how to use it as a decoder.
- A_0, A_1, A_2 are the three address lines or the three input lines. We have to apply the 3-bit binary data to these inputs. So these lines act as the 3-data input lines of the 3:8 line decoder.
- $\bar{O}_0$ to $\bar{O}_7$ are the 8-output active low lines. These lines act as 8-data output lines of the 3 : 8 line decoder.
- There are three enable inputs out of which $\bar{E}_1$ and $\bar{E}_2$ are the active low enable inputs whereas E_3 is an active high enable input.
- We have to make $\bar{E}_1 = \bar{E}_2 = 0$ and $E_3 = 1$ in order to enable the IC.
- The truth table for IC 74138 is shown in Table 6.20.4.



(a) Pin configuration of IC 74138

Pin names	Description
$A_0 - A_2$	Address inputs (Select lines)
$\bar{E}_1 - \bar{E}_2$	Enable inputs (Active Low)
E_3	Enable Input (Active HIGH)
$\bar{O}_0 - \bar{O}_7$	Outputs (Active Low)

(b) Pin names and description

(C-489) **Fig. 6.20.6**

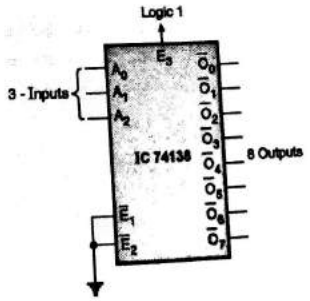
Table 6.20.4 : Truth table of 74138

Inputs						Outputs								
$\bar{E}_1$	$\bar{E}_2$	E_3	A_0	A_1	A_2	$\bar{O}_0$	$\bar{O}_1$	$\bar{O}_2$	$\bar{O}_3$	$\bar{O}_4$	$\bar{O}_5$	$\bar{O}_6$	$\bar{O}_7$	
H	×	×	×	×	×	H	H	H	H	H	H	H	H	Decoder is disabled
×	H	×	×	×	×	H	H	H	H	H	H	H	H	
×	×	L	×	×	×	H	H	H	H	H	H	H	H	
L	L	H	L	L	L	L	H	H	H	H	H	H	H	
L	L	H	H	L	L	H	L	H	H	H	H	H	H	Decoder is enabled
L	L	H	L	H	L	H	H	L	H	H	H	H	H	
L	L	H	H	H	L	H	H	H	L	H	H	H	H	
L	L	H	L	L	H	H	H	H	H	L	H	H	H	
L	L	H	H	L	H	H	H	H	H	H	L	H	H	
L	L	H	L	H	H	H	H	H	H	H	H	L	H	
L	L	H	H	H	H	H	H	H	H	H	H	H	L	

A_0 = LSB, A_2 = MSB, H = HIGH Voltage Level, L = LOW Voltage Level

× = Don't care condition

Fig. 6.20.7 shows the connection diagram for IC 74138 used as a 3 : 8 decoder.



(C-3581) **Fig. 6.20.7 : IC 74138 connected as 3:8 decoder**

6.20.7 Implementation of Boolean Function using Decoder :

A combination of decoder and gates can be used to implement that Boolean equations. This is demonstrated in the following example :

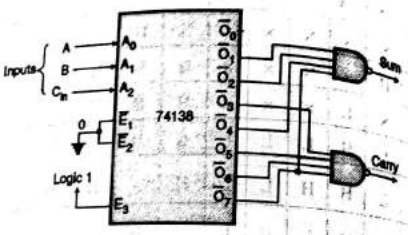
Scanned by CamScanner

Ex. 6.20.1 : Design a full adder using 3 : 8 decoder IC 74138. **Dec. 13, 5 Marks**

Soln. :

Table P. 6.20.1 : Truth table of full adder

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



(C-491) **Fig. P. 6.20.1 : Full adder implemented using IC 74138**

- The truth table of a full adder is shown in Table P. 6.20.1.
- The A, B and C_{in} inputs are applied to the A_0, A_1 and A_2 inputs of the decoder.
- The outputs of 74138 are active low. Hence we should apply $\bar{O}_1$, $\bar{O}_2$, $\bar{O}_4$ and $\bar{O}_7$ to a NAND gate as shown in Fig. P. 6.20.1 to obtain the sum output.
- Similarly outputs $\bar{O}_3$, $\bar{O}_5$, $\bar{O}_6$ and $\bar{O}_7$ to another NAND gate to obtain the carry output.
- Implementation of full adder is shown in Fig. P. 6.20.1.

All the enable terminals are connected to their respective active levels to enable the IC.

Ex. 6.20.2 : Implement Gray to Binary code converter using suitable decoder. **Dec. 08, 10 Marks**

Soln. :

Step 1 : Truth table for 3-bit Gray to Binary code converter :

Decimal	Gray Code Input			Binary Output		
	G_2	G_1	G_0	B_2	B_1	B_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	1	0	1	0
3	0	1	0	0	1	1
4	1	1	0	1	0	0
5	1	1	1	1	0	1
6	1	0	1	1	1	0
7	1	0	0	1	1	1

Step 2 : To obtain the output expressions for B_2, B_1 and B_0 :

$$B_2 = O_4 + O_5 + O_6 + O_7$$
$$B_1 = O_2 + O_3 + O_6 + O_7$$
$$B_0 = O_1 + O_3 + O_5 + O_7$$

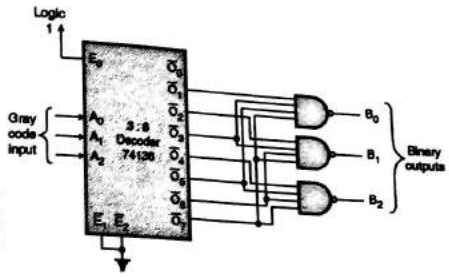
Step 3 : Implementation of gray to binary code converter using 3 : 8 decoder IC 74138 :

Since the outputs of IC74138 are active low, the output expressions obtained are modified as follows.

$$B_2 = \overline{\overline{O_4 + O_5 + O_6 + O_7}} = \overline{\bar{O}_4 \cdot \bar{O}_5 \cdot \bar{O}_6 \cdot \bar{O}_7}$$

$$B_1 = \overline{\overline{O_2 + O_3 + O_6 + O_7}}$$
$$= \overline{\bar{O}_2 \cdot \bar{O}_3 \cdot \bar{O}_6 \cdot \bar{O}_7}$$
$$B_0 = \overline{\overline{O_1 + O_3 + O_5 + O_7}}$$
$$= \overline{\bar{O}_1 \cdot \bar{O}_3 \cdot \bar{O}_5 \cdot \bar{O}_7}$$

Hence the logic diagram for Gray to Binary code converter using a 3 : 8 decoder and 4 input NAND gates is as shown in Fig. P. 6.20.2.



(C-1318) **Fig. P. 6.20.2 : Logic diagram for Gray to Binary code converter**

Ex. 6.20.3 : Implement a 3-bit binary to gray code converter using decoder IC 74138. **May 11, 8 Marks, Dec. 16, 10 Marks**

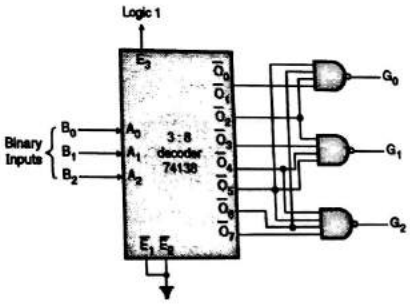
Soln. :

Step 1 : Write the truth table relating the binary and gray codes :

The truth table is as follows.

Table P. 6.20.3 : Truth table relating the binary and gray codes

Decimal	Binary inputs			Gray outputs		
	B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0



(C-492) **Fig. P. 6.20.3 : Binary to gray code converter using decoder 74138**

Step 2 : Obtain the expressions for G_2, G_1 and G_0 :

- Refer to the shaded portions of Table P. 6.20.3. Normally the expressions for G_2, G_1 and G_0 would have been written in the SOP form as,

$G_2 = O_4 + O_5 + O_6 + O_7$ $\quad$ $G_1 = O_2 + O_3 + O_4 + O_5$ $\quad$ $G_0 = O_1 + O_2 + O_5 + O_6$

- But the outputs of IC 74138 are active low. Hence we will have to convert these equations in terms of inverted "O" outputs as follows :

$$G_2 = O_4 + O_5 + O_6 + O_7 = \overline{\overline{O_4 + O_5 + O_6 + O_7}}$$

Scanned by CamScanner

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

$$\therefore \; G_2 = \overline{\bar{O}_4 \cdot \bar{O}_5 \cdot \bar{O}_6 \cdot \bar{O}_7} \quad \ldots(1)$$

$$\text{Similarly } G_1 = \overline{\bar{O}_2 \cdot \bar{O}_3 \cdot \bar{O}_4 \cdot \bar{O}_5} \quad \ldots(2)$$

$$\text{And } G_0 = \overline{\bar{O}_1 \cdot \bar{O}_2 \cdot \bar{O}_5 \cdot \bar{O}_6} \quad \ldots(3)$$

- Equations (1), (2) and (3) are implemented as shown in Fig. P. 6.20.3. Note that each equation represents a 4-input NAND gate.

Note : We can use IC 74138 as 3 : 8 decoder in order to implement the Boolean equations using 3 : 8 decoder.

Review Questions

Q. 1 Explain the working of a half adder. Draw its logic diagram.

Q. 2 Draw the logic diagram of full adder and its truth table.

Q. 3 Give the circuit diagram of a full adder using NAND gate and explain. Give its truth table.

Q. 4 What is similar between a half adder and a half subtractor ?

Q. 5 Build a full adder from half adder circuits.

Q. 6 With suitable block diagram explain the operation of n-bit serial adder.

Q. 7 Write a note on a 4 bit parallel binary adder.

Q. 8 Draw pin diagram of IC 7483 and explain its operation as 4-bit binary adder.

Q. 9 What is half subtractor ? Draw the logic diagram and truth table of half subtractor.

Q. 10 What is meant by a full subtractor ? Draw a full subtractor circuit.

Q. 11 Draw the circuit diagram of single digit BCD adder using IC 7483.

Q. 12 Implement a full subtractor circuit using only NAND gates.

Q. 13 In detail explain the working of 4-bit binary adder/subtractor using IC 7483.

Q. 14 For one digit BCD adder :

1. Draw circuit diagram. 2. Explain its working.
3. Can $(11)_{10}$ and $(9)_{10}$ be added by this circuit ? Justify.

Q. 15 Draw half subtractor circuit. Use NAND gates only. Explain its working.

Q. 16 Draw 4 bit adder/subtractor circuit using IC 7483 and IC 7486. Explain its working.

Q. 17 Draw the circuit of 4 bit binary parallel adder. Explain its working.

Q. 18 What is meant by a multiplexer ? Explain with block diagram the principle of multiplexing.

Q. 19 Explain with diagram and truth table the operation of 4 : 1 Mux.

Q. 20 Explain briefly with pin-diagram the following ICs IC-74157, IC-74151, IC-74150.

Q. 21 What is a 'Multiplexer tree' ?

Q. 22 Explain with diagram 64 to 1 multiplexer using four 16 to 1 multiplexer.

Q. 23 Explain with diagram 32 : 1 multiplexer using tree of 8 : 1 multiplexer.

Q. 24 Explain with diagram 32 : 1 multiplexer using two 16 : 1 multiplexer.

Q. 25 Give applications of multiplexers.

Q. 26 Explain with diagram the working of 1 to 8 demultiplexer.

Q. 27 Explain with diagram the working of 1 to 16 demultiplexer.

Q. 28 Explain with pin-diagram of ICs : IC74151, IC 74138.

Q. 29 Give the circuit diagram for 1 : 64 demux using tree of 1 : 16 demux.

Q. 30 Explain how demultiplexer can be used as a decoder.

Q. 31 What is the necessity of multiplexer ?

Q. 32 Draw the circuit diagram of 1 to 4 line demultiplexer and explain the working.

Q. 33 Draw circuit diagrams for the following :

1. 2 : 1 multiplexer. 2. 1 : 2 demultiplexer.

Q. 34 With a neat block diagram explain the function of an encoder.

Q. 35 What is meant by a priority encoder ? Give example.

Q. 36 What do you mean by a 'Decoder' ? Give its applications.

□□□

Scanned by CamScanner

CHAPTER 7

Module 4

Flip Flops

Syllabus :

Introduction, SR latch, concepts of Flip flops, SR, D, JK, T, Truth Tables and Excitation tables of all types, Race around condition, Master Slave JK Flip flops, Timing diagram, Flip flop conversion, State machines, State diagrams, State table, concept of Moore and Mealy machine.

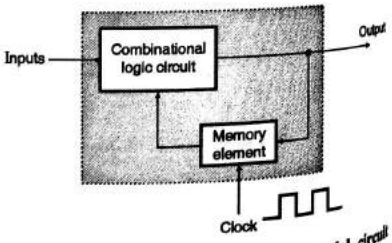
7.1 Introduction :

Combinational circuits :

- Till now we have discussed only the combinational circuits.
- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals. It does not depend on the past status of inputs.
- The combinational circuits do not use any memory. Therefore the previous states of input does not have any effect on the present state of the circuit.
- Also the sequence in which the inputs are being applied has no effect on the output of a combinational circuit. We do not have to use any timing and synchronization signal such as clock signal in a combinational circuit.

Sequential circuits :

- In the sequential circuit, the **timing** parameter also needs to be taken into consideration.
- The output of a sequential circuit depends on the present time inputs, the previous output (past) and the sequence in which the inputs are applied.
- In order to provide the previous input or output a memory element is required to be used. Thus a sequential circuit needs to use a memory element as shown in Fig. 7.1.1.



(C-560) **Fig. 7.1.1 : Block diagram of a sequential circuit**

- Fig. 7.1.1 shows the block diagram of a sequential circuit which includes the memory element in the feedback path.

Present state of sequential circuit :

The data stored by the memory element at any given instant of time is called as the present state of the sequential circuit.

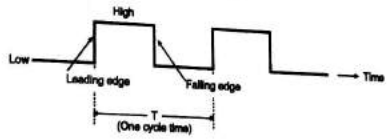
Next state :

The combinational circuit shown in Fig. 7.1.1 operates on the external inputs and the present state to produce new outputs. Some of these new outputs are stored in the memory element and called as the next state of the sequential circuit.

- The most important part of the sequential circuit seems to be the memory element. The memory element of Fig. 7.1.1 is known as Flip Flop (FF). It is the basic memory element.

7.1.1 Clock Signal :

- The clock signal shown in Fig. 7.1.2 is a timing signal. Every sequential signal will have this timing signal applied as an input signal as an input signal.
- Clock is a rectangular signal as shown in Fig. 7.1.2; with a duty cycle equal to 50%. That means its on time is equal to its off time.
- The clock signal repeats itself after every T seconds. Hence the clock frequency is f = 1/T.



(C-561) **Fig. 7.1.2 : Clock signal**

7.1.2 Clock Skew :

- Clock skew is defined as the time difference between the instants at which clock edges arrive at a pair of clock inputs.
- In a perfect system, the clock signals at various clock input pins of the system arrive at exactly the same instant of time and the skew is zero.
- But in real time systems, the edges do not arrive at exactly the same time and there is some skew.
- This clock skew occurs due to different delays the clock signal experiences on different paths from the clock generator to various circuits. The major reasons for this are :
  1. Different length of wires (wires introduce delay)
  2. Different number of gates (buffers) on the paths
  3. Use of flip-flops that clock on different edges (need for inverting clock for some flip-flops)
  4. The process of gating the clock to control loading of registers
- The **maximum allowable clock skew** for the system is equal to the difference between the shortest and longest path delays.
- Clock skew is an important design parameter in high-speed clock systems.

7.1.3 Comparison of Combinational and Sequential Circuits : MU : Dec. 15

University Questions

Q. 1 Differentiate in brief between combinational and sequential circuits. **(Dec. 15, 2 Marks)**

Sr. No.	Parameter	Combinational circuits	Sequential circuits
1.	Output depends on	Inputs present at that instant of time.	Present inputs and past inputs/outputs.
2.	Memory	Not necessary	Necessary
3.	Clock input	Not necessary	Necessary
4.	Examples	Adders, subtractors, code converters	Flip flops, shift registers, counters

Scanned by CamScanner

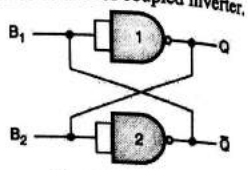
7.1.4 1-Bit Memory Cell (Basic Bistable Element) :

Flip-flop is also known as the basic digital memory circuit.

It has two stable states namely logic 1 state and logic 0 state. We can design it either using NOR gates or NAND gates.

A flip-flop can be designed by using the fundamental circuit shown in Fig. 7.1.3. NAND gates 1 and 2 are basically acting as inverters. Hence this circuit is called as a cross coupled inverter.

- Output of gate 1 is connected to the input of gate-2 and output of gate 2 is connected to input of gate-1 as shown in Fig. 7.1.3.



(C-562) **Fig. 7.1.3 : A cross coupled inverter as memory element**

Operation :

- Assume that output of gate-1 i.e. Q = 1. Hence $B_2 = 1$.
- As $B_2 = 1$, output of gate-2 i.e. $\bar{Q} = 0$. This makes $B_1 = 0$.
- Hence Q continues to be equal to 1.
- Similarly we can demonstrate that if we start with Q = 0, then we end up obtaining Q = 0 and $\bar{Q} = 1$.

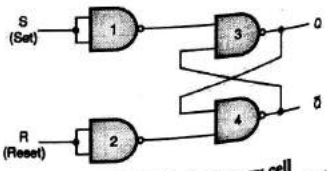
Conclusions :

From the above discussion we can draw the following conclusions :

- The outputs of the circuit (Q and $\bar{Q}$) will always be complementary. That means if Q = 0 then $\bar{Q} = 1$ and vice versa. They will never be equal $Q = \bar{Q} = 0$ or 1 is an invalid state.
- This circuit has two stable states. One of them corresponds to Q = 1, $\bar{Q} = 0$ and it is called as 1 state or **set state.** Whereas the other state corresponds to Q = 0, $\bar{Q} = 1$ and it is called as 0 state or **reset state.**
- If the circuit is in the reset state (Q = 0, $\bar{Q} = 1$), then it will continue to be in the reset state and if it is in the set state (Q = 1, $\bar{Q} = 0$) then it will continue to remain in the set state.
- This property of the circuit shows that it can store 1 bit of digital information. Therefore it is called as a **1-bit memory cell.**

7.1.5 Latch :

- The cross coupled inverter of Fig. 7.1.3 is capable of locking or latching the information. Hence this circuit is also called as a latch.
- The disadvantage of the cross coupled inverter circuit is that we cannot enter the desired digital data into it.



(C-563) **Fig. 7.1.4 : Modified memory cell**

- This disadvantage can be overcome by modifying the circuit as shown in Fig. 7.1.4. This modification will allow us to enter the desired digital data into the circuit.

Operation :

Case I : S = R = 0 (No change) :

- The outputs of gates 1 and 2 will become 1.

- Let Q = 0 and $\bar{Q} = 1$ initially. Hence both the inputs to gate 3 are 1 and the inputs to gate-4 are (01).
- So gate-3 output i.e. Q = 0 and gate-4 output i.e. $\bar{Q} = 1$.
- Thus with S = R = 0, there is no change in the state of outputs.

Case II : S = 1, R = 0 (Set) :

- Since S = 1 and R = 0, one of the inputs to gate-3 will be 0. This will force Q output to 1.
- Hence both the inputs to gate-4 will be 1. This forces $\bar{Q}$ to 0.
- Hence for S = 1, R = 0, the outputs are Q = 1 and $\bar{Q} = 0$. This is set state.

Case III : S = 0, R = 1 (Reset) :

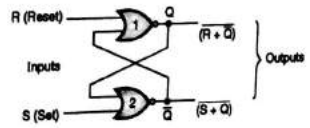
- If S = 0, R = 1 then one of the inputs to gate-4 will be 0. This will force the $\bar{Q}$ output to 1.
- Hence both the inputs to gate 3 will be 1. This forces Q to 0.
- Thus for S = 0, R = 1, the outputs are Q = 0, $\bar{Q} = 1$. This is the reset state or clear state.

Case IV : S = R = 1 (Race : Prohibited) :

- If S = R = 1 then outputs of gates 1 and 2 will be zero.
- Hence one of the inputs to gates 3 and 4 will be 0.
- So both the outputs Q and $\bar{Q}$ will try to become 1. It is not allowed as Q and $\bar{Q}$ should be complementary. Hence S = R = 1 condition is prohibited.

7.2 S-R Latch using NOR Gates :

- Latch is a bistable circuit which has two stable states.
- It has two outputs Q and $\bar{Q}$ which are complements of each other. The two input terminals to this latch are set (S) and reset (R).
- Latch is a sequential logic circuit which monitors all its inputs continuously and will change its output as soon as the input changes. It does not wait for the clock signal.
- Generally an enable input signal is given for a latch. When the enable signal is active, the output will change in response to a change in input.
- S-R (set – reset) latch is the simplest type of latch and Fig. 7.2.1 shows S-R latch constructed using the NOR gates.
- Two NOR gates are cross coupled. That means the output of NOR gate 1 is connected to one of the inputs of the NOR gate 2. And output of NOR gate 2 is connected to one of the inputs of NOR gate 1.



(C-564) **Fig. 7.2.1 : S-R latch using NOR gates**

- The outputs Q and $\bar{Q}$ can be expressed using Boolean equations as follows :

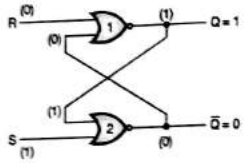
$$Q = \overline{R+\bar{Q}} \quad \text{and} \quad \bar{Q} = \overline{S+Q}$$

Scanned by CamScanner

7.2.1 Operation of S-R Latch :

- Remember that the NOR gate output goes to "0" when anyone its inputs becomes HIGH (1).
- If both the inputs are high (1), then the output is "0".
- Let us understand the operation under four different conditions :
  1. Case I : S = 1, R = 0
  2. Case II : S = 0, R = 1
  3. Case III : S = 0, R = 0
  4. Case IV : S = 1, R = 1
- For the operation refer Figs. 7.2.2(a), (b), (c), (d).

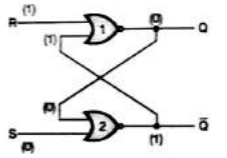
Case I : S = 1, R = 0 (Set)



(C-565) **Fig. 7.2.2(a) : Operation with S = 1, R = 0**

- As S = 1, output of NOR gate 2 i.e. $\bar{Q}$ becomes 0. R is already 0. Hence both inputs of NOR – 1 are 0.
- Therefore Q = 1.
- Hence both inputs to NOR – 2 are 1.
- Hence $\bar{Q}$ = 0. Thus a stable output state of Q = 1 and $\bar{Q}$ = 0 is reached.
- ∴ For S = 1, R = 0, Q = 1 and $\bar{Q}$ = 0. This is called as the "Set" mode.

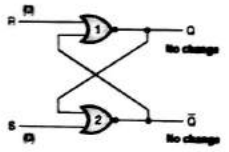
Case II : S = 0, R = 1 (Reset)



(C-566) **Fig. 7.2.2(b) : Operation with S = 0 and R = 1**

- As R = 1 output of NOR-1 i.e. Q becomes 0. S is already 0. Hence both inputs to NOR-2 are 0.
- Hence output of NOR-2 i.e. $\bar{Q}$ = 1. Thus the circuit reaches a stable state where Q = 0 and $\bar{Q}$ = 1.
- ∴ For S = 0 and R = 1, Q = 0 and $\bar{Q}$ = 1. This is known as the reset mode of operation.

Case III : S = 0, R = 0



(C-567) **Fig. 7.2.2(c) : Operation with S = R = 0**

- We know that $\bar{Q} = \overline{S+Q}$ and $Q = \overline{R+\bar{Q}}$
- Substitute S = R = 0

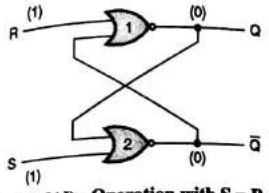
  $\therefore \bar{Q} = \overline{0+Q}$ and $Q = \overline{0+\bar{Q}}$

  $\therefore \bar{Q} = \bar{0}\cdot\bar{Q}$ and $Q = \bar{0}\cdot Q$

  according to De Morgan's theorems.
- $\therefore \bar{Q} = 1\cdot\bar{Q} = \bar{Q}$ and $Q = 1\cdot Q = Q$
- ∴ If S = R = 0, then Q and $\bar{Q}$ do not change their states.

Case IV : S = 1, R = 1

- If S = R = 1, then one input each of NOR-1 and NOR-2 is at logic 1. So outputs of both these gates will try to become 0.



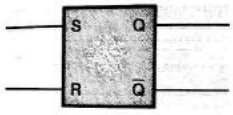
(C-568) **Fig. 7.2.2(d) : Operation with S = R = 1**

- Thus both the outputs Q and $\bar{Q}$ try to become 0 at the same time. This is not acceptable because Q and $\bar{Q}$ should be complements of each other.
- Therefore this state is an indeterminate state and therefore should be avoided.
- This state should not be used because it violates the basic requirement of a latch i.e. Q and $\bar{Q}$ should be complements of each other.

7.2.2 Symbol and Truth Table of S-R Latch :

- The symbol and truth table of S-R latch using NOR gates are as shown in Figs. 7.2.3(a) and (b) respectively.
- In the truth table Q_n and $\bar{Q}_n$ represent the present states of outputs i.e. these are the outputs before applying a new set of inputs.
- Q_{n+1} and $\bar{Q}_{n+1}$ represent the next states of outputs i.e. the outputs after apply a new set of inputs.

(C-569)



Q_n and $\bar{Q}_n$: Present states

Q_{n+1} and $\bar{Q}_{n+1}$: Next states

(a) Symbol

Inputs		Outputs				Comment
S	R	Q_n	$\bar{Q}_n$	Q_{n+1}	$\bar{Q}_{n+1}$	
0	0	0	1	0	1	No change (NC)
0	0	1	0	1	0	
0	1	0	1	0	1	Reset
0	1	1	0	0	1	
1	0	0	1	1	0	Set
1	0	1	0	1	0	
1	1	0	1	×	×	Prohibited state
1	1	1	0	×	×	

(b) Truth table of S-R latch

Fig. 7.2.3

Summary of operation of S-R latch :

The summary of operation of an S-R latch is as follows :

- For S = R = 0 the latch output does not change.
- S = 0, R = 1 is called as the "Reset" condition as Q = 0 and $\bar{Q}$ = 1.
- S = 1, R = 0 is called as the "Set" condition as Q = 1 and $\bar{Q}$ = 0.
- S = R = 1 is the prohibited state. The output is unpredictable. This condition should therefore be avoided.

Race condition :

- The condition S = R = 1 is called as "Race" condition.
- When any one input to a NOR gate is 1, its output becomes 0. Thus both the outputs will try to become 0. This is called as the RACE condition.

7.2.3 Characteristic Equation :

MU : Dec. 06

University Questions

Q.1 Obtain characteristics equation of S- R flip/flop and excitation table. **(Dec. 06, 6 Marks)**

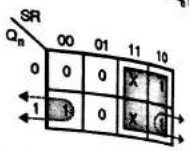
Scanned by CamScanner

- One way of explaining the behaviour of a latch or flip-flop is to use its truth table. The truth table is also called as **excitation table.** Another way of doing it is to use special type of equations called **characteristic equations.**
- The characteristic equation of a flip-flop is the equation which relates the next state of the flip flop or latch Q_{n+1} or $\bar{Q}_{n+1}$ to the current state and inputs (Q_n, S and R).
- Characteristic equation is actually obtained from the truth table of the flip flop or latch, using the K-map.

Characteristic equation of SR latch :

- Refer to the truth table of SR latch and write the K-map for the next state of output i.e. Q_{n+1} as shown in Fig. 7.2.4.
- After simplification, the characteristic equation of SR latch is given by,

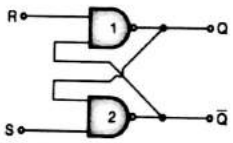
$$Q_{n+1} = S + \bar{R}\,Q_n \qquad ...(7.2.1)$$



(C-570) **Fig. 7.2.4 : K-map for next state Q_{n+1} of SR latch**

7.2.4 NAND Latch [S-R Latch using NAND Gates] :

- We can construct a S-R latch with NAND gates as shown in Fig. 7.2.5(a).
- Note that the outputs of two NAND gates are cross connected, in an identical manner as that in the NOR latch.
- Fig. 7.2.5(a) shows the NAND latch and Fig. 7.2.5(b) shows its truth table.
- The operation of S-R NAND latch is summarised in Fig. 7.2.6.



(a) NAND latch

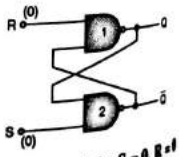
S	R	Q_{n+1}	$\bar{Q}_{n+1}$	
0	0	RACE	RACE	
0	1	0	1	→ Reset
1	0	1	0	→ Set
1	1	(NC) Q_n	(NC) $\bar{Q}_n$	→ Inactive

(b) Truth table

(C-571) **Fig. 7.2.5**

Case I : S = 0, R = 0 : Race

- When any one input of a NAND gate becomes 0, its output is forced to 1.
- Here S = R = 0. ∴ Q and $\bar{Q}$ both will be forced to be equal to 1.
- This is an undeterminate state and hence should be avoided.
- This is also called as Race condition.



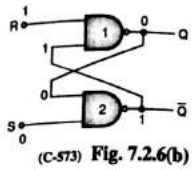
(C-572) **Fig. 7.2.6(a) : S = 0, R = 0**

Case II : S = 0, R = 1 : Reset

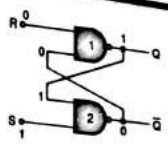
- Since S = 0, it forces $\bar{Q}$ to be 1.
- Hence both inputs to NAND-1 are 1.
- Hence Q = 0.
- Thus with S = 0 and R = 1 the outputs are Q = 0 and $\bar{Q}$ = 1.
- This is the reset condition.

Case III : S = 1, R = 0 : Set

- Since R = 0, Q is forced to 1.
- Hence both inputs to NAND-2 are 1.
- Hence $\bar{Q}$ = 0.
- Thus with S = 1 and R = 0 the outputs are Q = 1 and $\bar{Q}$ = 0.
- This is the set condition.



(C-573) **Fig. 7.2.6(b)**



(C-573) **Fig. 7.2.6(c)**

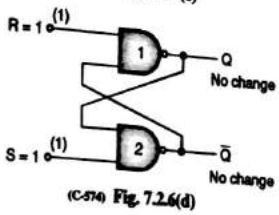
Case IV : S = 1, R = 1 : No change

- $Q_{n+1} = \overline{R \cdot \bar{Q}_n}$ and $\bar{Q}_{n+1} = \overline{S \cdot Q_n}$
- Using De-Morgan's theorem,

  $Q_{n+1} = \bar{R} + Q_n$ and $\bar{Q}_{n+1} = \bar{S} + \bar{Q}_n$.
- Substitute $\bar{R} = 0$ and $\bar{S} = 0$ to get,

  $Q_{n+1} = 0 + Q_n = Q_n$ and $\bar{Q}_{n+1} = 0 + \bar{Q}_n = \bar{Q}_n$.

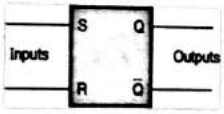
  Thus there is no change in the outputs if S = R = 1.
- The symbol for S-R latch is shown in Fig. 7.2.7 alongwith the summary of operation.



(C-574) **Fig. 7.2.6(d)**

Circuit symbol and summary of operation of S-R latch.

- For S = 0, R = 0, both the outputs will be forced to become 1. This is RACE condition and should be available.
- For S = 0, R = 1, the outputs are Q = 0, $\bar{Q}$ = 1 and it is called as the Reset condition.
- For S = 1, R = 0, the outputs are Q = 1, $\bar{Q}$ = 0 and it is called as the set condition.
- For S = R = 1, there is no change in the output state.



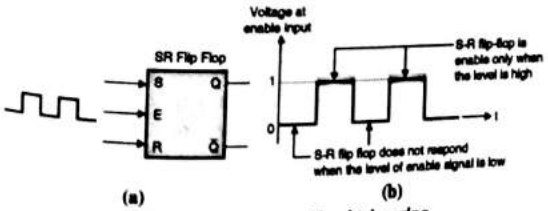
(C-575) **Fig. 7.2.7 : Symbol of S-R latch**

7.3 Triggering Methods :

- In the latches and flip-flops, we use the additional signal called clock signal.
- Depending on which portion of the clock signal the latch or flip-flop responds to, we can classify them into two types :

  1. Level triggered circuits 2. Edge triggered circuits

7.3.1 Concept of Level Triggering :



(C-576) **Fig. 7.3.1 : Concept of level triggering**

Scanned by CamScanner

- The latch or flip-flop circuits which respond to change in their inputs, only if their enable input (E) held at an active level which may be either HIGH or LOW level are called as **level triggered** latches or flip-flops. Thus these circuits do not respond at the rising or falling edges of clock. They only respond to the steady HIGH or LOW levels of the clock signal.
- Fig. 7.3.1(a) shows the symbol of a level triggered SR flip flop and Fig. 7.3.1(b) shows the clock signal applied at its input.

7.3.2 Types of Level Triggered Flip-flops :

There are two types of level triggered flip-flops :

1. Positive level triggered. 2. Negative level triggered.

Positive level triggered :

- If the outputs of a flip-flop respond to the input changes, only when its clock inputs at HIGH (1) level, then it is called as the positive level triggered flip-flop.
- The block diagram shown in Fig. 7.3.1(a) is a positive level triggered S-R FF.

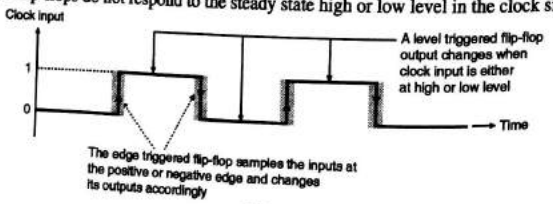
Negative level triggered FF :

If the outputs of a flip-flop respond to the input changes, only when its clock input is at LOW (0) level, then it is called as the negative level triggered flip-flop.

Note : The level triggering is not used practically, due to some of its disadvantages.

7.3.3 Concept of Edge Triggering :

- The flipflops which change their outputs only corresponding to the positive (rising) or negative (falling) edge of the clock input are called as **edge triggered flipflops.**
- These flip-flops are therefore said to be edge sensitive or edge triggered rather and not level triggered.
- The rectangular signal applied to the clock input of a flip-flop is shown in Fig. 7.3.2. If the same signal is applied as the clock signal to an edge triggered flip flop, then its outputs will change only at either rising (positive) edge or at the falling (negative) edge of the clock. The edge triggered flip-flops do not respond to the steady state high or low level in the clock signal at all.



(C-577) **Fig. 7.3.2**

7.3.4 Types of Edge Triggered Flip Flops :

- There are two types of edge triggered flip flops :
  1. Positive edge triggered flip flops 2. Negative edge triggered flip flops.

- Positive edge triggered flip flops will allow its outputs to change in response to its inputs only at the instants corresponding to the rising edges of clock (or positive spikes). Its outputs will not respond to change in inputs at any other instant of time.
- Negative edge triggered flip flops will respond only to the negative going edges (or spikes) of the clock.

7.4 Gated Latches (Level Triggered SR Flip Flop) :

- We have discussed the RS latches using the NAND and NOR gates.
- Now 2 more NAND gates are added to the basic SR latch and one more input called enable (E) is added, in order to obtain a gated SR latch or a level triggered SR flip-flop.
- A level voltage (0 or 1) or a clock signal can be applied to the enable (E) input.
- These flipflops will respond to the inputs if and only if we apply an **active level** at the enable input. This active level can be either 0 or 1 depending on the type of flip-flop.
- Such flipflops are called as level triggered flipflops or gated latches or clocked flipflops.

7.4.1 Types of Level Triggered (Clocked) Flip Flops :

There are two types of level triggered latches :

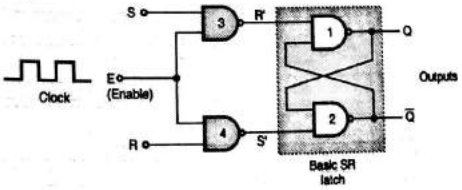
1. Positive level triggered. 2. Negative level triggered.

7.5 The Gated S-R Latch (Level Triggered S-R Flip Flop) : MU : Dec. 04

University Questions

Q.1 Design a clocked sequential R.S. flip-flop. **(Dec. 04, 5 Marks)**

- The gated S-R latch is shown in Fig. 7.5.1. It is also called as clocked SR flip flop. It is basically the S-R latch using NAND gates with an additional "enable" (E) input. It is also called as **level triggered** S-R FF.
- The outputs of basic S-R latch used to change instantly in response to any change made at the input. But this does not happen with the gated S-R latch.
- For this circuit, the change in output will take place if and only if the enable input (E) is made active. This circuit being positive level triggered, will respond to changes in input only if the enable input is held at logic 1 level.
- In short this circuit will operate as an S-R latch if E = 1 (Enable input is active) but there is no change in the outputs if E = 0 (Enable input is inactive).



(C-578) **Fig. 7.5.1 : Gated S - R latch**

Operation :

Case I : S = X, R = X, E = 0 (No change)

- Since enable E = 0, the outputs of NAND gates 3 and 4 will be forced to be 1 irrespective of the values of S and R.
- That means R′ = S′ = 1. These are the inputs of the basic S-R latch enclosed in the dotted box in Fig. 7.5.1.

Scanned by CamScanner

- Hence the outputs of NAND latch i.e. Q and $\bar{Q}$ will not change. Thus if E = 0, then there is no change in the output of the gated S-R latch.

Case II : S = R = 0 E = 1 : No change

- If S = R = 0 then outputs of NAND gates 3 and 4 are forced to become 1.
- Hence R′ and S′ both will be equal to 1. Since S′ and R′ are the inputs of the basic S-R latch using NAND gates, there will be no change in the state of outputs. Thus for S = R = 0 the output state of this flip-flop remains unchanged.

Case III : S = 0, R = 1, E = 1 (Reset)

- Since S = 0, output of NAND-3 i.e. R′ = 1. And as R = 1 and E = 1 the output of NAND-4 i.e. S′ = 0.
- Hence $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$. This is the reset condition.

Case IV : S = 1, R = 0, E = 1 (Set)

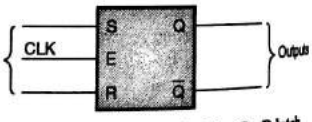
- Output of NAND 3 i.e. R′ = 0 and output of NAND 4 i.e. S′ = 1.
- Hence output of S-R NAND latch is $Q_{n+1} = 1$ and $\bar{Q}_{n+1} = 0$.
- This is the reset condition.

Case V : S = 1, R = 1, E = 1 (RACE)

- As S = 1, R = 1 and E = 1, the outputs of NAND gates 3 and 4 both are 0. i.e. S′ = R′ = 0.
- Hence the "Race" condition will occur in the basic NAND latch. This operation should be avoided as both Q and $\bar{Q}$ will try to become 1 at the same time as discussed earlier.

Symbol and truth table :

The symbol and truth table of the gates S-R latch are as shown in Figs. 7.5.2(a) and (b) respectively.



(C-579) **Fig. 7.5.2(a) : Symbol for S - R latch**

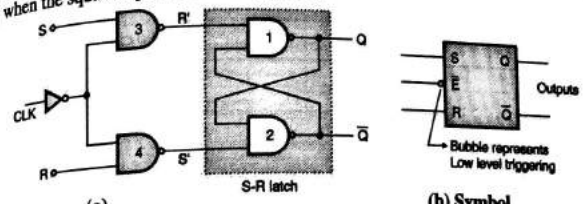
Inputs				Outputs		Comments
Case	Enable E	S	R	Q_{n+1}	$\bar{Q}_{n+1}$	
I	0	×	×	Q_n	$\bar{Q}_n$	No change as E = 0
II	1	0	0	Q_n	$\bar{Q}_n$	No change (NC)
III	1	0	1	0	1	Reset condition
IV	1	1	0	1	0	Set condition
V	1	1	1	RACE (Indeterminate)		Avoid this condition

Fig. 7.5.2(b) : Truth table of gated S - R latch

7.5.1 Negative Level Triggered SR Flip Flop :

- Fig. 7.5.3 shows the circuit diagram of a negative level triggered SR flip-flop. It is the same circuit that we discussed in the previous section with only one additional inverter.

- Due to the additional inverter connected to the enable terminal, this circuit becomes sensitive to the low level (0) applied to the enable input. Hence it will enable the outputs if E = 0.
- If a square wave is applied to the enable input then the latch output will respond to input changes only when the square input is at its low (0) level.



(a) (b) Symbol

(C-580) **Fig. 7.5.3 : Negative level triggered S - R latch**

- Fig. 7.5.3(b) shows the symbol of negative level triggered S-R latch. Note that there is a bubble added to the enable input which is active low input.

7.5.2 Disadvantage of S-R latch :

From the truth tables of the S-R latch using NOR and NAND gates we can draw the following important conclusion :

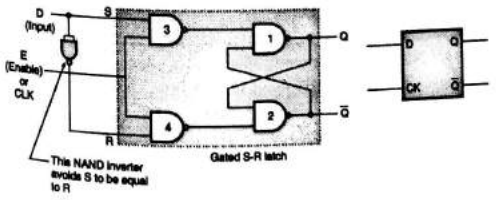
1. When S = R = 0 or S = R = 1, the outputs Q and $\bar{Q}$ either don't change (NC) or they are indeterminate (invalid) due to race condition.
2. This disadvantage of S-R latch can be overcome by using the gated D latch discussed in the next section.

7.5.3 Application of S-R latch :

It is used as electronic timer.

7.6 The Gated D Latch (Clocked D Flip Flop) :

- The circuit diagram of the gated D latch is shown in Fig. 7.6.1(a) and its logic symbol is shown in Fig. 7.6.1(b). This is also called as level triggered D flip-flop or clocked D flip-flop.
- Note that D latch is the simple gated S-R latch with a small modification. A NAND inverter is connected between its S and R inputs as shown in Fig. 7.6.1(a).
- This latch has only one input denoted by D.
- Due to the NAND inverter, S and R inputs will always be the complements of each other. Hence the input conditions such as S = R = 0 or S = R = 1, will never appear. This will avoid the problems associated with SR = 00 and SR = 11 conditions of SR flip-flop.



(a) Gated D latch (b) Logic symbol of gated D latch

(C-3419) **Fig. 7.6.1**

Scanned by CamScanner

Truth table : Truth table for the gated D latch is given in Table 7.6.1.

Table 7.6.1 : Truth table for the gated D latch

Inputs		Outputs		Comment
E	D	Q_{n+1}	$\bar{Q}_{n+1}$	
0	×	Q_n	$\bar{Q}_n$	No change (NC)
1	0	0	1	Reset condition
1	1	1	0	Set condition

Operation :

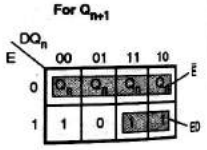
- If E = 0 then the latch is disabled. Hence there is no change in output.
- If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state the, next state is $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$. This is the reset condition.
- On the other hand if E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and $Q_{n+1} = 1$, $\bar{Q}_{n+1} = 0$ irrespective of the present state.

From the truth table it is evident that Q output is same as D input, in otherwords Q output follows the D input. If D = 0 then Q = 0 and if D = 1 then Q = 1. However output follows input after some propagation delay hence the other name of the D flip-flop is delay flip-flop.

Characteristic equation for D latch :

- Refer to the truth table of D latch and write the K-map for Q_{n+1} as shown in Fig. 7.6.2.
- After simplification we get the characteristic equation of D latch as,

$$Q_{n+1} = E \cdot D + \bar{E}$$



(C-584) **Fig. 7.6.2 : K-map for D latch**

7.7 Gated JK Latch (Level Triggered JK Flip Flop) :

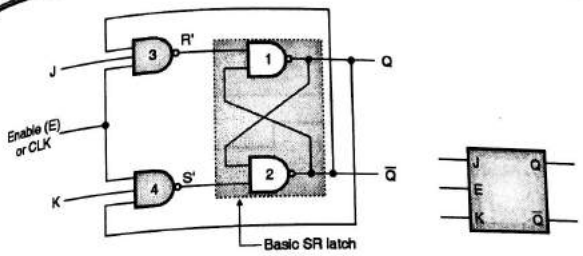
MU : May 05. Dec. 06. Dec. 16

University Questions

Q. 1 Design a J-K flip flop using basic NAND cell. **(May 05, 5 Marks, Dec. 06, 3 Marks)**

Q. 2 Draw JK flip-flop using SR flip-flop and additional gates. Explain briefly the race around condition in JK flip-flop. **(Dec. 16, 4 Marks)**

- The JK latch using NAND gates is shown in Fig. 7.7.1(a). It consists of the basic SR latch and an enable input. It is also called as level triggered JK flip-flop.
- Note the outputs Q and $\bar{Q}$ have been fed back and connected to the inputs of NAND gates 4 and 3 respectively.
- The JK latch of Fig. 7.7.1(a) responds, to the input changes if a positive level is applied at the enable (E) input. Hence it is a positive level triggered latch.



(a) A level triggered JK flip-flop **(b) Symbol of the gated JK latch**

(C-3420) **Fig. 7.7.1**

Operation :

- The operation of NAND JK latch is exactly identical to that of the positive edge triggered JK flip flop discussed in section 7.10.1.
- The only difference between them is that, this circuit is level triggered.
- The operation of NAND JK latch has been summarized in Table 7.7.1.

Table 7.7.1 : Truth table of positive level triggered JK latch

Case No.	Inputs			Outputs		State
	E	J	K	Q_{n+1}	$\bar{Q}_{n+1}$	
Case I	0	×	×	Q_n	$\bar{Q}_n$	No change
Case II	1	0	0	Q_n	$\bar{Q}_n$	No change
Case III	1	0	1	0	1	Reset
Case IV	1	1	0	1	0	Set
Case V	1	1	1	$\bar{Q}_n$	Q_n	Toggle

7.7.1 Race Around Condition in JK Latch :

MU : Dec. 02. May 06. Dec. 06. Dec. 08. Dec. 09. Dec. 10. May 11. Dec. 11. Dec. 12. Dec. 15. Dec. 16

University Questions

Q. 1 What is roll condition ? Why does it occur ? **(Dec. 02, 2 Marks)**

Q. 2 Explain in detail the "Race Around Condition". What are the conditions to occur it ? **(May 06, 4 Marks)**

Q. 3 Explain race around condition. **(Dec. 06, 3 Marks)**

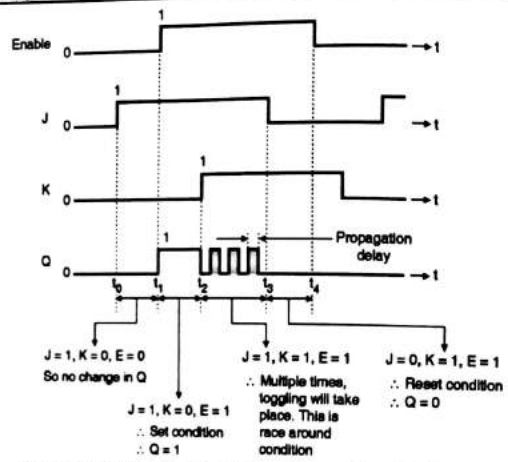
Q. 4 What is race condition ? **(Dec. 08, 5 Marks)**

Q. 5 Write short note on : Race around condition in J-K flip-flop. **(Dec. 09, Dec. 10, May 11, 5 Marks, Dec. 11, 7 Marks, Dec. 12, 20 Marks)**

Scanned by CamScanner

Q. 6 Explain the race around condition in JK flip-flop. State various methods to overcome it. **(Dec. 15, 2 Marks)**

Q. 7 Draw JK flip-flop using SR flip-flop and additional gates. Explain briefly the race around condition in JK flip-flop. **(Dec. 16, 4 Marks)**



(C-586) **Fig. 7.7.2 : Waveforms for various modes of a JK latch**

- The "Race Around Condition" that we are going to explain occurs when J = K = 1 i.e. when the latch is in the toggle mode.
- Refer Fig. 7.7.2 which shows the waveforms for the various modes, when a rectangular waveform is applied to the "Enable" input.

Interval $t_0 - t_1$

- During this interval J = 1, K = 0 and E = 0.
- Hence the latch is disabled and there is no change in Q.

Interval $t_1 - t_2$

- During this interval J = 1, K = 0 and E = 1.
- Hence this is a set condition and Q becomes 1.

Interval $t_2 - t_3$: Race around

- At instant t_2, J = K = 1 and E = 1 Hence the JK latch is in the toggle mode and Q becomes low (0) and $\bar{Q} = 1$.
- These changed outputs get applied at the inputs of NAND gates 3 and 4 of the JK latch. Thus the new inputs to Gates 3 and 4 are :

  NAND - 3 : J = 1, E = 1, $\bar{Q} = 1$.

  NAND - 4 : K = 1, E = 1, Q = 0.
- Hence R′ will become 0 and S′ will become 1.

- Therefore after a time period corresponding to the propagation delay, the Q and $\bar{Q}$ outputs will change to, Q = 1 and $\bar{Q} = 0$.
- These changed output again get applied to the inputs of NAND-3 and 4 and the outputs will toggle again.
- Thus as long as J = K = 1 and E = 1, the outputs will keep toggling indefinitely as shown in Fig. 7.7.2. This multiple toggling in the J-K latch is called as Race Around condition. It must be avoided.

Interval $t_3 - t_4$

- During this interval J = 0, K = 1 and E = 1. Hence it is the reset condition.
- So Q becomes zero.

How to avoid race around condition ?

The race around condition in JK latch can be avoided by :

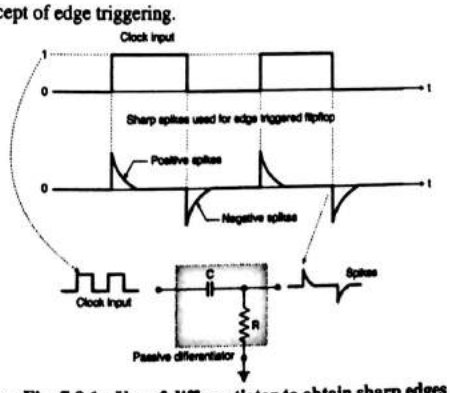
- Using the edge triggered JK flip flop.
- Using the master slave JK flip flop.

7.7.2 Difference between Latch and Flip-flop :

- Latches and flip flops both are basically the bistable elements.
- As discussed earlier a latch has got an enable input. As long as it is active, the latch output will keep changing according to the changes in its input. In other words latch is a level triggered flip flop.
- But flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously.
- The flip-flops are therefore said to be edge sensitive or edge triggered rather than being level triggered like latches.

7.8 Edge Triggered Flip Flops :

- We have already discussed the concept of edge triggering.
- For the edge triggered flip flops, it is necessary to apply the clock signal (timing signal) in the form of sharp positive and negative spikes instead of in the form of a rectangular pulse train.
- Such sharp spikes are shown in Fig. 7.8.1. These spikes can be derived from the rectangular clock pulses with the help of a passive differentiator circuit shown in Fig. 7.8.1.
- Thus the passive differentiator acts as a pulse shaping circuit.



(C-587) **Fig. 7.8.1 : Use of differentiator to obtain sharp edges**

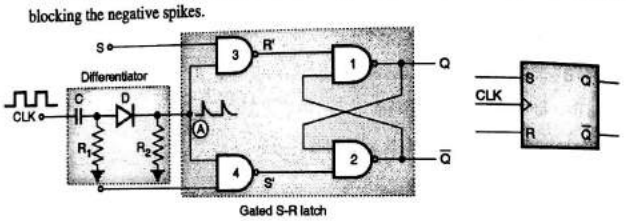
Scanned by CamScanner

7.8.1 Positive Edge Triggered S-R Flip Flop :

Fig. 7.8.2(a) shows the circuit diagram of a positive edge triggered S-R flip flop and Fig. 7.8.2(b) shows the logic symbol for it.

Operation :

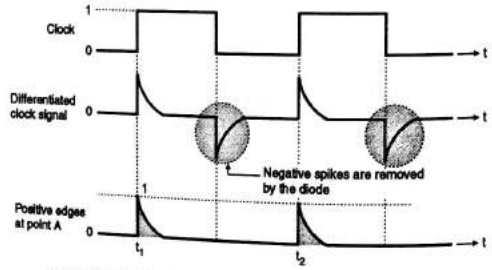
- Note that the SR flip flop of Fig. 7.8.2(a) consists of a differentiator circuit and a gated S-R latch (level triggered SR flip-flop) which we have already discussed.
- C - R_1 acts as a differentiator and converts the rectangular clock pulses into positive and negative spikes.
- The diode acts as rectifying diode and allows only the positive spikes to pass through it while, blocking the negative spikes.



(a) Positive edge triggered S-R flip flop **(b) Logic symbol**

(C-588) **Fig. 7.8.2**

- Thus we get only the positive spikes, corresponding to the leading edges of the clock signal, as shown in Fig. 7.8.3 at point A which is enable input of the gated S-R latch.



(C-589) **Fig. 7.8.3 : Generation of positive triggering pulses**

- Due to these sharp positive spikes applied at point "A", the gated S-R latch in the S-R flip flop will be enabled only for a short duration of time when the positive spike is present at A. (at instants t_1, t_2 ... in Fig. 7.8.3)

- At these instants, the flip-flop behaviour is exactly identical to that of the enabled gated S-R latch (enabled level triggered SR flip-flop).
- The truth table of positive edge triggered S-R flip flop will also be identical to that of the gated S-R latch with only one change. The "enable" input will now be replaced by clock input. And the outputs will change only if a positive edge is applied to the clock input. The positive clock edge is denoted by an arrow (↑) in Table 7.8.1.
- The truth table of a positive edge triggered SR flip flop is shown in Table 7.8.1.

Table 7.8.1 : Truth table of a positive edge triggered SR flip flop

	Inputs			Outputs		Remark
	CLK	S	R	Q_{n+1}	$\bar{Q}_{n+1}$	
FF is disabled	0	×	×	Q_n	$\bar{Q}_n$	No change (NC)
	1	×	×	Q_n	$\bar{Q}_n$	No change (NC)
	↓	×	×	Q_n	$\bar{Q}_n$	No change (NC)
FF responds only to the positive edges of clock	↑	0	0	Q_n	$\bar{Q}_n$	No change (NC)
	↑	0	1	0	1	Reset
	↑	1	0	1	0	Set
	↑	1	1	Race	Race	Avoid

↓ = Negative edge of clock, ↑ = Positive edge of clock

- Note that this flip-flop does not respond to the positive or negative levels in the clock signal. Similarly it does not respond to the negative edges of the clock.
- This flip flop will respond only to the positive edges of clock signal.
- With positive edge of the clock, the SR flip flop behaves in the following way.

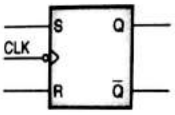
S = R = 0	→	No change in output
S = 0, R = 1	→	$Q_{n+1} = 0, \bar{Q}_{n+1} = 1$ Reset condition
S = 1, R = 0	→	$Q_{n+1} = 1, \bar{Q}_{n+1} = 0$ Set condition
S = R = 1	→	Race condition.

7.8.2 Negative Edge Triggered S - R Flip Flop :

- The internal circuit (with NAND gates) of the negative edge triggered S-R flip flop is exactly same as that for the positive edge triggered SR flip-flop discussed earlier.

Scanned by CamScanner

- The differentiator circuit is slightly modified in order to make this flip flop respond to the negative (falling) edges of the clock input.
- Fig. 7.8.4 shows the circuit symbol of the negative edge triggered S-R flip flop and Table 7.8.2 shows its truth table.



(C-590(a)) **Fig. 7.8.4 : Circuit symbol of negative edge triggered SR FF**

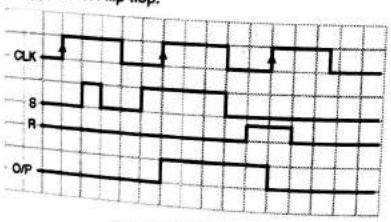
Table 7.8.2 : Truth table of negative edge triggered S-R FF

Inputs			Outputs		State	
CLK	S	R	Q_{n+1}	$\bar{Q}_{n+1}$		
0	×	×	Q_n	$\bar{Q}_n$	No change (NC)	FF is disabled
1	×	×	Q_n	$\bar{Q}_n$	No change (NC)	
↑	×	×	Q_n	$\bar{Q}_n$	No change (NC)	
↓	0	0	Q_n	$\bar{Q}_n$	No change (NC)	FF responds only to negative edges
↓	0	1	0	1	Reset	
↓	1	0	1	0	Set	
↓	1	1	Race	Race	Avoid	

↑ = Positive edge of clock
↓ = Negative edge of clock

Ex. 7.8.1 : Draw the waveform for the Q output for the input waveforms shown in Fig. P. 7.8.1 for the positive edge triggered SR flip-flop.

Soln. :



(C-591) **Fig. P. 7.8.1**

7.9 Edge Triggered D Flip Flop :

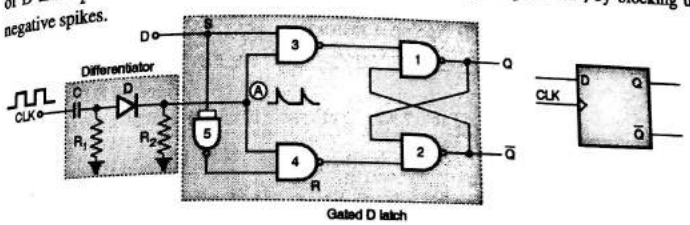
The edge triggered D flip flops can be of two types :

1. Positive edge triggered D flip flop
2. Negative edge triggered D flip flop.

These flip flops can be derived from the level triggered D latch which we have discussed in section 7.6.

7.9.1 Positive Edge Triggered D Flip Flop :

- Fig. 7.9.1(a) shows the positive edge triggered D flip flop. It consists of a gated D latch and a differentiator circuit. The symbol is as shown in Fig. 7.9.1(b).
- The clock pulses are applied to the circuit through a differentiator formed by R_1C and a rectifier circuit consisting of diode D and R_2.
- The NAND gates 1 through 5 form a D latch.
- The differentiator converts the clock pulses into positive and negative spikes and the combination of D and R_2 will allow only the positive spikes to pass through to point "A", by blocking the negative spikes.



(a) Positive edge triggered D-flip flop **(b) Symbol**

(C-592) **Fig. 7.9.1**

- The operation of edge triggered D FF is exactly same as that of the gated D latch discussed earlier with only one difference. The D latch had an enable terminal whereas the D flip flop has a clock input.
- The gated D latch of Fig. 7.9.1 is enabled by the positive spikes obtained at point A. Hence the outputs will change based on the inputs at these particular instants only. Thus the edge triggered D flip-flop responds only to the positive (leading) edges of the clock pulses.
- At any other instants of time, the D flip flop will not respond to the changes in input.

Truth table : Table 7.9.1 shows the truth table of a positive edge triggered D flip flop.

Table 7.9.1 : Truth table of a positive edge triggered D flip flop

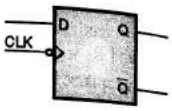
	Inputs		Outputs		State
	CLK	D	Q_{n+1}	$\bar{Q}_{n+1}$	
FF disabled	0	×	Q_n	$\bar{Q}_n$	No change
	1	×	Q_n	$\bar{Q}_n$	No change
	↓	×	Q_n	$\bar{Q}_n$	No change
FF responds only to the positive edges	↑	0	0	1	Q follows D input
	↑	1	1	0	

From the truth table it is clear that the Q output of the flip flop follows the D input.

Scanned by CamScanner

7.9.2 Negative Edge Triggered D Flip Flop :

- The symbol for negative edge triggered D flip flop is shown in Fig. 7.9.2.
- This F/F responds only to the negative edges of the clock pulses. This is how it is different from the positive edge triggered flip flop.
- Otherwise, the operation of the negative edge triggered D flip flop is exactly same as that of the positive edge triggered D flip flop. The truth table of negative edge triggered D flip flop is shown in Table 7.9.2.



(C-593) **Fig. 7.9.2 : Symbol of negative edge triggered D flip flop**

Table 7.9.2 : Truth table of negative edge triggered D flipflop

	Inputs		Outputs		Comment
	CLK	D	Q_{n+1}	$\bar{Q}_{n+1}$	
FF disabled	0	×	Q_n	$\bar{Q}_n$	No change
	1	×	Q_n	$\bar{Q}_n$	No change
	↑	×	Q_n	$\bar{Q}_n$	No change
FF responds only to the negative edges	↓	0	0	1	Q output follows D input
	↓	1	1	0	

7.9.3 Applications of D Flipflop :

1. As a delay element.
2. In the digital latches.

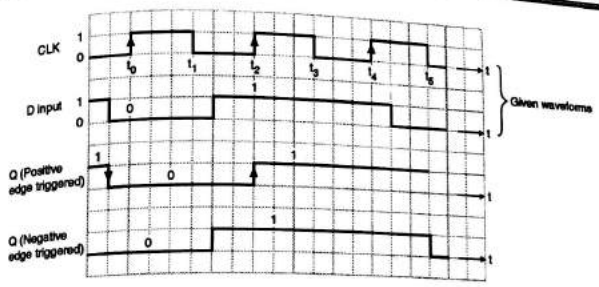
Transparent latch :

- In level triggered D FF (D latch) the Q output always follows the changes in D input.
- Hence D latch is sometimes called as transparent latch.

Ex. 7.9.1 : Draw the output waveform for a positive and negative edge triggered D flip flop, if the clock and D input waveforms are as shown in Fig. P. 7.9.1.

Soln. :

The required waveforms are as shown in Fig. P. 7.9.1.



(C-3421) **Fig. P. 7.9.1**

7.10 Edge Triggered J-K Flip Flop :

- Edge triggered J-K flip flops are of two types :

1. Positive edge triggered JK flip flop 2. Negative edge triggered JK flip flop

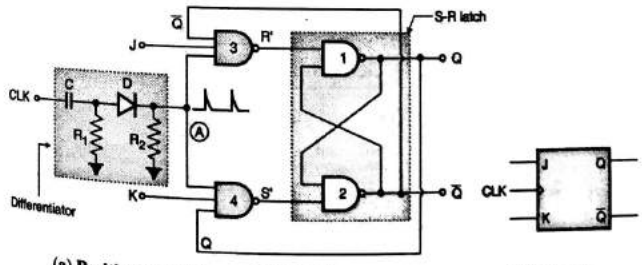
7.10.1 Positive Edge Triggered JK Flip Flop :

MU : Dec. 02

University Questions

Q.1 Design a clocked J–K flipflop using a basic NAND cell using NAND gates only. **(Dec. 02, 8 Marks)**

- The circuit diagram of the positive edge triggered JK flip flop is shown in Fig. 7.10.1 (a) and its circuit symbol is shown in Fig. 7.10.1 (b).



(a) Positive edge triggered JK flip flop **(b) Symbol**

(C-596) **Fig. 7.10.1**

Operation :

- The clock signal is a train of rectangular or square waves.
- It is passed through a differentiator ($C - R_1$) and a rectifier (D and R_2), to obtain only the positive spikes at point A as already discussed. The negative spikes are blocked by the diode.
- NAND gates 1 and 2 form the basic S-R latch. The other two NAND gates (3 and 4) have three inputs each. The outputs Q and $\bar{Q}$ are fed back to the inputs of gates 4 and 3 respectively.

Scanned by CamScanner

- Referring to Fig. 7.10.1 (a) we can write the mathematical expressions for S′ and R′ as follows.

$$S' = \overline{K \cdot Q \cdot CLK} \quad \text{and} \quad R' = \overline{J \cdot \bar{Q} \cdot CLK}$$

- Let us now understand the operation step by step.

Case I : CLK = 0 or 1 i.e. level

- If CLK = 0 or 1 i.e. level and no pulse this flip flop is disabled, because the output of differentiator is zero for any level input.
- Therefore Q and $\bar{Q}$ output do not change their state.

Case II : If clock edge is falling edge (↓)

- For the falling edge of the clock, the rectifier (D – R_2) will block the negative spike. Hence voltage at point A is logic 0.
- So one input to both the NAND gates 3 and 4 is 0.
- Hence both S′ and R′ will be forced to 1.
- For an SR latch using NAND gates, the outputs will remain unchanged if S = R = 1.

∴ Corresponding to the falling edge of clock, the outputs Q and $\bar{Q}$ will remain unchanged.

Case III : CLK = ↑, J = 0, K = 0 (No change)

- Since J = 0, R′ = 1 and as K = 0, S′ = 1.
- As S′ = R′ = 1 the outputs Q and $\bar{Q}$ will not change their state even though a positive edge of the clock pulse is being applied.

$$\therefore J = 0, K = 0, Q_{n+1} = Q \text{ and } \bar{Q}_{n+1} = \bar{Q}. \text{ No change in output.}$$

Case IV : CLK = ↑, J = 0, K = 1 (Reset)

- Recall the expressions for S′ and R′,

$$S' = \overline{K \cdot Q \cdot CLK} \quad \text{and} \quad R' = \overline{J \cdot \bar{Q} \cdot CLK}$$

- If the previous state of Q and $\bar{Q}$ is Q = 1 and $\bar{Q}$ = 0 then

$$S' = \overline{1 \cdot 1 \cdot 1} = 0 \quad \text{and} \quad R' = \overline{0 \cdot 0 \cdot 1} = 1$$

- Therefore according to the operation of S-R latch if S′ = 0, R′ = 1 then Q = 0, and $\bar{Q}$ = 1.

$$\therefore Q = 0 \text{ and } \bar{Q} = 1$$

- Thus with J = 0, K = 1 and positive going clock, the JK flip flop will **reset.**
- If Q = 0 and $\bar{Q}$ = 1 before the application of clock pulse, then there will not be any change in their state.

Case V : CLK = ↑, J = 1, K = 0 (Set)

- If the previous state of Q and $\bar{Q}$ is Q = 0 and $\bar{Q}$ = 1 then,

$$S' = \overline{0 \cdot 0 \cdot 1} = 1 \quad \text{and} \quad R' = \overline{1 \cdot 1 \cdot 1} = 0$$

- Hence according to the operation of S-R latch, if S′ = 1 and R′ = 0 then,

$$Q = 1 \text{ and } \bar{Q} = 0 \text{.... i.e. the flip flop is set.}$$

- If Q and $\bar{Q}$ are already 1,0 then there will not be any change in the output state.

Case VI : CLK = ↑, J = 1, K = 1 Toggle mode

We will discuss this case for two different previous cases.

• Previously let Q = 1 and $\bar{Q}$ = 0.	• Previously let Q = 0 and $\bar{Q}$ = 1
↓	↓
• $S' = \overline{K \cdot Q \cdot CLK}$ and $R' = \overline{J \cdot \bar{Q} \cdot CLK}$	• $S' = \overline{1 \cdot 0 \cdot 1}$ and $R' = \overline{1 \cdot 1 \cdot 1}$
↓	↓
∴ $S' = \overline{1 \cdot 1 \cdot 1} = 0$ and $R' = \overline{1 \cdot 0 \cdot 1} = 1$	• So S′ = 1 and R′ = 0
↓	↓
• Hence the SR latch will reset	• Hence the SR latch will set
↓	↓
• So $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$	• So $Q_{n+1} = 1$ and $\bar{Q}_{n+1} = 0$

- From the operation discussed above, we conclude that when J = K = 1 and a positive clock edge is applied, then Q and $\bar{Q}$ outputs are inverted i.e. $Q_{n+1} = \bar{Q}_n$ and $\bar{Q}_{n+1} = Q_n$.
- This is called as the **TOGGLING** mode. This is a very important mode of operation.

Truth table of JK flip flop :

Table 7.10.1 : Truth table for positive edge triggered JK FF

Case No.	Inputs			Outputs		State
	CLK	J	K	Q_{n+1}	$\bar{Q}_{n+1}$	
Case I	0 or 1	×	×	No change	No change	Flip flop is disabled
Case II	↓	×	×	No change	No change	
Case III	↑	0	0	No change	No change	
Case IV	↑	0	1	0	1	Reset
Case V	↑	1	0	1	0	Set
Case VI	↑	1	1	$\bar{Q}_n$	Q_n	Toggle

Scanned by CamScanner

7.10.2 Characteristic Equation of JK Flip Flop :

MU : May 07

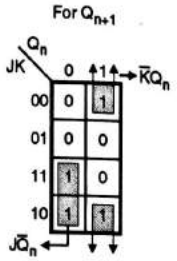
University Questions

Q. 1 Write the truth table and excitation of J-K flip flop. **(May 07, 2 Marks)**

Refer to the truth table of JK flip flop (Table 7.10.2) and write the K-map for Q_{n+1} as shown in Fig. 7.10.2.

Table 7.10.2 : Truth table

J	K	Q_n	Q_{n+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



(C-597) **Fig. 7.10.2 : K-map for $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$**

$\therefore$ The characteristic equation is $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

7.10.3 How does an Edge Triggered JK FF Avoid Race Around Condition ?

MU : Dec. 02, Dec. 04, May 06

University Questions

Q. 1 Why does it occur ? What modification is required if race condition is to be avoided ? **(Dec. 02, 2 Marks)**

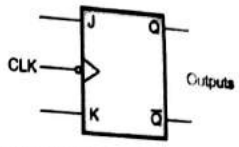
Q. 2 What is race around condition ? Is it in R.S. flip-flop ? How is it avoided in J-K flip-flop ? **(Dec. 04, 5 Marks)**

Q. 3 How do you remove the "Race Around Condition" ? **(May 06, 4 Marks)**

- For the race around to take place, it is necessary to have the enable input high along with J = K = 1.
- As the enable input remains high for a long time in a JK latch, the problem of multiple toggling arises which is known as Race condition.
- But in edge triggered JK flip flop, the positive clock pulse is present only for a very short time because it is in the form of a narrow differentiated spike.
- Hence by the time the toggled outputs (Q and $\bar{Q}$) return back to the inputs of NAND gates 3 and 4, the positive clock spike has died down to zero. Hence the multiple toggling can not take place.
- Thus the edge triggering avoids the race around condition.

7.10.4 Negative Edge Triggered JK Flip-Flop :

- The symbol for negative edge triggered JK flip-flop is shown in Fig. 7.10.3.
- This FF responds only to the negative edges of the clock pulses.
- The rest of the operation of this FF is exactly same as that of the positive edge triggered JK FF.
- The truth table of negative edge triggered JK FF is as shown in Table 7.10.3.



(C-598) **Fig. 7.10.3 : Symbol of negative edge triggered JK FF**

Table 7.10.3 : Truth table of negative edge triggered JK FF

Inputs			Outputs		State
CLK	J	K	Q_{n+1}	$\bar{Q}_{n+1}$	
0 or 1	×	×	Q_n	$\bar{Q}_n$	NC (FF is disabled)
↑	×	×	Q_n	$\bar{Q}_n$	
↓	0	0	Q_n	$\bar{Q}_n$	No change
↓	0	1	0	1	Reset
↓	1	0	1	0	Set
↓	1	1	$\bar{Q}_n$	Q_n	Toggle

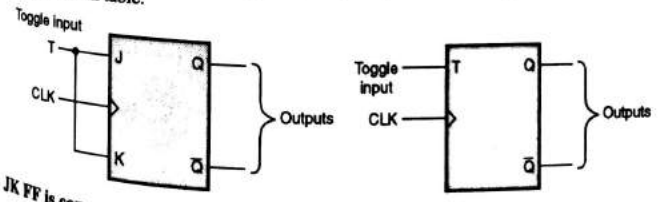
7.11 Toggle Flip Flop (T Flip Flop) :

MU : May 05

University Questions

Q. 1 Design a J-K flip flop using basic NAND cell. Design T-flip flop using same method. **(May 05, 5 Marks)**

- Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only one input denoted by "T", as shown in Fig. 7.11.1(a).
- The symbol for positive edge triggered T flip flop is shown in Fig. 7.11.1(b) and Table 7.11.1 shows its truth table.



(a) JK FF is converted into T flip flop (b) Logic symbol of positive edge triggered T flip flop

(C-599) **Fig. 7.11.1**

Scanned by CamScanner

Table 7.11.1 : Truth table of a Toggle FF (positive edge triggered)

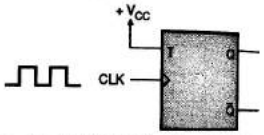
Inputs		Outputs		State
CLK	T	Q_{n+1}	$\bar{Q}_{n+1}$	
↑	0	Q_n	$\bar{Q}_n$	No change
↓	×	Q_n	$\bar{Q}_n$	No change
1	×	Q_n	$\bar{Q}_n$	No change
0	×	Q_n	$\bar{Q}_n$	No change
↑	1	$\bar{Q}_n$	Q_n	Toggle

- When T = 0, J = K = 0. Hence the outputs Q and $\bar{Q}$ won't change.
- But if T = 1, then J = K = 1 and the outputs will toggle corresponding to every leading edge of clock signal.
- This has been illustrated in Ex. 7.11.1.

Ex. 7.11.1 : Determine the output Q for the set up shown in Fig. P. 7.11.1(a). What is the relation between the frequency of clock and that of Q output ?

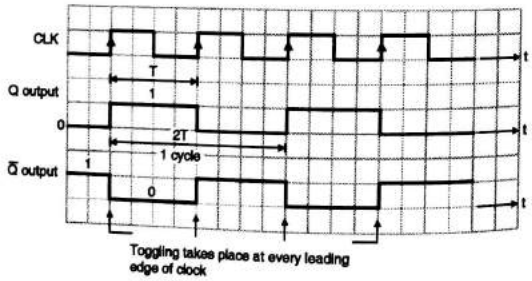
Soln. :

- The flip flop of Fig. P. 7.11.1(a) is a toggle flip flop with T = 1. It is a positive edge triggered flip flop.



(C-600) **Fig. P. 7.11.1(a) : Given set up**

- Hence the outputs will toggle in response to each rising edge of the clock, as shown in Fig. P. 7.11.1(b).



(C-601) **Fig. P. 7.11.1 (b) : Waveforms for a T F/F**

Relation between input and output frequency :

- Referring to Fig. P. 7.11.1(b),

  1 cycle period of CLK signal = T, ∴ Clock frequency $f_{CLK} = \frac{1}{T}$

  1 cycle period of Q output = 2 T, ∴ Output frequency $f_o = \frac{1}{2T}$

- But $(\frac{1}{T}) = f_{CLK}$

$$\therefore \text{Output frequency } f_o = \frac{f_{CLK}}{2}$$

...Ans.

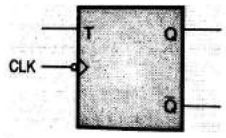
The T flip flop divides the clock frequency by 2. Hence a T flip flop can be used as a frequency divider.

7.11.1 Negative Edge Triggered T Flip Flop :

- Fig. 7.11.2 shows the logic symbol of a negative edge triggered toggle flip flop and Table 7.11.2 gives its truth table.

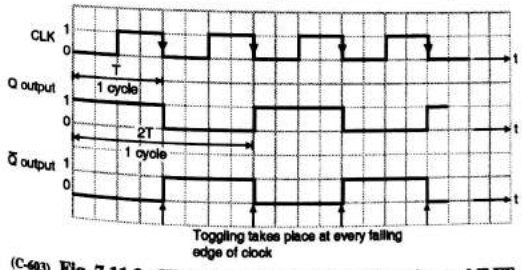
Table 7.11.2 : Truth table

Inputs		Outputs		State
T	CLK	Q_{n+1}	$\bar{Q}_{n+1}$	
0	×	Q_n	$\bar{Q}_n$	No change
1	↑	Q_n	$\bar{Q}_n$	No change
1	0	Q_n	$\bar{Q}_n$	No change
1	1	Q_n	$\bar{Q}_n$	No change
1	↓	$\bar{Q}_n$	Q_n	Toggle



(C-602) **Fig. 7.11.2 : Logic symbol of a negative edge triggered toggle flip flop**

- The waveforms for negative edge triggered toggle FF are shown in Fig. 7.11.3.



(C-603) **Fig. 7.11.3 : Waveforms for negative edge triggered T FF**

- The toggle input T = 1. The outputs Q and $\bar{Q}$ toggle corresponding to every falling edge of clock.
- This F/F also, acts as a frequency divider. The frequency of Q and $\bar{Q}$ outputs is half the frequency of clock input.

Scanned by CamScanner

7.11.2 Application of T F/F :

The T F/F acts as the basic building block of a ripple counter.

7.12 Master Slave (MS) JK Flip Flop :

MU : Dec. 02, Dec. 03, Dec. 07, Dec. 08, May 13

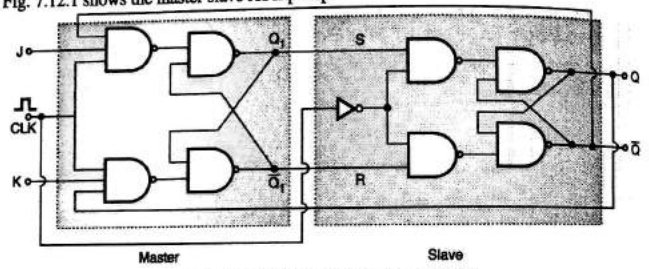
University Questions

Q. 1 Write note on : Master slave J-K flip flop. **(Dec. 02, Dec. 03, Dec. 07, 6 Marks)**

Q. 2 What is race condition ? How it is overcome in Master-slave J.K flip flop ? Explain. **(Dec. 08, 5 Marks)**

Q. 3 Explain working of Master-Slave J-K flip flop. **(May 14, 10 Marks)**

- Fig. 7.12.1 shows the master slave JK flip flop.



(C-606) **Fig. 7.12.1 : Master slave JK FF**

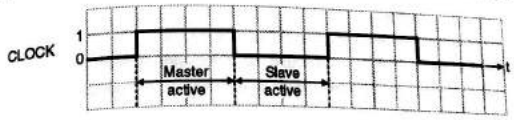
- It is a combination of a clocked JK latch (level triggered JK FF) and clocked SR latch (level triggered SRFF).
- The clocked JK latch acts as the master and the clocked SR latch acts as the slave.
- Master is positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Thus both master and slave circuits are level triggered circuits.
- Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and the master is inactive.
- Table 7.12.1 gives truth table of master slave JK flip flop.

Table 7.12.1 : Truth table of master slave JK FF

Case	Inputs			Outputs		Remark
	CLK	J	K	Q_{n+1}	$\bar{Q}_{n+1}$	
I	×	0	0	Q_n	$\bar{Q}_n$	No change
II	⎍ (1)	0	0	Q_n	$\bar{Q}_n$	No change
III	⎍ (1)	0	1	0	1	Reset
IV	⎍ (1)	1	0	1	0	Set
V	⎍ (1)	1	1	$\bar{Q}_n$	Q_n	Toggle

Operation :

- We will discuss the operation of the master slave JK FF with reference to its truth table.
- We must always remember one important thing that in the positive half cycle of the clock, the master is active and in the negative half cycle, the slave is active. This is shown in Fig. 7.12.2.



(C-607) **Fig. 7.12.2**

Case I : Clock = ×, J = K = 0 (No change)

- For clock = 1, the master is active, slave inactive. As J = K = 0. ∴ Outputs of master i.e. Q_1 and $\bar{Q}_1$ will not change. Hence the S and R inputs to the slave will remain unchanged.
- As soon as clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged.

∴ The outputs will not change if J = K = 0.

Case II : Clock = ⎍, J = K = 0 (No change)

This condition has been already discussed in case I.

Case III : Clock = ⎍, J = 0 and K = 1 (Reset)

- Clock = 1 : Master active, slave inactive.
- ∴ Outputs of the master become $Q_1 = 0$ and $\bar{Q}_1 = 1$. That means S = 0 and R = 1.
- Clock = 0 : Slave active, master inactive
- ∴ Outputs of the slave become Q = 0 and $\bar{Q} = 1$. This is the RESET operation.
- Again if clock = 1 : Master active, slave inactive.

∴ Even with the changed outputs Q = 0 and $\bar{Q} = 1$ fed back to master, its outputs will be $Q_1 = 0$ and $\bar{Q}_1 = 1$. That means S = 0 and R = 1.

- Hence with clock = 0 and slave becoming active, the outputs of slave will remain Q = 0 and $\bar{Q} = 1$.
- Thus we get a stable output from the Master slave.

Case IV : CLK = ⎍, J = 1, K = 0 (Set)

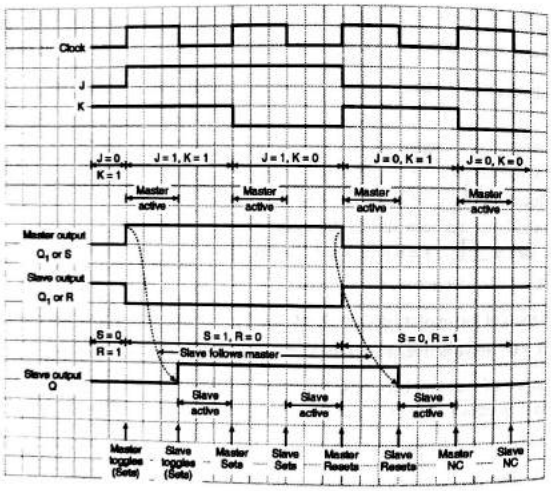
- Clock = 1 : Master active, slave inactive.
- ∴ Outputs of master become $Q_1 = 1$ and $\bar{Q}_1 = 0$ i.e. S = 1, R = 0.
- Clock = 0 : Master inactive, slave active.
- ∴ Outputs of slave become Q = 1 and $\bar{Q} = 0$.

Again if clock = 1 then it can be shown that the outputs of the slave are stabilized to Q = 1 and $\bar{Q} = 0$.

Scanned by CamScanner

Case V : CLK = ⎍, J = 1, K = 1 (Toggle without Race)

- Clock = 1 : Master active, slave inactive.
  ∴ Outputs of master will toggle. So S and R also will be inverted.
- Clock = 0 : Master inactive, slave active.
  ∴ Outputs of the slave will toggle.
- These changed output are returned back to the master inputs.
- But since clock = 0, the master is still inactive. So it does not respond to these changed outputs.
- This avoids the multiple toggling which leads to the race around condition. Thus the master slave flip flop will avoid the race around condition.
- The waveforms for the master slave flip flop are shown in Fig. 7.12.3.



(C-605) **Fig. 7.12.3 : Waveforms of master slave JK flip flop**

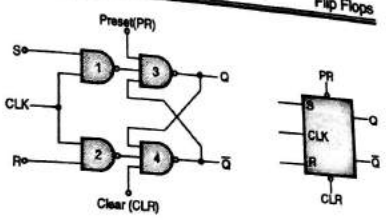
Observations from the waveforms :

- We can make the following important observations from the waveforms of the master slave JK FF :
  1. The slave always follows the master, after a delay of half clock cycle period.
  2. The multiple toggling or the race around condition is successfully avoided.

7.13 Preset and Clear Inputs :

- In the flip flops discussed so far when the power switch is turned on, the state of outputs Q and $\bar{Q}$ can be anything that means either Q = 0, $\bar{Q}$ = 1 or Q = 1, $\bar{Q}$ = 0. It is not predictable.
- But this uncertainty cannot be tolerated in certain applications. In some applications it is necessary to initially set or reset the flip flops precisely.
- This can be practically achieved by adding two more inputs to a flip flop, called **preset (PR)** and **clear (CLR)** inputs.

- These inputs are called as **direct** or **asynchronous inputs** because we can apply them any time between clock pulses without thinking about their synchronization with the clock. That means applying PR or CLR inputs is not related to the clock in any way.
- The S-R flipflop with preset and clear is shown in Fig. 7.13.1(a) and its symbol is shown in Fig. 7.13.1(b).



(a) S-R FF with preset and clear **(b) Symbol**

(C-609) **Fig. 7.13.1**

- Note that both these inputs are active low inputs. This is indicated by the bubbles on these inputs in Fig. 7.13.1(b).

Operation :

Case I : PR = CLR = 1

- If PR = CLR = 1 then both are inactive and the S-R FF operates as per the truth table of the conventional SR flip-flop.

Case II : PR = 0 and CLR = 1 (Preset condition)

- As PR = 0, the output of gate 3 of Fig. 7.13.1(a) will be 1.
- That means Q = 1.
- Therefore all the three inputs to the gate 4 will be 1 and $\bar{Q}$ will become 0.
- Thus making PR = 0 will set the flip flop. Note that with PR = 0 and CLR = 1 the flip flop is set irrespective of the status of S, R or clock inputs. Therefore it is said that the PR input has higher priority than S, R or CLK inputs.

Case III : PR = 1, CLR = 0 (Clear or Reset condition)

- If PR = 1 and CLR = 0, then the output of gate-4 i.e. $\bar{Q}$ = 1.
- Therefore all the three inputs to gate-3 will be 1 and hence Q = 0.
- Thus making CLR = 0 will reset the flip flop. Note that with CLR = 0 and PR = 1, the FF is reset irrespective of the status of S, R or clock inputs. Therefore it is said that CLR has higher priority than S, R or CLK inputs.

Case IV : PR = 0, CLR = 0

- This condition should never be used, because it leads to an uncertain state.
- The operation of SR flip flop with preset and clear inputs is summarized in Table 7.13.1.

Table 7.13.1 : Summary of operation of SR FF

Inputs			Output	Operation performed
CLK	PR	CLR		
1	1	1	Q_{n+1}	Normal S-R flip flop
×	0	1	1	FF is set.
×	1	0	0	FF is reset.

- The don't care conditions (×) marked in the CLK column indicate that PR and CLR inputs have higher priority than clock input.

7.13.1 Synchronous Preset and Clear Inputs :

- Preset and clear inputs can be of two types :
  1. Asynchronous with clock.
  2. Synchronous with clock.

Scanned by CamScanner

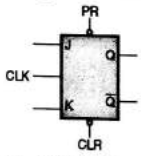
- We have already discussed the asynchronous preset and clear inputs.
- The operation of synchronous preset and clear inputs can be understood from the truth table given in Table 7.13.2.

Table 7.13.2 : Truth table for S-R FF with synchronous preset and clear

Inputs					Outputs	
P_r	C_r	CLK	S	R	Q_{n+1}	$\bar{Q}_{n+1}$
×	×	0	×	×	NC	NC
0	0	1	×	×	RACE	RACE
0	1	1	×	×	0	1
1	0	1	×	×	1	0
1	1	1	0	0	NC	NC
1	1	1	0	1	0	1
1	1	1	1	0	1	0
1	1	1	1	1	RACE	RACE

7.13.2 JK Flip Flop with Preset and Clear Inputs :

- A JK flip flop with preset and clear inputs is shown in Fig. 7.13.2. These are the synchronous preset and clear terminals.
- Both these inputs are active low and have higher priority than all the other inputs.



(a) Symbol of a JK FF with preset and clear inputs

Inputs			Output	Operation performed
CLK	PR	CLR		
1	1	1	Q_{n+1}	Normal JK FF
×	0	1	1	FF is set
×	1	0	0	FF is reset

(b) Summary of operation of SR FF

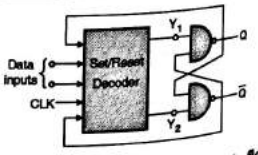
(C-610) **Fig. 7.13.2**

7.13.3 Applications of JK Flip Flop :

1. Shift register 2. Counters

7.14 Design of a Clocked Flip Flop :

- In this section we will develop a generalised design procedure that is applicable to all types of flip flops. We should be able to design any flip flop by using this procedure.
- Fig. 7.14.1 shows the general model of flip flop. It consists of the basic binary cell coupled with a set-reset decoder.



(C-3621) **Fig. 7.14.1 : The general model of flip flop**

7.14.1 Design Procedure :

Step 1 : Study the given characteristic table (truth table) which specifies the desired inputs and outputs and make a truth table for Y_1 and Y_2 as output variables by considering the following points :

1. For the given condition do we need to set the cell ? ($Q_{n+1} = 1$).
2. For the given condition do we need to reset the cell ? ($Q_{n+1} = 0$).
3. For the given condition do we need to leave the cell as it is ? (No change).

Step 2 : Draw the K-maps for Y_1 and Y_2 outputs and minimize them.

Step 3 : Realize these equations and draw the complete circuit diagram using gates.

7.14.2 Design Example :

Ex. 7.14.1 : Design the clocked SR flip flop for the characteristic table given by Table P. 7.14.1.

Soln. :

- For each row in the given characteristic table, find the values of Y_1 and Y_2 keeping S-R FF in mind.
- For example, consider the first row of Table P. 7.14.1. $Q_n = 0$ and we want $Q_{n+1} = 0$. In order to achieve this Y_1 must equal to 1 because $\bar{Q}_n = 1$. (This will make NAND-1 output equal to 0). Similarly Y_2 can be 0 or 1 as $Q_n = 0$ ($\bar{Q}_{n+1} = \overline{Y_2 \cdot Q}$). Hence we have entered a (X) i.e. don't care for Y_2 is row-1.
- Similarly the other entries for Y_1 and Y_2 can be made.

Table P. 7.14.1 : Truth table

	Characteristic table					Truth table for outputs		
	CLK	S	R	Q_n	Q_{n+1}	Y_1	Y_2	
Disabled	0	0	0	0	0	1	X	Row-1
	0	0	0	1	1	X	1	
	0	0	1	0	0	1	X	
	0	0	1	1	1	X	1	
	0	1	0	0	0	1	X	
	0	1	0	1	1	X	1	
	0	1	1	0	0	1	X	S = R = 1 with CLK = 0 can happen.
	0	1	1	1	1	X	1	
Enabled	1	0	0	0	0	1	X	
	1	0	0	1	1	X	1	
	1	0	1	0	0	1	X	
	1	0	1	1	0	1	0	
	1	1	0	0	1	0	1	
	1	1	0	1	1	X	1	
	1	1	1	0	X	X	X	S = R = 1 with CLK = 1 must never happen
	1	1	1	1	X	X	X	

Note : For CLK = 0, the FF output will not change. So $Q_{n+1} = Q_n$ irrespective of S and R.

Scanned by CamScanner

Step 1 : K maps for Y_1 and Y_2 :

For Y_1

RQ_n \ $CLKS$	00	01	11	10
00	1	1	0	1
01	X	X	X	X
11	X	X	X	1
10	1	1	X	1

$\bar{S} + \overline{CLK}$

$\therefore Y_1 = \bar{S} + \overline{CLK} = \overline{S \cdot CLK}$

For Y_2

RQ_n \ $CLKS$	00	01	11	10
00	X	X	1	X
01	1	1	1	1
11	X	1	X	0
10	X	X	X	1

$\bar{R} + \overline{CLK}$

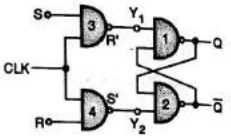
$\therefore Y_2 = \bar{R} + \overline{CLK} = \overline{R \cdot CLK}$

(C-3622) **Fig. P. 7.14.1(a) : K-maps and simplification**

Step 2 : Draw the logic circuit :

The circuit diagram of clocked R-S flip flop is shown in Fig. P. 7.14.1(b).

Note that this circuit is same as Fig. 7.13.1 (Section 7.13) discussed earlier in this chapter.



(C-3623) **Fig. P. 7.14.1(b) : Realization of S-R FF using gates**

7.15 Excitation Table of Flip-Flop :

- Till now we have written the truth tables of various flip flops. The truth tables are also known as the **characteristic tables.**
- But while designing the sequential circuits, sometimes the present and next state of a circuit are given and we are expected to find the corresponding input condition. We need to use the **excitation tables** of flip flops to do this. These tables are different from the characteristic tables.
- Present state is the state prior to application of clock pulse and the next state means the state after application of clock pulse.

7.15.1 Excitation Table of SR Flip Flops :

MU : Dec. 06, May 10

University Questions

Q. 1 Obtain excitation table for S- R flip/flop. **(Dec. 06, May 10, 3 Marks)**

- For example the outputs of an S-R FF before clock pulse are $Q_n = 0$ and $\bar{Q}_n = 1$ and it is expected that these outputs should remain unchanged after application of clock. Then what must be the values of inputs S_n and R_n to achieve this ?
- Refer to the truth table of SR FF to answer this question.

The answer is, for the following two conditions the outputs remain unchanged at $Q = 0$ and $\bar{Q} = 1$.

Condition 1 : S_n and $R_n = 0 \rightarrow$ Refer first row of Table 7.15.1.

Condition 2 : $S_n = 0$ $R_n = 1 \rightarrow$ Refer third row of Table 7.15.1.

Table 7.15.1 : Truth table of SR flip flop

	CLK	S_n	R_n	Q_{n+1}	$\bar{Q}_{n+1}$
Row	↑	0	0	Q_n	$\bar{Q}_n$
	↑	0	1	0	1
Row	↑	1	0	1	0
	↑	1	1	Race	Race

- From the two conditions mentioned above we conclude that S_n input should be equal to 0 and R_n input can be 0 or 1 (don't care) if we want to maintain $Q = 0$ and $\bar{Q} = 1$ before and after clock.
- Similarly we can find the input conditions (Values of S and R inputs) for all the possible situations that may exist on the output side.
- The table containing all these output situations and the corresponding input conditions is called as the "excitation table" of a flip flop.
- The excitation table of SR flip flop is shown in Table 7.15.2.

Table 7.15.2 : Excitation table of SR flip flop

	Present state of Q output	Next state of Q output	S_n input	R_n input
Case I	0	0	0	×
Case II	0	1	1	0
Case III	1	0	0	1
Case IV	1	1	×	0

Description of excitation table of SR FF :

We have already discussed case I.

Case II : Q should change from 0 to 1

- This is nothing but the set condition.
- Hence $S_n = 1$ and $R_n = 0$ should be the inputs.

Case III : Q should change from 1 to 0

- This is nothing but the reset condition.
- Hence S_n should be 0 and R_n should be 1.

Case IV : Q should be 1. No change

- To satisfy this requirement, we have two possible input conditions :

  Condition 1 : $S_n = R_n = 0$ i.e. No change in output.

  Condition 2 : $S_n = 1$ and $R_n = 0$.
- From these conditions we conclude that S_n can be either 0 or 1 i.e. don't care and $R_n = 0$. Hence the inputs corresponding to this case is

$$S_n = \times \quad \text{and} \quad R_n = 0$$

- Similarly we can write the excitation tables for the other flip flops.

7.15.2 Excitation Table of D Flip Flop :

MU : May 07, May 10

University Questions

Q. 1 Write excitation table of J-K Flip Flop. **(May 07, 2 Marks)**

Q. 2 Calculate the characteristics equation using characteristic table of D flip-flop. **(May 10, 2 Marks)**

Excitation table of a D flip flop is given by Table 7.15.3.

Table 7.15.3 : Excitation table of a D flip flop

Q output		Input
Present state	Next state	D_n
0	0	0
0	1	1
1	0	0
1	1	1

Scanned by CamScanner

7.15.3 Excitation Table of JK Flip Flop :

MU : May 10

University Questions

Q. 1 Calculate the characteristics equation using characteristic table of JK flip-flop. **(May 10, 2 Marks)**

Excitation table of a JK flip flop is given by Table 7.15.4.

Table 7.15.4 : Excitation table of JK flip flop

	Q output		Inputs	
	Present state	Next state	J_n	K_n
Case I	0	0	0	×
Case II	0	1	1	×
Case III	1	0	×	1
Case IV	1	1	×	0

Don't care conditions result due to toggling

- Refer to Case II of Table 7.15.4. For the change in Q output from 0 to 1 we have the following two input conditions :

  Condition I : $J_n = 1$ and $K_n = 0$ i.e. set condition.

  Condition II : $J_n = 1$ and $K_n = 1$ i.e. toggle.

- Hence $J_n = 1$ and $K_n = \times$ (0 or 1) corresponding to case II.
- Similarly for case III also, the don't care condition corresponds to toggle mode.

7.15.4 Excitation Table of T Flip Flop :

MU : May 10

University Questions

Q. 1 Calculate the characteristics equation using characteristic table of T flip-flop. **(May 10, 2 Marks)**

Excitation table of T flip flop is shown in Table 7.15.5.

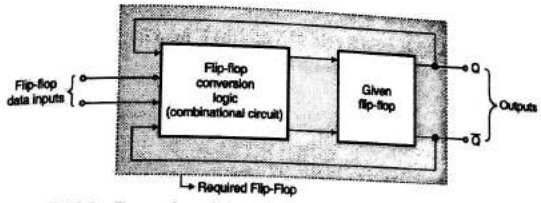
Table 7.15.5 : Excitation table of T flip-flop

Q output		Input
Present state	Next state	T_n
0	0	0
0	1	1
1	0	1
1	1	0

7.16 Conversion of Flip Flops :

- The conversion from one type of flip flop to the other (say SR FF to JKFF), needs a systematic approach using the excitation tables and K map simplifications.
- Fig. 7.16.1 shows a generalized model for conversion from one flip flop to the other.
- As shown in Fig. 7.16.1 the required flip flop is actually a combination of the given flip flop and a combinational logic circuit using gates. (Such a combinational circuit is called as the flip flop conversion logic).

- The inputs to FF conversion logic are, the flip flop data inputs and the outputs of given flip-flop i.e. the given FF and the desired FF.
- The conversion logic is designed by combining the excitation tables of both the flip flops.
- The truth table of the conversion logic has data inputs and Q and $\bar{Q}$ outputs of the given FF as inputs whereas the inputs of the given FF are the outputs of the truth table.
- Then we draw the K map for each output and obtain the simplified expressions. The conversion logic is then implemented using gates.



(C-611) **Fig. 7.16.1 : General model used to convert one type of FF to the other**

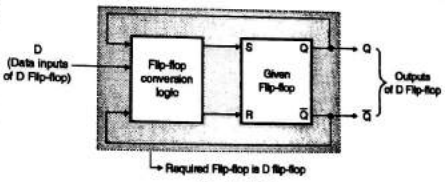
7.16.1 Conversion from S-R Flip Flop to D Flip Flop :

MU : Dec. 04, May 08, May 10, Dec. 10, May 12, Dec. 12, May 14, May 16

University Questions

Q. 1 Convert SR-flip-flop to D flip-flop. **(Dec. 04, May 08, May 10, Dec. 10, May 12, Dec. 12, May 14, May 16, 5 Marks)**

- Refer Fig. 7.16.2. Here the given FF is SR FF and the required FF is D FF.
- The truth table for the conversion logic is shown in Table 7.16.1. The inputs are D and Q whereas outputs are S and R.



(C-612) **Fig. 7.16.2**

- The truth table is prepared by combining the excitation tables of D F/F and SR FF.

Table 7.16.1 : Truth table for SR to D FF conversion

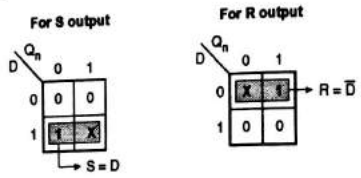
Inputs			Outputs	
D	Present state Q_n	Next state Q_{n+1}	S	R
0	0	0	0	×
1	0	1	1	0
0	1	0	0	1
1	1	1	×	0

← Entries from excitation table of D FF →

← Entries from excitation table of SR FF →

Scanned by CamScanner

- Now write the K maps for the S and R outputs as shown in Figs. 7.16.3(a) and (b).

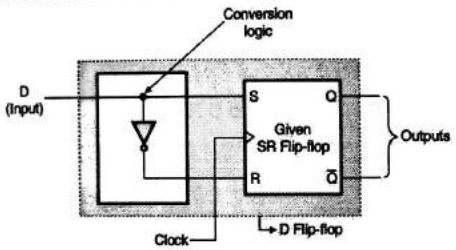


(a) K map for S (b) K map for R

(C-613) **Fig. 7.16.3**

Logic diagram :

The logic diagram for SR FF to D FF is shown in Fig. 7.16.4.



(C-614) **Fig. 7.16.4 : SR flip flop to D flip flop**

7.16.2 Conversion of JK FF to T FF :

MU : Dec. 15

University Questions

Q. 1 Convert JK flip-flop into T-flip-flop (show only the design without steps). **(Dec. 15, 2 Marks)**

Step 1 : Write the truth table for conversion :

The required truth table is obtained from the excitation tables of JK and T flip flops as follows :

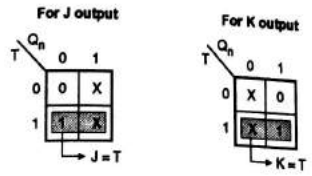
Table 7.16.2 : Truth table for conversion from JK to T FF

Inputs			Outputs	
T	Present state of Q	Next state of Q	J	K
0	0	0	0	×
1	0	1	1	×
1	1	0	×	1
0	1	1	×	0

← Excitation table of T FF →

← Excitation table of JK FF →

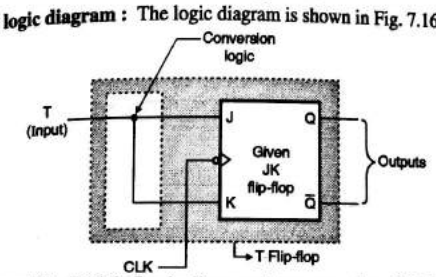
Step 2 : K maps and simplification : The K maps for outputs J and K are shown in Fig. 7.16.5.



(a) K map for J output (b) K map for K output

(C-615) **Fig. 7.16.5**

Step 3 : Draw the logic diagram : The logic diagram is shown in Fig. 7.16.6.



(C-616) **Fig. 7.16.6 : Logic diagram for conversion of JK FF to T FF**

7.16.3 SR Flip Flop to T Flip Flop :

MU : Dec. 04, Dec. 05, May 10, Dec. 10, May 15

University Questions

Q. 1 Convert SR FF to T FF and draw waveform.

(Dec. 04, Dec. 05, May 10, Dec. 10, May 15, 5 Marks)

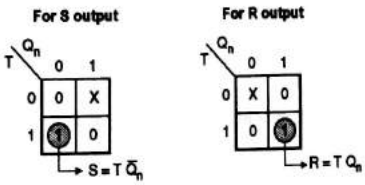
The stepwise conversion process is as follows :

Step 1 : Write the truth table :

Table 7.16.3 : Truth table for SR FF to T FF

Inputs			Outputs	
T	Present state Q_n	Next state Q_{n+1}	S	R
0	0	0	0	×
1	0	1	1	0
1	1	0	0	1
0	1	1	×	0

Step 2 : Write the K maps and obtain the expressions for S and R :



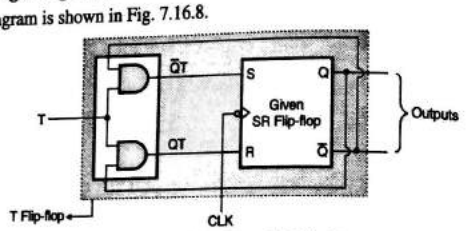
(a) K map for S (b) K map for R

(C-617) **Fig. 7.16.7**

Scanned by CamScanner

Step 3 : Draw the logic diagram :

The logic diagram is shown in Fig. 7.16.8.



(C-618) **Fig. 7.16.8 : Conversion from SR flip flop to T flip flop**

7.16.4 SR Flip Flop to JK Flip Flop :

MU : Dec. 04, May 12, Dec. 12, May 14, May 15, May 16

University Questions

Q. 1 Convert : SR FF to JK FF. **(Dec. 04, May 12, Dec. 12, May 14, May 15, May 16, 5 Marks)**

Step 1 : Write the truth table for SR to JK :

The truth table for SR to JK flip flop conversion is shown in Table 7.16.4.

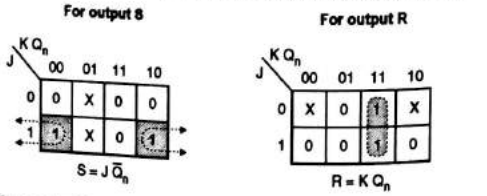
Table 7.16.4 : Truth table for SR to JK FF conversion

Inputs				Outputs	
J	K	Present state Q_n	Next state Q_{n+1}	S	R
0	0	0	0	0	×
0	1	0	0	0	×
1	0	0	1	1	0
1	1	0	1	1	0
0	1	1	0	0	1
1	1	1	0	0	1
0	0	1	1	×	0
1	0	1	1	×	0

Excitation table of JKFF

Excitation table of SR FF

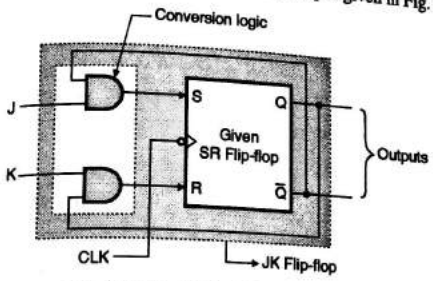
Step 2 : K maps and simplification : K maps for S and R outputs are shown in Fig. 7.16.9.



(a) K-map for S output (b) K-map for R output

(C-619) **Fig. 7.16.9**

Step 3 : Logic diagram : The logic diagram of SR to JK flip flop is given in Fig. 7.16.10.



(C-620) **Fig. 7.16.10 : SR to JK flip flop conversion**

7.16.5 Conversion of D Flip Flop to T Flip Flop :

Step 1 : Write the truth table for D to T FF conversion :

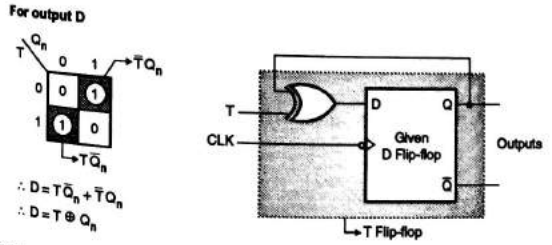
The truth table is as follows :

Table 7.16.5 : Truth table for D to T FF conversion

Inputs			Output
T	Present state Q_n	Next state Q_{n+1}	D
0	0	0	0
1	0	1	1
1	1	0	0
0	1	1	1

Step 2 : K map and simplification and logic diagram :

K map is shown in Fig. 7.16.11(a) and logic diagram is shown in Fig. 7.16.11(b).



(a) K map and simplification (b) Logic diagram

(C-621) **Fig. 7.16.11 : D flip flop to T flip flop**

Scanned by CamScanner

7.16.6 T Flip Flop to D Flip Flop Conversion :

MU : Dec. 10

University Questions

Q. 1 Convert T flip-flop to D flip-flop. **(Dec. 11, 10 Marks)**

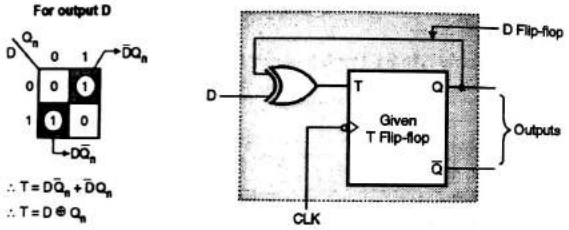
Step 1 : Write the truth table : The truth table is as given in Table 7.16.6.

Table 7.16.6 : Truth table for T FF to D FF conversion

Inputs			Output
D	Previous state Q_n	Next state Q_{n+1}	T
0	0	0	0
1	0	1	1
0	1	0	1
1	1	1	0

Step 2 : K maps simplification and logic diagram :

The K map is shown in Fig. 7.16.12(a) and the logic diagram is given in Fig. 7.16.12(b).



(a) K map for D output **(b) Logic diagram of T to D flip flop conversion**

(C-622) **Fig. 7.16.12**

7.16.7 JK Flip Flop to D Flip Flop Conversion :

MU : Dec. 05, May 11, May 14, Dec. 14, Dec. 15

University Questions

Q. 1 Convert JK FF to SR and D FF. **(Dec. 05, May 11, May 14, Dec. 14, Dec. 15, 5 Marks)**

Step 1 : Write the truth table for JK to D conversion : The truth table is as follows :

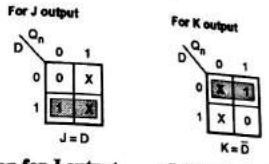
Table 7.16.7 : Truth table for JK to D flip flop conversion

	Inputs		Outputs	
D	Previous state Q_n	Next state Q_{n+1}	J	K
0	0	0	0	×
1	0	1	1	×
0	1	0	×	1
1	1	1	×	0

← Excitation table of D FF →

← Excitation table of JK FF →

Step 2 : K maps and simplification :

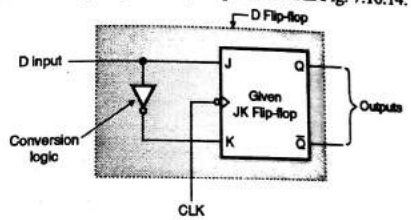


(a) K map for J output **(b) K map for K output**

(C-623) **Fig. 7.16.13**

Step 3 : Draw the logic diagram :

The logic diagram for JK flip flop to D flip flop is shown in Fig. 7.16.14.



(C-624) **Fig. 7.16.14 : Logic diagram for conversion from JK FF to D FF**

7.16.8 JK Flip Flop to SR Flip Flop Conversion :

MU : May 11, May 14, Dec. 14

University Questions

Q. 1 Convert JK FF to SR and D FF. **(May 11, May 14, Dec. 14, 5 Marks)**

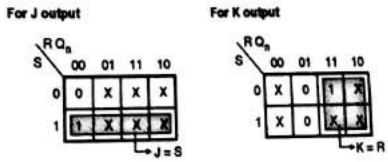
Step 1 : Write the truth table for JK to SR :

The truth table is shown in Table 7.16.8.

Table 7.16.8 : Truth table for JK to SR conversion

Inputs				Outputs	
S	R	Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	0	0	×
0	1	0	0	0	×
1	0	0	1	1	×
1	0	0	1	1	×
0	1	1	0	×	1
0	0	1	0	×	1
0	0	1	1	×	0
1	0	1	1	×	0

Step 2 : K-maps and simplifications :



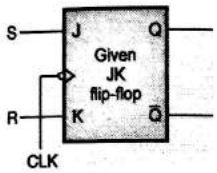
(a) **(b)**

(C-625) **Fig. 7.16.15**

Scanned by CamScanner

Step 3 : Logic diagram :

The logic diagram for JK to SR FF is shown in Fig. 7.16.15(c).



(C-626) **Fig. 7.16.15(c) : JK to SR FF conversion**

7.16.9 D FF to SR FF Conversion :

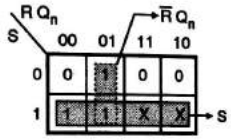
Step 1 : Write the truth table for D to S-R Conversion :

Table 7.16.9 : Truth table for D to S-R conversion

Inputs				Output
S	R	Present state Q_n	Next state Q_{n+1}	D
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
0	1	1	0	0
0	0	1	1	1
1	0	1	1	1

Step 2 : K maps and simplification :

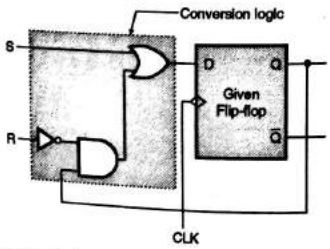
For D



$\therefore D = S + \bar{R} Q_n$

(C-627) **Fig. 7.16.16 : K-map and simplification**

Step 3 : Logic diagram :



(C-628) **Fig. 7.16.17 : Logic diagram for D to S-R FF conversion**

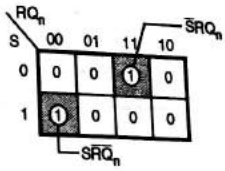
7.16.10 T FF to SR FF Conversion :

Step 1 : Write the truth table for T to S-R conversion :

Table 7.16.10 : Truth table for T to S-R conversion

Inputs				Output
S	R	Present state Q_n	Next state Q_{n+1}	T
0	X	0	0	0
1	0	0	1	1
0	1	1	0	1
X	0	1	1	0

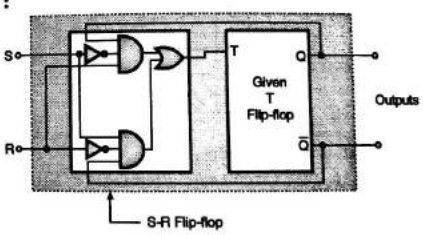
Step 2 : K maps and simplification for T output :



$\therefore T = S\bar{R}\bar{Q}_n + \bar{S} R Q_n$

(C-629) **Fig. 7.16.18 : K-map and simplification**

Step 3 : Logic diagram :



(C-630) **Fig. 7.16.19 : Logic diagram for T to S-R FF conversion**

7.16.11 Conversion from D FF to JK FF :

Step 1 : Write the truth table :

Table 7.16.11 shows the truth table obtained from the excitation tables of D and JK FFs.

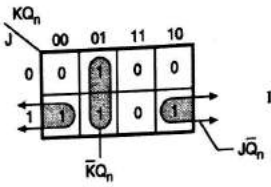
Table 7.16.11 : Truth table for D to JK conversion

Inputs				Output
J	K	Present state Q_n	Next state Q_{n+1}	D
0	X	0	0	0
1	X	0	1	1
X	1	1	0	0
X	0	1	1	1

Scanned by CamScanner

Step 2 : K maps and simplification :

For output D

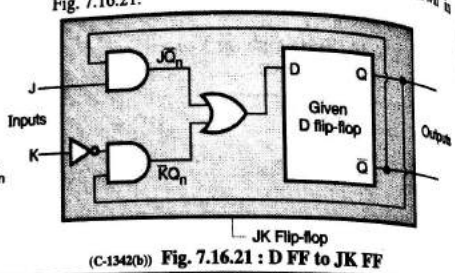


$\therefore D = \overline{K}Q_n + J\overline{Q}_n$

(C-1342(a)) **Fig. 7.16.20**

Step 3 : Logic diagram :

The logic diagram for D FF to JK FF is shown in Fig. 7.16.21.



(C-1342(b)) **Fig. 7.16.21 : D FF to JK FF**

Ex. 7.16.1 : Convert a SR flip flop to a MN flip flop where MN flip flop characteristic table is : **May 03, Dec. 05, 10 Marks**

M	N	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	$\overline{Q}_n$

Soln. :

Step 1 : **Write the circuit excitation table and state table :**

Table P. 7.16.1(a) : Circuit excitation table of SR F/F

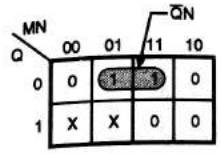
Present state	Next state	FF inputs	
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Table P. 7.16.1(b) : State table

Inputs Present state			Next state	FF Outputs	
Q	M	N	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	1	0
0	1	0	0	0	X
0	1	1	1	1	0
1	0	0	1	X	0
1	0	1	1	X	0
1	1	0	0	0	1
1	1	1	0	0	1

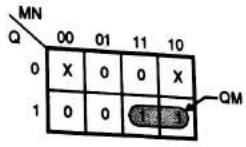
Step 2 : **K-maps and simplification :**

For S output



$\therefore S = \overline{Q}N$

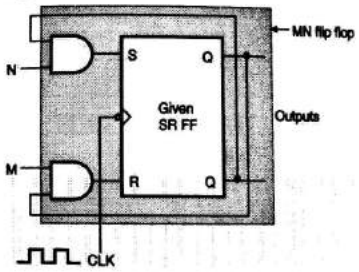
For R output



$\therefore R = QM$

(C-3718) **Fig. P. 7.16.1(a)**

Step 3 : **Draw the logic diagram :**



(C-3719) **Fig. P. 7.16.1(b)**

Ex. 7.16.2 : Convert JK flip flop to MN flip flop where MN flip flop truth table is **Dec. 03, 8 Marks**

M	N	Q_{n+1}
0	0	1
0	1	0
1	0	$\overline{Q}_n$
1	1	Q_n

Soln. :

Step 1 : **Write the circuit excitation table and state table :**

Table P. 7.16.2(a) : Circuit excitation table of JK F/F

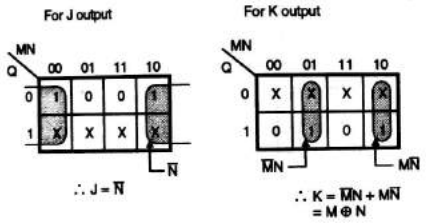
Present state	Next state	FF inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Scanned by CamScanner

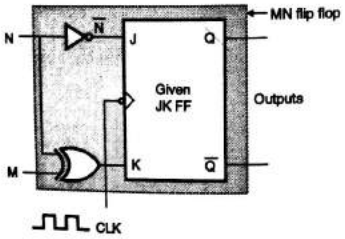
Table P. 7.16.2(b) : State table

Inputs Present state			Next state	FF Outputs	
Q	M	N	Q_{n+1}	J	K
0	0	0	1	1	X
0	0	1	0	0	X
0	1	0	1	1	X
0	1	1	0	0	X
1	0	0	1	X	0
1	0	1	0	X	1
1	1	0	0	X	1
1	1	1	1	X	0

Step 2 : K-maps and simplification :

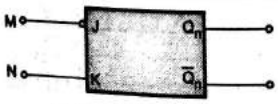


(C-3720) **Fig. P. 7.16.2(a)**



(C-3721) **Fig. P. 7.16.2(b)**

Ex. 7.16.3 : Consider the MN FF as shown in Fig. P. 7.16.3(a). Obtain its characteristics table, characteristics equation and excitation table.



(C-3722) **Fig. P. 7.16.3(a)**

Soln. :

Table P. 7.16.3(a) : Excitation table

N	M	Q_n	Q_{n+1}	J	K
0	0	0	1	1	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	1	X	0
1	0	0	1	1	X
1	0	1	0	X	1
1	1	0	0	0	X
1	1	1	0	X	1

K-map simplification :

(C-3723) **Fig. P. 7.16.3(b)**

Characteristic equation :

$$Q_{n+1} = \overline{M}\,Q_n + N\,\overline{Q}_n$$

Table P. 7.16.3(b) : Characteristic table

N	M	Q_n	Q_{n+1}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Scanned by CamScanner

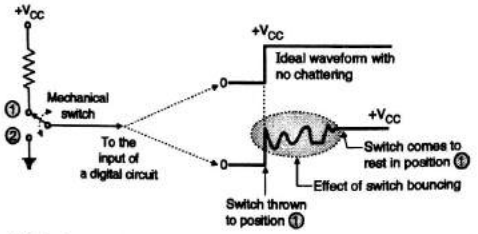
7.17 Applications of Flip Flops :

Some of the important applications of the flip flops are :

1. Elimination of keyboard debounce
2. As a memory element
3. In various types of registers
4. In counters/timers.
5. As a delay element.

7.17.1 Application of SR Latch for Elimination of Keyboard Debounce :

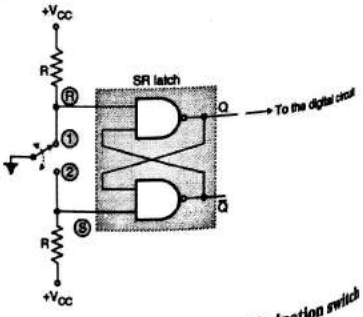
- Sometimes we need to use mechanical switches as input devices in order to enter digital information (0 or 1) into a digital system.
- **Switch bouncing or chattering** is a very serious problem associated with these switches.
- When the pole of a switch (also called as the arm) is thrown from say position 2 to 1, (see Fig. 7.17.1(a)), it chatters or bounces a number of times before finally coming to rest in position 1.
- The effect of this unwanted bouncing has been illustrated in Fig. 7.17.1(b). The input to the logic circuit will oscillate due to switch chattering.



(a) Mechanical switch **(b) Effect of switch bouncing**

(C-641) **Fig. 7.17.1**

- This produces an oscillatory output from a sequential circuit and creates problems in the operation of the system.
- This problem can be sorted out by using bounce elimination switch as shown in Fig. 7.17.1(a).
- Fig. 7.17.2 shows how we can use the SR latch for creating a bounce elimination switch.

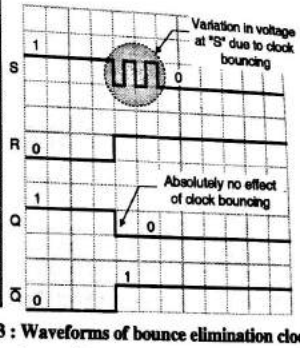


(C-642) **Fig. 7.17.2 : A bounce elimination switch**

Operation :

- Initially let the switch be in position 1
- Therefore S = 1 and R = 0. Hence Q = 1
- Now the switch is thrown to position 2
- Therefore S = 0 and R = 1 as the switch first ∴ Q = 0
- Now if the switch bounces off the contact 2, then S = R = 1
- Hence Q output will remain unchanged. Hence Q = 0
- After some bounces, the switch comes to rest at position 2

(C-643)



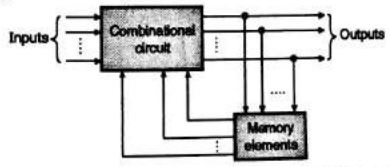
(C-644) **Fig. 7.17.3 : Waveforms of bounce elimination clock**

Thus at the Q output of SR latch we get a clean waveform without any oscillations as shown in Fig. 7.17.3.

Note : The other applications of Flip Flops such as registers and counters have been discussed in chapters 8 and 9 of this book.

7.18 Sequential Circuits :

- The general model of a sequential circuit is shown in Fig. 7.18.1. It shows that a sequential circuit is made of a combinational circuit alongwith the memory elements.
- The memory elements will be always connected between output and input of the combinational circuit. Therefore it is said to be appearing in the feedback path.



(C-652) **Fig. 7.18.1 : General model of a sequential circuit**

- The output of a sequential circuit depends on the present state as well as the past state.
- The present state is stored in the memory elements. These present states are applied to the combinational circuit alongwith the external inputs. Thus the outputs of the combinational block i.e. the outputs and the next state of the sequential circuit are decided by the external inputs and the present state.

7.18.1 Classification of Sequential Circuits :

The sequential circuits are classified into two categories as follows :

1. Synchronous sequential circuits 2. Asynchronous sequential circuits

Synchronous sequential circuits :

In the synchronous sequential circuits, the contents of memory elements can be changed only at the rising or falling edges of clock signal.

Scanned by CamScanner

Asynchronous sequential circuits :

In the asynchronous sequential circuit, the contents of memory elements can be changed at any instant of time.

Table 7.18.1 : Comparison of synchronous and asynchronous sequential circuits

Sr. No.	Synchronous sequential circuits	Asynchronous sequential circuits
1.	These circuits are easy to design.	Difficult to design.
2.	A clocked FF acts as a memory element.	An unclocked FF or time delay element is used as memory element.
3.	Slower, because the delays correspond to those of the memory element.	Faster as the clock is not present.
4.	The status of memory element is affected only at the active edge of clock if the input is changed.	The status of memory element will change any time as soon as the input is changed.

7.19 Analysis of Clocked Sequential Circuits :

- The behaviour of a clocked sequential circuit is dependent on the inputs, the outputs and the state of its flip-flops.
- The outputs as well as the next state both are function of inputs and present state.
- The analysis of the given clocked sequential circuit includes writing the state table and drawing the state diagram for the given circuit.
- In this section, we will introduce the concepts such as state diagram, state table, state equation and input equations and the step by step analysis procedure.

7.19.1 State Table :

MU : Dec. 02, Dec. 14, May 16

University Questions

Q. 1 Write short note on : State table (Dec. 02, 3 Marks, Dec. 14, 7 Marks, May 16, 5 Marks)

- We use the truth table for explaining the relation between its inputs and outputs of a combinational circuit.
- But it is not suitable to use the truth table to describe the input output relation of a sequential circuit.
- Instead we use the state table to describe the input output relation of the sequential circuits.
- The basic building block of a sequential circuit is a flip flop. The outputs Q_A, Q_B ... etc. of such flip flops will be used as **state inputs** to a sequential circuit. They are also called as **state variable.**
- In addition to this "x" represents an external input and Y represents the output of the sequential circuit.
- Y will be dependent on the state variables (Q_A, Q_B, ...) and the external input x.
- The general format of a state table is shown in Table 7.19.1.
- The state table tells us about the relation between the present state, next state, external inputs and output.

Table 7.19.1 : General format of state table

Present state	Next state		Output Y	
	x = 0	x = 1	x = 0	x = 1
Q_A Q_B	Q_A Q_B	Q_A Q_B		
0 0	0 0	0 1	0	0
0 1	1 1	0 1	0	0
1 0	1 0	0 0	0	1
1 1	1 0	1 1	1	0

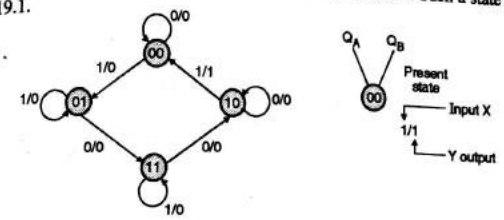
7.19.2 State Diagram :

MU : Dec. 02

University Questions

Q. 1 Write short note on : State diagrams. (Dec. 02, 2 Marks)

- The information available in the state table is represented graphically using the state diagram.
- The state diagram is drawn by using the state table as a reference. Such a state diagram is shown in Fig. 7.19.1.

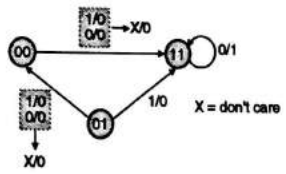


(C-653) **Fig. 7.19.1 : State diagram**

- The circle represents the present state. The arrows between the circles define the state transition say from 00 to 01 or 01 to 11.
- If there is directed line connecting the same circle then it means that the next state is same as the present state.
- The lines joining the circles are labeled with a pair of binary numbers with a " / " in between. For example the line joining 00 and 01 is labeled as 1/0.
- Note that 00 to 01 transition takes place when x = 1 and Y = 0 (see row-1 of the state table). Hence 1 in 1/0 corresponds to x and 0 corresponds to y.

Don't care condition in the state diagram :

- Sometimes the same next state is reached for more than one present states.
- This is called as don't care condition in the state diagram, as shown in Fig. 7.19.2.



(C-654) **Fig. 7.19.2 : Don't care condition in state diagram**

7.19.3 State Equation :

- State equation is an algebraic equation. The left side of this equation represents the next state of the flip flops.
- And the right hand side of this equation specifies the present state conditions which make the next state equal to 1.
- For example consider the following state expression.

$$Q_{A(n+1)} = (\bar{Q}_A Q_B + Q_A \bar{Q}_B + Q_A Q_B)x + Q_A \bar{Q}_B x \ldots\ldots$$

Next state

Combinations of present state and inputs which make the next state equal to 1.

- State equation is also called as application equation.

Scanned by CamScanner

- We can derive the state equation from the state table using the K maps. The state equation is the Boolean function with time included into it.
- If the right hand side part is zero and we apply a clock pulse, then the next state will be equal to zero.

Ex. 7.19.1 : For the clocked D FF write the state table, draw the state diagram and write the state equation.

Soln. :

Step 1 : Write the truth table :

Table P. 7.19.1(a) represents the truth table for a clocked D flip flop.

Table P. 7.19.1(a) : Truth table of a clocked D FF

Inputs		Outputs	
CLK	D	Q_{n+1}	$\bar{Q}_{n+1}$
0	X	Q_n	$\bar{Q}_n$
1	X	Q_n	$\bar{Q}_n$
↓	X	Q_n	$\bar{Q}_n$
↑	0	0	1
↑	1	1	0

No change in output

Q output follows the D input

Step 2 : Write the state table :

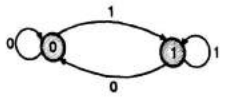
Table P. 7.19.1(b) represents the state table for the clocked D FF.

Table P. 7.19.1(b) : State table of a clocked D FF

Present state Q_n	Next state Q_{n+1}	
	D = 0	D = 1
0	0	1
1	0	1

Step 3 : State diagram :

State diagram of clocked D FF is shown in Fig. P. 7.19.1.



(C-455) **Fig. P. 7.19.1 : State diagram of a clocked D FF**

Step 4 : Write excitation table :

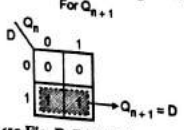
The excitation table for D FF is given below.

Table P. 7.19.1(c) : Excitation table for D FF

Present state Q_n	Next state Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Step 5 : Write the state equation :

The K-map for output Q_{n+1} is shown in Fig. P. 7.19.1(a).



∴ **State equation**

$Q_{n+1} = D$

(C-456) **Fig. P. 7.19.1(a) : K-map and state equation**

Ex. 7.19.2 : For the clocked JK FF write the state table, draw the state diagram and write the state equation.

Soln. :

Step 1 : Write the truth table :

Table P. 7.19.2(a) represents the truth table for a clocked JK flip flop.

Table P. 7.19.2(a) : Truth table of JK FF

Inputs			Output
CLK	J	K	Q_{n+1}
0	X	X	Q_n (NC)
1	X	X	Q_n (NC)
↓	X	X	Q_n (NC)
↑	0	0	Q_n (NC)
↑	0	1	0
↑	1	0	1
↑	1	1	$\bar{Q}_n$

No change in output

Step 2 : Write the state table :

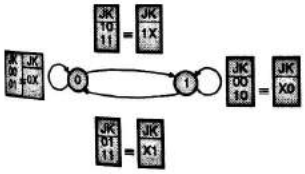
Table P. 7.19.2(b) represents the state table for a clocked JK FF.

Table P. 7.19.2(b) : State table of clocked JK FF

Present state Q_n	Next state Q_{n+1}			
	JK = 00	JK = 01	JK = 10	JK = 11
0	0	0	1	1
1	1	0	1	0

Step 3 : Draw the state diagram :

State diagram is as shown in Fig. P. 7.19.2(a).



(C-457) **Fig. P. 7.19.2(a) : State diagram of a JK flip flop**

Step 4 : Write the excitation table :

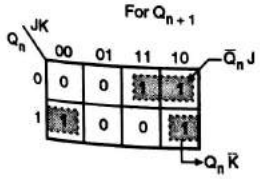
The excitation table of JK FF is as shown in Table P. 7.19.2(c).

Table P. 7.19.2(c) : Excitation table for JK FF

Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Step 5 : Write the state equation :

The K-map for output Q_{n+1} is shown in Fig. P. 7.19.2(b).



∴ **State equation**

$Q_{n+1} = \bar{Q}_n J + Q_n \bar{K}$ ...Ans.

(C-458) **Fig. P. 7.19.2(b) : K-map state equation**

Scanned by CamScanner

Ex. 7.19.3 : For a toggle FF write the state table, draw the state diagram and write the state equation.

Soln. :

Step 1 : Write the truth table :

Table P. 7.19.3(a) gives the truth table for a T FF.

Table P. 7.19.3 (a) : Truth table for a T FF

CLK	T	Q_{n+1}
0	X	Q_n
1	X	Q_n
↑	X	Q_n
↓	0	Q_n
↓	1	$\bar{Q}_n$

Step 2 : Write the state table :

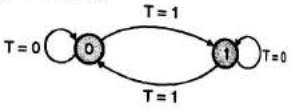
The state table is shown in Table P. 7.19.3(b).

Table P. 7.19.3(b) : State table for a T FF

Present state Q_n	Next state Q_{n+1}	
	T = 0	T = 1
0	0	1
1	1	0

Step 3 : Draw the state diagram :

The state diagram is shown in Fig. P. 7.19.3(a).



(C-659) **Fig. P. 7.19.3(a) : State diagram of a T FF**

Step 4 : Write the excitation table :

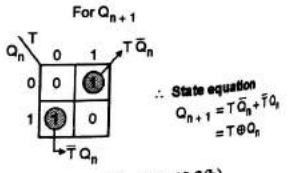
Table P. 7.19.3(c) represents the excitation table of T FF.

Table P. 7.19.3(c) : Excitation table for T FF

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Step 5 : Write the state equation :

The K-map for Q_{n+1} output is shown in Fig. P. 7.19.3(b). The state equation can be obtained from this K-map.



(C-660) **Fig. P. 7.19.3(b)**

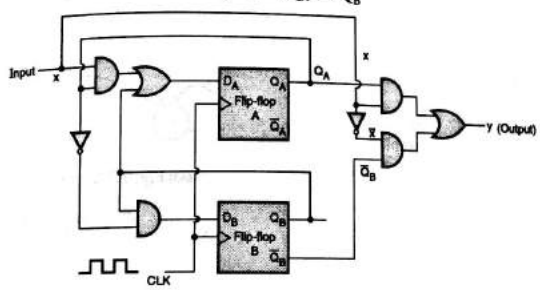
7.19.4 Flip Flop Input Equations :

- The logic diagram of a sequential circuit consists of flip flops and gates. Fig. 7.19.3 shows an example circuit.
- The interconnected gates form a combinational circuit and we can specify it by using the Boolean expressions.
- If the types of flip flops are known and all the Boolean expressions also are known to us, then it is possible for us to draw the logic diagram of a sequential circuit.

Output equations :

- We can express the part of combinational circuit that generates the external outputs by the Boolean expressions called as the output equations.
- The output equation for the sequential circuit of Fig. 7.19.3 is,

$$y = x Q_A + \bar{x} \bar{Q}_B \quad ...(7.19.1)$$



(C-661) **Fig. 7.19.3 : A sequential circuit**

Input equations :

- The part of the combinational circuit that generates the inputs to flip flops is expressed by Boolean equations called as flip flop input equations.
- The input equations for the sequential circuit of Fig. 7.19.3 are,

$$D_A = x \cdot Q_A + Q_B \quad ...(7.19.2)$$

and

$$D_B = \bar{Q}_A \cdot Q_B \quad ...(7.19.3)$$

- The input equations are also called as **excitation equations.**
- With the input and output equations we can draw the logic diagram of a sequential circuit.

7.20 Analysis Procedure for Circuits Containing Flip-Flops :

The procedure used for analyzing a clocked sequential circuit is as follows :

1. Write the input and output equations from the given circuit.
2. Obtain the state table.
3. Draw the state diagram.

7.20.1 Analysis with D Flip Flops :

- We will demonstrate the procedure for analysing a clocked sequential circuit with D-flip flop by using a simple circuit.
- This circuit can be described by the following input equation.

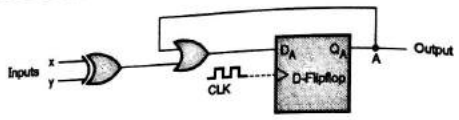
$$D_A = Q_A + (x \oplus y) \quad ...(7.20.1)$$

where x and y are the inputs and Q_A is the output. No output equation can be written as the output is coming directly from the flip flop output.

Step 1 : Draw the logic diagram :

From the given input equation, we can draw the logic diagram as shown in Fig. 7.20.1.

Scanned by CamScanner



(C-662) **Fig. 7.20.1 : Logic diagram of sequential circuit using D-FF**

Step 2 : Obtain the state table :

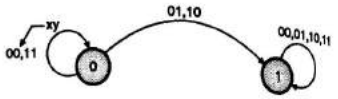
The state table for the sequential circuit of Fig. 7.20.1 is as shown in Table 7.20.1. The next state of a D flip flop is given by,

$$A_{n+1} = D_A = A + (x \oplus y)$$

Table 7.20.1 : State table

Present state	Inputs		Next state
A	x	y	A_{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

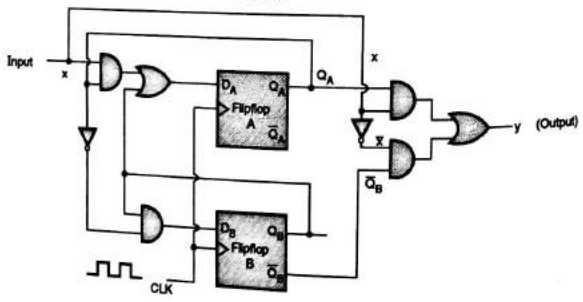
Step 3 : Draw the state diagram :



(C-663) **Fig. 7.20.2 : State diagram**

The state diagram drawn from the state table is shown in Fig. 7.20.2.

Ex. 7.20.1 : Analyze the sequential circuit of Fig. P. 7.20.1(a). Write the input, output equations, obtain the state table and draw the state diagram.



(C-664) **Fig. P. 7.20.1(a) : Given circuit**

Soln. :

Step 1 : Write the input and output equations :

The input equations for the sequential circuit of Fig. P. 7.20.1(a) are,

$$D_A = x \cdot Q_A + Q_B \quad ...(1)$$

$$\text{and} \quad D_B = \bar{Q}_A \cdot Q_B \quad ...(2)$$

- The output equation for the sequential circuit of Fig. P. 7.20.1(a) is,

$$y = x Q_A + \bar{x} \bar{Q}_B \quad ...(3)$$

Step 2 : Obtain the state table :

- The state table is as shown in Table P. 7.20.1. It consists of four sections :
  1. Present state 2. Inputs
  3. Next state 4. Outputs
- The present state and inputs are obtained by listing all the binary combinations.
- The output is obtained from the output equation.
- The next state is obtained from the **state equation.** For a D flip flop the state equation is same as input equation.

$$\therefore \quad A_{n+1} = x \cdot A + B \quad ...(4)$$

$$\text{and} \quad B_{n+1} = \bar{A} \cdot B \quad ...(5)$$

Note that we have written A for Q_A and B for Q_B for simplifying the expressions.

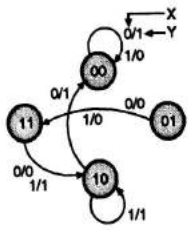
- The state table is shown in Table P. 7.20.1.

Table P. 7.20.1 : State table

Present state		Input	Next state		Output
A	B	X	A_{n+1}	B_{n+1}	$y = xA + \bar{x}\bar{B}$
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	1	0	0
1	1	1	1	0	1

Step 3 : State diagram :

- There are four possible states 00, 01, 10 and 11. The state diagram derived from the state table is as shown in Fig. P. 7.20.1(b).



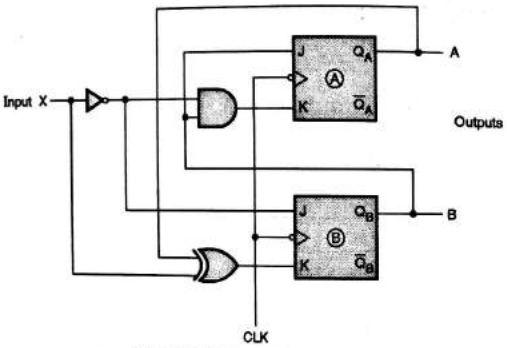
(C-665) **Fig. P. 7.20.1(b) : State diagram**

Scanned by CamScanner

7.20.2 Analysis Using JK Flip-Flops :

- In case of a D flip flop, the state equation which is essential for obtaining the next state value in the state table is same as the input equation. This is because in case of a D flip flop the output follows the input.
- But in case of other flip flops, it is not the case. For the JK or T flip flops, we have to refer to their characteristic tables (truth tables) or characteristic equation to obtain the next state value.
- Let us demonstrate this first using the characteristic table and then by using the characteristic equation. Refer to the following two examples for the same.

Ex. 7.20.2 : Analyze the clocked sequential circuit of Fig. P. 7.20.2(a). Write the state table by using the characteristic table of JK flip flop. Also draw the state diagram.



(C-666) **Fig. P. 7.20.2(a) : Given circuit**

Soln. :

Step 1 : Write the input and output equations :

- The input equations are :

$$J_A = B, \quad K_A = \overline{X}B \qquad ...(1)$$

$$J_B = \overline{X}, \quad K_B = X \oplus A = X\overline{A} + \overline{X}A \qquad ...(2)$$

- The outputs are obtained directly from the flip flop outputs. As no combinational circuit is involved in producing the outputs, there are no output equations.

Step 2 : Obtain the state table :

We can obtain the next state values of a sequential circuit that uses JK or T flip flops using the following procedure.

1. Obtain the input equations of flip flops.
2. List the binary values of each input equation.
3. Use the characteristic (truth) table of the flip flop to determine the next state value in the state table.

- Using this procedure, we can obtain the state table of Table P. 7.20.2.

- The present state AB and input X columns list all the possible eight binary combinations 000 to 111.
- From these values and the input equations, we obtain the values of flip flop inputs J_A, K_A, J_B and K_B. Note that the columns labled flip flop inputs is not a part of the state table but it is essential for finding the next state.
- Now from the values of FF inputs, using the JK FF truth table, find the next state values to complete the state table.

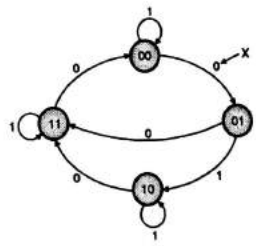
Table P. 7.20.2 : State table

Present state		Input	FF inputs				Next state	
A	B	X	$J_A = B$	$K_A = B\overline{X}$	$J_B = \overline{X}$	$K_B = A \oplus X$	A_{n+1}	B_{n+1}
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0
0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	0
1	0	0	0	0	1	1	1	1
1	0	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	0	1	1

- For example, consider the first row (shaded) in Table P. 7.20.2. J_A K_A = 00 and A = 0. As per truth table of JK FF there is no change in output if JK = 00. Hence the next state A_{n+1} = A = 0. In the same row J_B K_B = 10 and B = 0. Now as per truth table of JK FF, the FF is set if JK = 10. Hence B_{n+1} = 1. All the other entries in the next state column have been entered in a similar way.

Step 3 : Draw the state diagram :

- The state diagram derived from the state table is as shown in Fig. P. 7.20.2(b).



(C-667) **Fig. P. 7.20.2(b) : State diagram**

Ex. 7.20.3 : For the same circuit of previous example, obtain the state table by using the characteristic equations.

Soln. :

- The next state values in the state table can be obtained by using the characteristic equations of JK FF by following the procedure given below.

1. Obtain the input equations of the flip flops.
2. Substitute the input equations into the characteristic equations of FF and get the state equations.
3. Use these state equations to determine the next state values in the state table.

Scanned by CamScanner

- This procedure can be demonstrated as follows.

Step 1 : Input equations :

As obtained in the previous example, the input equations are :

$$J_A = B, \quad K_A = B\bar{X} \qquad ...(1)$$

and

$$J_B = \bar{X}, \quad K_B = A \oplus X \qquad ...(2)$$

Step 2 : Obtain the state equations :

- We have obtained the characteristic equation of a JK FF earlier as :

$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n \qquad ...(3)$$

- Slightly modify this equation to get the characteristic equations of the two flip flops as follows :

$$A_{n+1} = J_A\bar{A} + \bar{K}_A A \qquad ...(4)$$

and

$$B_{n+1} = J_B\bar{B} + \bar{K}_B B \qquad ...(5)$$

- Now substitute the input equations i.e. Equations (1) and (2) into Equations (4) and (5) to get the state equations as follows :

$$A_{n+1} = B\bar{A} + \left(\overline{B\,\overline{X}}\right)A = \bar{A}B + (\bar{B} + X)A$$

$$\therefore \quad A_{n+1} = \bar{A}B + A\bar{B} + AX = (A \oplus B) + AX \qquad ...(6)$$

and

$$B_{n+1} = \bar{X}\bar{B} + (A \oplus X)B = \bar{X}\bar{B} + (\bar{A}X + A\bar{X})B$$

$$\therefore \quad B_{n+1} = \bar{X}\bar{B} + \bar{A}BX + AB\bar{X} \qquad ...(7)$$

- Equations (6) and (7) are the required state equations. We can use them to derive the state table.

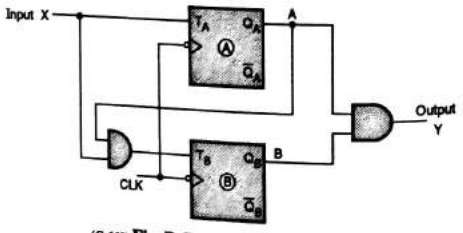
7.20.3 Analysis using T Flip Flops :

- The procedure used for the analysis of a circuit using T-Flip flop is same as that used for the analysis of a circuit using JK FF.
- The characteristic equation of a T FF is as follows.

$$Q_{n+1} = T \oplus Q_n = T\bar{Q}_n + \bar{T}Q_n \qquad ...(7.20.2)$$

- The analysis procedure is demonstrated in the following example.

Ex. 7.20.4 : Analyze the circuit of Fig. P. 7.20.4(a). Obtain the state table and draw the state diagram.



(C-668) **Fig. P. 7.20.4(a) : Given circuit**

Soln. :

Step 1 : Write the input equations and output equation :

Input equations :

$$T_A = X \qquad ...(1)$$

$$T_B = A \cdot X \qquad ...(2)$$

Output equation :

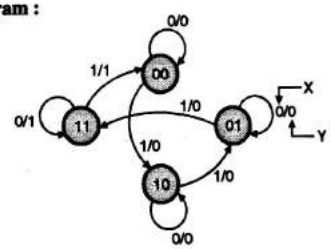
$$Y = A \cdot B \qquad ...(3)$$

Step 2 : Write the state table :

Table P. 7.20.4 : State table

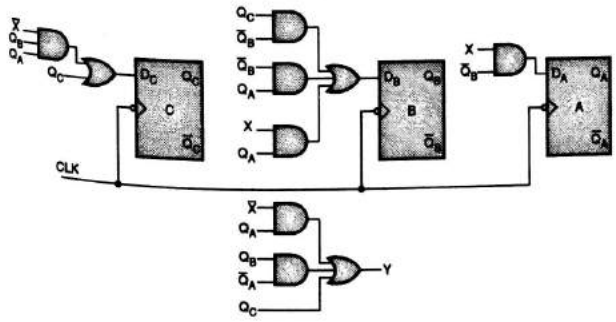
Present state		Input	Flip flop inputs		Next state		Output
A	B	X	$T_A = X$	$T_B = AX$	A_{n+1}	B_{n+1}	$Y = AB$
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0
1	0	1	1	1	0	1	0
1	1	0	0	0	1	1	1
1	1	1	1	1	0	0	1

Step 3 : Draw the state diagram :



(C-669) **Fig. P. 7.20.4(b) : State diagram**

Step 5 : Realization : The realization is shown in Fig. P. 7.20.4(b).



(C-684) **Fig. P. 7.20.4(c)**

Scanned by CamScanner

7.20.4 Models for Synchronous Sequential Circuits :

- There are two models developed for representing the synchronous sequential circuits namely :
  1. Moore circuit 2. Mealy circuit
- In the synchronous sequential circuits, the combinational circuit block consists of a combination of logic gates where as the flip flops are used as memory elements.
- A synchronous sequential circuit is called as a **Moore circuit** if its output depends only on the present state of flip flops. The output does not depend on the external inputs.
- A synchronous sequential circuit is called as a **Mealy circuit** if its output is dependent on the present state of flip flops as well as the external inputs.

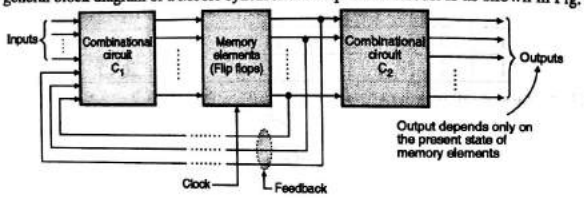
7.21 Moore Circuit :

MU : May 04

University Questions

Q. 1 Write short note on : Moore machine. **(May 04, 5 Marks)**

- The general block diagram of a Moore synchronous sequential circuit is as shown in Fig. 7.21.1.



(C-670) **Fig. 7.21.1 : A general form of Moore circuit**

- From this block diagram it is illustrated that for a Moore circuit the output depends only on the present state of the memory elements i.e. flip flops. The external inputs do not influence the outputs.
- The combinational circuit C_1 on the input side is sometimes called as the next state decoder. The external inputs and the feedback signals from the memory elements are applied to this combinational circuit.
- The outputs of the memory elements (present state) are applied to another combinational circuit C_2 which is also called as output decoder. The outputs are produced at the output of this block.
- Note that the clock signal is applied only to the memory element block which consists of the flip flops.

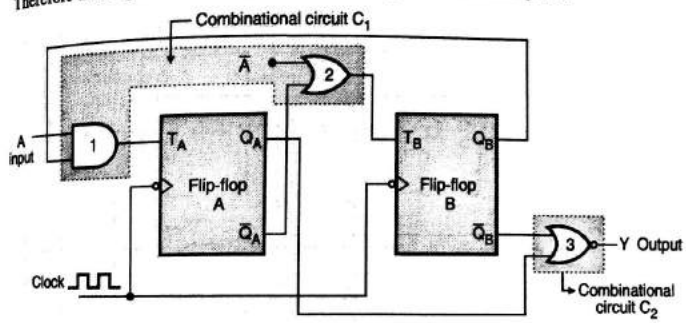
7.21.1 Example of Moore Circuit :

- The example of a Moore circuit is shown in Fig. 7.21.2.
- Flip flops A and B act as the memory elements for the circuit.
- Gates 1 and 2 form the input combinational circuit C_1, because the external inputs A and $\bar{A}$ are applied to one of their inputs, whereas the other inputs are coming from the Q_B and $\bar{Q}_A$ outputs of the memory elements.

- Gate-3 acts as the output decoder or combinational circuit C_2. The inputs to this gate are from the outputs of the two memory elements FF-A and FF-B. Note that the external inputs are not connected to the output decoder block.

$$\therefore \; Y = \overline{Q_A + \bar{Q}_B}$$

- Thus the output Y depends only on the present state of flip flops.
- Both the flip flops are negative edge triggered and the clock input is applied to both the flip flops simultaneously.
- Therefore the output Y will change only after the application of clock pulse.



(C-671) **Fig. 7.21.2 : Example of Moore circuit**

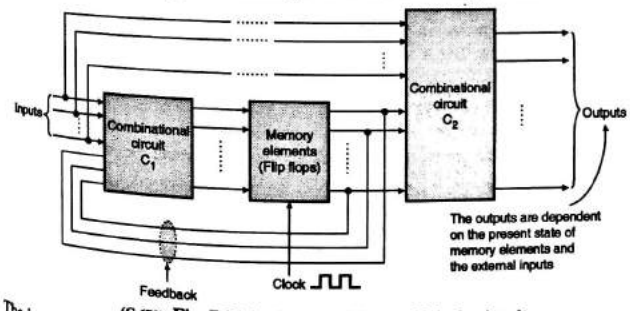
7.22 Mealy Circuit :

MU : May 04

University Questions

Q. 1 Write short note on : Mealy machine. **(May 04, 5 Marks)**

- The generalised block diagram of a Mealy circuit is shown in Fig. 7.22.1.



(C-673) **Fig. 7.22.1 : A general form of Mealy circuit**

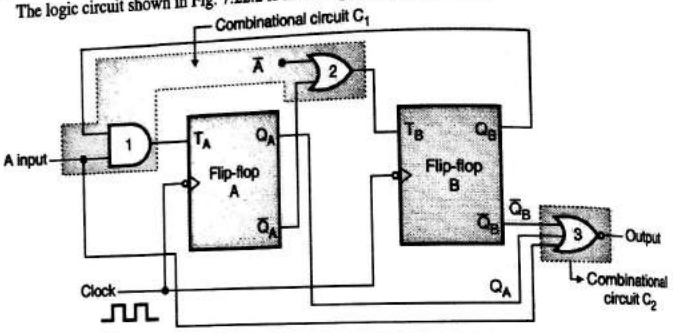
- The input combinational circuit (C_1) which is also called as the next state decoder receives inputs from the outputs of memory elements (present state) and from the external inputs.

Scanned by CamScanner

- But the most important point to be noted is that the output combinational circuit C_2 (also called as output decoder) receives signal from the output of memory elements as well as the external inputs.
- This shows that the outputs of Mealy circuit is dependent on the present state of memory elements as well as the external inputs.

7.22.1 Example of Mealy Circuit :

- The logic circuit shown in Fig. 7.22.2 is an example of Mealy circuit.



(C-674) **Fig. 7.22.2 : Example of Mealy circuit**

- As usual, the flip flops A and B form the memory element. Both the FFs are negative edge triggered and the clock signal has been applied to both the flip flops simultaneously.
- Gates 1 and 2 form the input combinational circuit C_1 and gate-3 forms the output combinational circuit i.e. C_2.
- Note that the inputs to gate-3 are Q_A, $\bar{Q}_B$ and A. That means the output is now dependent on the outputs of memory elements (present state) as well as the external input A.

$$Y = Q_A + \bar{Q}_B + A$$

- However any changes in "A" in between the successive negative clock edges will not affect the outputs of the flip flops. But it will definitely affect the final output, because it will affect the combinational circuit C_2.
- Thus a false output may be produced.
- To eliminate this problem, the external inputs should be allowed to change their state only in synchronization with the active edge of clock and not in between the active clock edges of the successive clock pulses.

7.22.2 Comparison of Moore and Mealy Models :

Table 7.22.1 : Comparison of Moore and Mealy models

Sr. No.	Moore model	Mealy model
1.	The final output depends only on the present state of memory elements.	The final output depends on the present state of memory elements and the external inputs.

Sr. No.	Moore model	Mealy model
2.	The output changes only after the active clock edge.	Output can change in between the clock edges if the external inputs change.
3.	The implementation of a logic function needs more number of states than Mealy circuit.	Implementation of the same logic function requires less number of states than Moore circuit.

7.23 Analysis of Clocked Sequential Circuits :

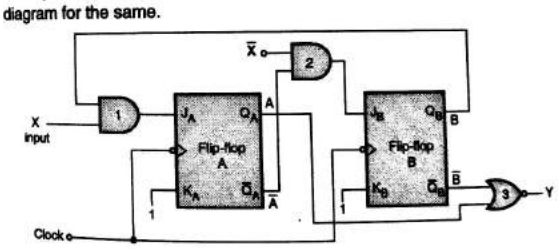
- The sequential circuit to be analysed is given to us.
- The meaning of analysis is that we have to identify the type of circuit, write its state table and draw the state diagram.
- This procedure will be clear after solving the following examples.

Analysis procedure :

1. Identify the type of synchronous sequential circuit (Moore or Mealy).
2. Derive the state table.
3. Draw the state diagram.

This procedure will be clear after solving the following example.

Ex. 7.23.1 : Identify the type of circuit shown in Fig. P. 7.23.1(a), write the state table and draw the state diagram for the same.



(C-675) **Fig. P. 7.23.1(a) : Given circuit**

Soln. :

Step 1 : Type of circuit :

Since output $Y = \overline{A + \bar{B}}$, it is dependent only on the present state of the memory elements. Hence this is a Moore circuit.

Step 2 : Write the state table :

The state table is as shown in Table P. 7.23.1(a).

Table P. 7.23.1(a) : State table

Present state		Next state		Output
A	B	X = 0	X = 1	Y

Scanned by CamScanner

Description :

- The present state column in the state table represents the FF outputs before application of clock pulse.
- The next state column represents the outputs of FF outputs after the application of clock pulse. It is dependent on the present state as well as the external input X.
- Output Y column values are obtained from the present state column of the state table.

Derivation of state table :

The derivation of state table is as follows. Let us consider each present state and obtain the corresponding next state and Y output.

Present state AB = 00 :

Next state with X = 0	Next state with X = 1
• $J_A = X \cdot B = 0 \cdot 0 = 0$ and $K_A = 1$	• $J_A = X \cdot B = 1 \cdot 0 = 0$ and $K_A = 1$
• Hence $A_{n+1} = 0$	• Hence $A_{n+1} = 0$
• $J_B = \bar{X} \cdot \bar{A} = 1 \cdot 1 = 1$ and $K_B = 1$	• $J_B = \bar{X} \cdot \bar{A} = 0 \cdot 1 = 0$ and $K_B = 1$
So FF-B will toggle	
• Hence $B_{n+1} = 1$	• Hence $B_{n+1} = 0$
• Hence next state = 01	• Hence next state = 00

Output $Y = \overline{A + \bar{B}} = \overline{0 + 1}$

$\therefore Y = 0$

Present state AB = 01 :

Next state with X = 0	Next state with X = 1
• $J_A = X \cdot B = 0 \cdot 1 = 0$ and $K_A = 1$	• $J_A = X \cdot B = 1 \cdot 1 = 1$ and $K_A = 1$
• Hence $A_{n+1} = 0$	• Hence FF-A toggles
• $J_B = \bar{X} \cdot \bar{A} = 1 \cdot 1 = 1$ and $K_B = 1$	• $\therefore A_{n+1} = 1$
• Hence $B_{n+1} = 0$ as FF-B toggles	• $J_B = \bar{X} \cdot \bar{A} = 0 \cdot 1 = 0$ and $K_B = 1$
• Hence next state = 00	• Hence $B_{n+1} = 0$
	• Hence next state = 10

Output $Y = \overline{A + \bar{B}} = \overline{0 + 0} = 1$

Present state AB = 10 :

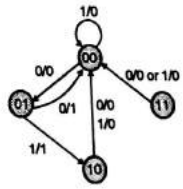
Next state with X = 0	Next state with X = 1
• $J_A = X \cdot B = 0 \cdot 0 = 0$ and $K_A = 1$	• $J_A = X \cdot B = 1 \cdot 0 = 0$ and $K_A = 1$
• Hence $A_{n+1} = 0$	• Hence $A_{n+1} = 0$
• $J_B = \bar{X} \cdot \bar{A} = 1 \cdot 0 = 0$ and $K_B = 1$	• $J_B = \bar{X} \cdot \bar{A} = 0 \cdot 0 = 0$ and $K_B = 1$
• Hence $B_{n+1} = 0$	• Hence $B_{n+1} = 0$
• Therefore next state = 00	• Therefore the next state is 00

Output $Y = \overline{A + \bar{B}} = \overline{1 + 1} = 0$

- Similarly we can obtain the next state and output for the present state of 1 1 as follows :
  Present state : AB = 11
  Next state (X = 0) : 00 next state (X = 1) = 00
  Output Y = 0
- The complete state table is given in Table P. 7.23.1(b).

Table P. 7.23.1(b) : State table

Present state	Next state $A_{n+1} B_{n+1}$		Output Y
A B	X = 0	X = 1	
0 0	0 1	0 0	0
0 1	0 0	1 0	1
1 0	0 0	0 0	0
1 1	0 0	0 0	0



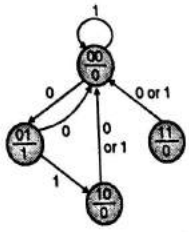
(C-676) **Fig. P. 7.23.1(b) : State diagram**

Step 3 : Draw the state diagram :

The state diagram is drawn with the help of state table. The state diagram is shown in Fig. P. 7.23.1(b).

Step 4 : Modified state diagram :

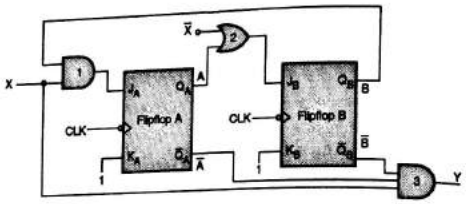
The modified state diagram for Moore circuit is shown in Fig. P. 7.23.1(c). Note that the binary number corresponding to Y output has been included inside the circle itself.



(C-677) **Fig. P. 7.23.1(c) : Modified state diagram for the given Moore circuit**

Ex. 7.23.2 : Identify the circuit shown in Fig. P. 7.23.2(a). Write the state table and draw state diagram for the same.

Scanned by CamScanner



(C-678) **Fig. P. 7.23.2(a) : Given circuit**

Soln. :

Step 1 : Identify the type of circuit :

The given circuit is a Mealy circuit since the Y output depends on the present state as well as the X input.

Step 2 : Write the state table :

- The state table for given circuit is given in Table P. 7.23.2.
- Note that the output Y contains two columns one each for X = 0 and X = 1. This is because in Mealy circuit the output depends on external input as well as the present state.

$$Y = \bar{A} \cdot \bar{B} \cdot X \quad ...(1)$$

Where $\bar{A}\,\bar{B}$ represents the present state of flip-flops.

Table P. 7.23.2 : State table

Present state	Next state $A_{n+1} B_{n+1}$		Output $Y = \bar{A} \cdot \bar{B} \cdot X$	
A B	X = 0	X = 1	X = 0	X = 1
0 0	0 1	0 0	0	1
0 1	0 0	1 0	0	0
1 0	0 1	0 1	0	0
1 1	0 0	0 0	0	0

$J_A = X \cdot B \qquad J_B = \bar{X} + A$

Derivation of state table :

- Let me demonstrate the procedure to obtain the next state and output corresponding to first two present states AB = 00 and 01. You can follow the same procedure for the remaining states.
- **Present state AB = 00 :**

Next state for X = 0	Next state for X = 1
• $J_A = X \cdot B = 0 \cdot 0 = 0$ and $K_A = 1$	• $J_A = X \cdot B = 1 \cdot 0 = 0$ and $K_A = 1$
• Hence $A_{n+1} = 0$	• Hence $A_{n+1} = 0$
• $J_B = \bar{X} + A = 1 + 0 = 1$ and $K_B = 1$ Hence FF-B will toggle	• $J_B = \bar{X} + A = 0 + 0 = 0$ and $K_B = 1$
• Therefore $B_{n+1} = 1$	• Hence $B_{n+1} = 0$
• Hence next state = 01	• Therefore the next state = 00

Output Y

- For X = 0, $Y = \bar{A}\,\bar{B}\,X = 1 \cdot 1 \cdot 0 = 0$
- For X = 1, $Y = \bar{A}\,\bar{B}\,X = 1 \cdot 1 \cdot 1 = 1$

- **Present state AB = 0 1 :**

Next state with X = 0	Next state for X = 1
• $J_A = X \cdot B = 0 \cdot 1 = 0$ and $K_A = 1$	• $J_A = X \cdot B = 1 \cdot 1 = 1$ and $K_A = 1$ So FF-A will toggle.
• Hence $A_{n+1} = 0$	• Hence $A_{n+1} = 1$
• $J_B = \bar{X} + A = 1 + 0 = 1$ and $K_B = 1$ So FF-B will toggle.	• $J_B = \bar{X} + A = 0 + 0 = 0$ and $K_B = 1$
• Hence $B_{n+1} = 0$	• Hence $B_{n+1} = 0$
• Therefore next state = 00	• Therefore next state = 10

Output Y

- For X = 0, $Y = \bar{A}\,\bar{B}\,X = 1 \cdot 0 \cdot 0 = 0$
- For X = 1, $Y = \bar{A}\,\bar{B}\,X = 1 \cdot 0 \cdot 1 = 0$

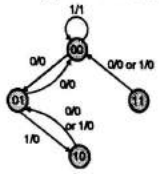
- Similarly we can obtain the next state and output Y for the remaining two present states to complete the state table.

Step 3 : Draw the state diagram :

The state diagram for the given Mealy circuit is shown in Fig. P. 7.23.2(c). The state table has been repeated in Fig. P. 7.23.2(b) for reference.

Present state	Next state $A_{n+1} B_{n+1}$		Output Y	
A B	X = 0	X = 1	X = 0	X = 1
0 0	0 1	0 0	0	1
0 1	0 0	1 0	0	0
1 0	0 1	0 1	0	0
1 1	0 0	0 0	0	0

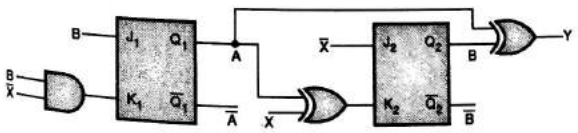
(b) State table repeated



(C-679) **(c) State diagram**

Fig. P. 7.23.2

Ex. 7.23.3 : Analyze the circuit and hence draw state diagram. State whether it is a Mealy machine or a Moore machine.



(C-3035) **Fig. P. 7.23.3(a)**

Soln. :

Step 1 : Type of circuit :

Since output $Y = A \oplus B$, it is dependent only on the present state of the memory elements. Hence this is a Moore circuit.

Scanned by CamScanner

Step 2 : Write the state table :

Present state		Next state		Output Y
A	B	A_{n+1} X = 0	B_{n+1} X = 1	
0	0	01	00	0
0	1	01	10	1
1	0	00	00	1
1	1	00	10	0

Present state AB = 00

Next state with X = 0	Next state with X = 1
• $J_1 = B = 0$ and $K_1 = B\bar{X} = 0$ Hence $A_{n+1} = 0$	• $J_1 = B = 0$ and $K_1 = B\bar{X} = 0$ Hence $A_{n+1} = 0$
• $J_2 = \bar{X} = 1$ and $K_2 = A \oplus X = 0$ Hence $B_{n+1} = 1$	• $J_2 = \bar{X} = 0$ and $K_2 = A \oplus X = 0$ Hence $B_{n+1} = 0$
• Hence next state = 01	• Hence next state = 00
Output $Y = A \oplus B = 0$	

Present state AB = 01

Next state with X = 0	Next state with X = 1
• $J_1 = B = 0$ and $K_1 = B\bar{X} = 1 \cdot 1 = 1$ Hence $A_{n+1} = 0$	• $J_1 = B = 1$ and $K_1 = B\bar{X} = 1 \cdot 0 = 0$ Hence $A_{n+1} = 1$
• $J_2 = \bar{X} = 1$ and $K_2 = A \oplus X = 0 \oplus 0 = 0$ Hence $B_{n+1} = 1$	• $J_2 = \bar{X} = 0$ and $K_2 = A \oplus X = 0 \oplus 1 = 1$ Hence $B_{n+1} = 0$
• Hence next state = 01	• Hence next state = 10
Output $Y = A \oplus B = 0 \oplus 1 = 1$	

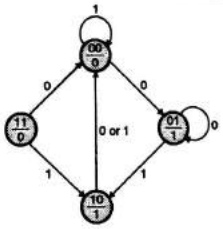
Present state AB = 10

Next state with X = 0	Next state with X = 1
• $J_1 = B = 0$ and $K_1 = B\bar{X} = 0 \cdot 1 = 0$ Hence $A_{n+1} = 0$	• $J_1 = B = 0$ and $K_1 = B\bar{X} = 0 \cdot 0 = 0$ Hence $A_{n+1} = 0$
• $J_2 = \bar{X} = 1$ and $K_2 = A \oplus X = 1 \oplus 0 = 1$ Hence $B_{n+1} = 0$	• $J_2 = \bar{X} = 0$ and $K_2 = A \oplus X = 1 \oplus 1 = 0$ Hence $B_{n+1} = 0$
• Hence next state = 00	• Hence next state = 00
Output $Y = A \oplus B = 1 \oplus 0 = 1$	

Present state AB = 11

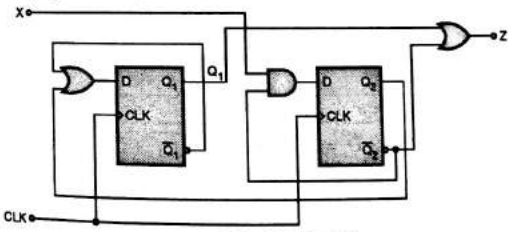
Next state with X = 0	Next state with X = 1
• $J_1 = B = 1$ and $K_1 = B\bar{X} = 1 \cdot 1 = 1$ Hence $A_{n+1} = 0$	• $J_1 = B = 1$ and $K_1 = B\bar{X} = 1 \cdot 0 = 0$ Hence $A_{n+1} = 1$
• $J_2 = \bar{X} = 1$ and $K_2 = A \oplus X = 1 \oplus 0 = 1$ Hence $B_{n+1} = 0$	• $J_2 = \bar{X} = 0$ and $K_2 = A \oplus X = 1 \oplus 1 = 0$ Hence $B_{n+1} = 0$
• Hence next state = 00	• Hence next state = 10
Output $Y = A \oplus B = 1 \oplus 1 = 0$	

Step 3 : Draw the state diagram :



(C-3036) **Fig. P. 7.23.3(b)**

Ex. 7.23.4 : Analyze the clocked synchronous state machine shown in Fig. P. 7.23.4(a). Write excitation equation, excitation/transition table, and state/output table (use state names A-D for $Q_1 Q_2$ = 00 – 11).



(C-3430) **Fig. P. 7.23.4(a)**

Soln. :

Step 1 : **Type of circuit :** Since output $Z = Q_1 + \overline{Q_2}$, it is dependent on the present state of memory elements. Hence Moore circuits.

Step 2 : **Write the state table :**

From the analysis of state machine, write the excitation equations

$$D_1 = \overline{Q_1} + Q_2 \qquad D_2 = X\,\overline{Q_2} \qquad \text{And} \quad Z = Q_1 + \overline{Q_2}$$

Scanned by CamScanner

State table :

Present state	Next state Q_{1+1} Q_{2+1}		Output
$Q_1 Q_2$	X = 0	X = 1	Z
00	10	11	1
01	10	10	0
10	00	01	1
11	10	10	1

Description :

Present state A i.e. $Q_1 Q_2 = 00$

Next state with X = 0	Next state with X = 1
• $D_1 = \overline{Q_1} + Q_2 = 1 + 0 = 1$, Hence $Q_1 = 1$	• $D_1 = \overline{Q_1} + Q_2 = 1$, Hence $Q_1 = 1$
• $D_2 = X\,\overline{Q_2} = 0 \cdot 1 = 0$, Hence $Q_2 = 0$	• $D_2 = X\,\overline{Q_2} = 1 \cdot 1 = 1$, Hence $Q_2 = 1$
• Hence next state = 10	• Hence next state = 11
Output $Z = Q_1 + \overline{Q_2} = 1$	

Present state B i.e. $Q_1 Q_2 = 01$

Next state with X = 0	Next state with X = 1
• $D_1 = \overline{Q_1} + Q_2 = 1 + 1 = 1$, Hence $Q_1 = 1$	• $D_1 = \overline{Q_1} + Q_2 = 1 + 1 = 1$, Hence $Q_1 = 1$
• $D_2 = X\,\overline{Q_2} = 0 \cdot 0 = 0$, Hence $Q_2 = 0$	• $D_2 = X\,\overline{Q_2} = 1 \cdot 0 = 0$, Hence $Q_2 = 0$
• Hence next state = 10	• Hence next state = 10
Output $Z = Q_1 + \overline{Q_2} = 0 + 0 = 0$	

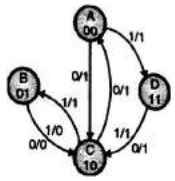
Present state C i.e. $Q_1 Q_2 = 10$

Next state with X = 0	Next state with X = 1
• $D_1 = \overline{Q_1} + Q_2 = 0 + 0 = 0$, Hence $Q_1 = 0$	• $D_1 = \overline{Q_1} + Q_2 = 0 + 0 = 0$, Hence $Q_1 = 0$
• $D_2 = X\,\overline{Q_2} = 0 \cdot 1 = 0$, Hence $Q_2 = 0$	• $D_2 = X\,\overline{Q_2} = 1 \cdot 1 = 1$, Hence $Q_2 = 1$
• Hence next state = 00	• Hence next state = 01
Output $Z = Q_1 + \overline{Q_2} = 1 + 1 = 1$	

Present state D i.e. $Q_1 Q_2 = 11$

Next state with X = 0	Next state with X = 1
• $D_1 = \overline{Q_1} + Q_2 = 0 + 1 = 1$, Hence $Q_1 = 1$	• $D_1 = \overline{Q_1} + Q_2 = 0 + 1 = 1$, Hence $Q_1 = 1$
• $D_2 = X\,\overline{Q_2} = 0 \cdot 0 = 0$, Hence $Q_2 = 0$	• $D_2 = X\,\overline{Q_2} = 1 \cdot 0 = 0$, Hence $Q_2 = 0$
• Hence next state = 10	• Hence next state = 10
Output $Z = Q_1 + \overline{Q_2} = 1 + 0 = 1$	

Step 3 : **Draw the state diagram :**



(C-3431) **Fig. P. 7.23.4(b)**

Review Questions

Q.1 Mention the types of digital systems. Explain with block diagram their operating principle.
Q.2 What is a flip-flop ?
Q.3 State and explain the triggering methods used for flip-flops.
Q.4 What is the function of preset and clear inputs in flip-flop ?
Q.5 Explain with truth table the working of clocked RS flip-flop.
Q.6 State the disadvantages of RS flip-flop. How can they be avoided ?
Q.7 Explain with diagram the working of D type flip-flop. Give its truth table.
Q.8 Give reason why D flip-flop is called as data latch ?
Q.9 Draw the circuit using logic gates of a T-type flip-flop. Draw its symbol and write its truth table.
Q.10 Explain S-R flip flop using NOR gates.
Q.11 Describe how two cross-coupled NAND gates form a R-S flip-flop ? Write its truth table.
Q.12 What are the various types of flip-flops ?
Q.13 Draw the circuit of SR flip-flop using NAND gate.
Q.14 Draw the schematic diagram of JK flip-flop and describe its working. Write down its truth table.
Q.15 What is race around condition ?
Q.16 Draw the circuit of J-K flip-flop using NAND gate.
Q.17 Draw a neat diagram of master slave J-K flip-flop. Explain how race around condition is avoided using master slave J-K flip-flop ?
Q.18 Explain the working of the master slave JK flip-flop.
Q.19 Can a flip flop be used as a memory ? If so how many bits can be stored by R-S flip flop ?
Q.20 Explain T flip-flop.
Q.21 Explain the following flip-flops :
1. Clocked SR
2. JK with preset and clear
3. Master Slave JK
4. D type and T type
Q.22 Design a conversion logic to convert a JK flip-flop to a D flip-flop.
Q.23 Write a short note on race around condition in JK flip-flop.
Q.24 Draw neat circuit diagram of clocked JK flip-flop using NAND gates. Give its truth table and explain race around condition.
Q.25 What is race around condition ? How does it gets eliminated in master slave JK FF ? Explain.
Q.26 Explain how JK FF is converted into :
1. D FF
2. T FF

□□□

Scanned by CamScanner

CHAPTER 8

Module 4

Counters

Syllabus :

Counters : Design of Synchronous and Asynchronous counters, Modulus of counters, Up-Down counters, Sequence generator.

8.1 Introduction :

- The digital circuit used for counting pulses is known as counter. It is a sequential circuit.
- Counters is the most widely used application of flip-flops. It is a group of flip-flops with a clock signal applied.
- Counters count the number of clock pulses. Therefore with some modifications we can use them for measuring frequency or time period.

8.1.1 Types of Counters :

- Counters are basically of two types :
  1. Asynchronous or ripple counters. 2. Synchronous counters.
- **Asynchronous or ripple counters :** For these counters the external clock signal is applied to one flip flop and then the output of preceding flip-flop is connected to the clock of next flip-flop.
- **Synchronous counters :** In synchronous counters all the flip-flops receive the external clock pulse is applied to all the flip-flops simultaneously. Ring counter and Johnson counter are the examples of synchronous counters.

8.1.2 Classification of Counters :

- Depending on the way in which the counter outputs change, the synchronous or asynchronous counters are classified as follows :

Counters (Synchronous/Asynchronous)

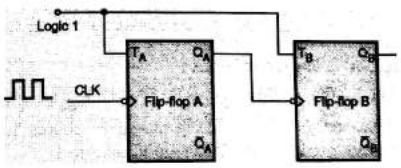
Up counters | Down counters | Up/Down counters

- **Up counters** are the counters that count from small to big count. Their output goes on increasing as they receive clock pulses. For example, the output of an up counter will be 0-1-2-3...

- **Down counters** are the counters that count from large to small count. Their output goes on decreasing as they receive clock pulses. For example, the output of a down counter will be 7-6-5-4-3...
- **Up/Down counter** is the combination of up counter and down counter.

8.2 Asynchronous/Ripple Up Counters :

- Fig. 8.2.1 shows the logic diagram of a 2-bit ripple up counter. The number of flip-flops used is 2. Note that the number of bits will always be equal to the number of flip-flops. Thus a 4 bit counter will use four flip-flops.
- The toggle (T) flip-flops are being used. But we can use the JK flip-flops also with J and K connected permanently to logic 1.
- External clock is applied to the clock input of flip-flop A which is the LSB flip-flop and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.



(C-771) **Fig. 8.2.1 : A two bit asynchronous binary up counter**

Operation of the counter :

- Initially let both the flip-flops be in reset condition.

$$\therefore \quad Q_B Q_A = 00$$

On the first negative going clock edge :

- As soon as the first falling edge of the clock hits FF-A, it will toggle as $T_A = 1$. Hence Q_A will become equal to 1.
- Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in the status of Q_B because FF-B is a negative edge triggered FF.
- Therefore after the first clock pulse the counter outputs are

$$Q_B Q_A = 01 \qquad \text{..... After the first CLK pulse}$$

At the second falling edge of clock :

- On the arrival of second falling clock edge, FF-A toggles again and Q_A changes from 1 to 0.

$$\therefore \quad Q_A = 0 \qquad \text{...Corresponding to } 2^{nd} \text{ negative clock edge.}$$

- This change in Q_A (from 1 to 0) acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will change from 0 to 1.

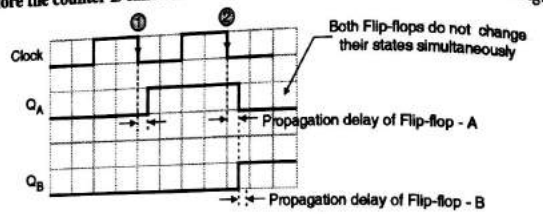
$$\therefore \quad Q_B = 1$$

- Hence after the second clock pulse the counter outputs are

$$Q_B Q_A = 10 \qquad \text{..... After the second CLK pulse}$$

Scanned by CamScanner

- Note that both the outputs are changing their state. But both the changes do not take place simultaneously. Q_A will change first from 1 to 0 and then Q_B will change from 0 to 1. This is due to the propagation delay of FF-A. So both flip-flops will never trigger at the same instant. **Therefore the counter is called as an asynchronous counter.** This is shown in Fig. 8.2.2.



(C-772) **Fig. 8.2.2 : FFs do not change their state simultaneously**

At the third falling edge of clock :

- On arrival of the third falling edge, FF-A toggles again and Q_A becomes 1 from 0.
- Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1.

$\therefore \quad Q_B Q_A = 11$ ..... After the third CLK pulse

At the 4th negative clock edge :

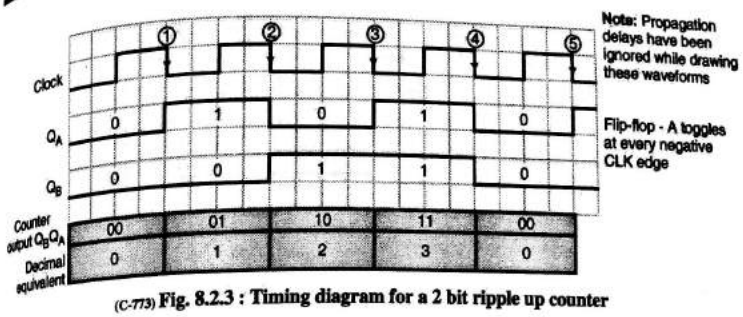
- On the 4th falling clock edge, FF-A toggles and Q_A changes from 1 to 0.
- This negative going change in Q_A acts as a negative clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0.

$\therefore \quad Q_B Q_A = 00$ ..... After the fourth CLK pulse

- So the counter has reached the original state. The counter operation will now repeat.
- Table 8.2.1 summarizes the operation of the counter and Fig. 8.2.3 shows the timing waveforms.

Table 8.2.1 : Summary of operation of a 2-bit binary ripple up counter

Clock	Counter outputs		State number	Decimal equivalent of counter output
	Q_B (MSB)	Q_A (LSB)		
Initially	0	0	–	0
1st (↓)	0	1	1	1
2nd (↓)	1	0	2	2
3rd (↓)	1	1	3	3
4th (↓)	0	0	4	0



(C-773) **Fig. 8.2.3 : Timing diagram for a 2 bit ripple up counter**

Why is it called counter ?

See Fig. 8.2.3. The decimal count corresponds to the number of clock pulses, which counter has received. Thus this circuit counts the clock pulses. Hence it is called as counter.

Number of states :

- As seen from Table 8.2.1, this counter has four distinct states of output namely 00, 01, 10 and 11. In general the number of states = 2^n where n is equal to the number of flip-flops.

Maximum count :

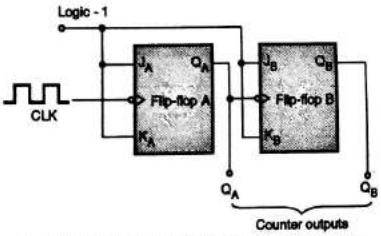
- As seen from Table 8.2.1, the maximum count is 3 (decimal) i.e. 1 1 binary.

$$\text{Maximum count} = 3 = 2^2 - 1$$

- In general the maximum count = $(2^n - 1)$, where n = Number of flip-flops.

8.2.1 Two Bit Asynchronous Up Counter using JK Flip-Flops :

- A 2 bit asynchronous counter up using JK flip-flops is shown in Fig. 8.2.4. Note that the J and K inputs of both the flipflops are connected to logic 1 so actually JK flip-flops are converted into T flip-flops.
- The operation of this circuit is exactly same as that of the counter using the T flip-flops.

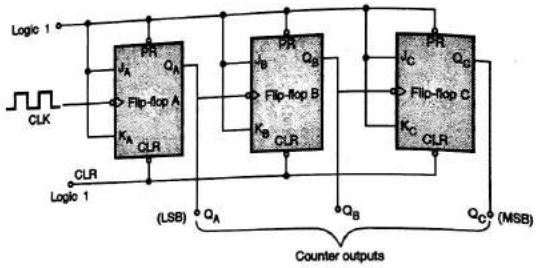


(C-774) **Fig. 8.2.4 : Two bit ripple up counter using JK flip-flops**

8.2.2 3 Bit Asynchronous Up Counter :

- We can apply all the basic concepts which were introduced for the 2-bit ripple up counter to the 3-bit ripple up counter.

Scanned by CamScanner



(C-775) **Fig. 8.2.5 : 3-bit ripple up counter**

- Fig. 8.2.5 shows the logic diagram of a 3-bit ripple up counter. Since it is a 3-bit counter, we need to use 3-flip-flops.
- Operation of the 3-bit ripple up counter takes place in exactly similar manner as that of a 2-bit counter.
- Table 8.2.2 summarizes the operation of the 3-bit asynchronous up counter.
- Note that the asynchronous preset and clear terminals are also being used. Both of them are active low inputs. Hence for the normal output of the counter preset and clear terminals should be connected to logic 1.

Table 8.2.2 : Summary of operation of a 3-bit ripple up counter

Clock	Flip-flop outputs			State	Decimal equivalent
	Q_C (MSB)	Q_B	Q_A (LSB)		
Initially	0	0	0	1	0
1st (↓)	0	0	1	2	1
2nd (↓)	0	1	0	3	2
3rd (↓)	0	1	1	4	3
4th (↓)	1	0	0	5	4
5th (↓)	1	0	1	6	5
6th (↓)	1	1	0	7	6
7th (↓)	1	1	1	8	7
8th (↓)	0	0	0	1	0

Number of states :

Number of states = $2^n = 2^3 = 8$.

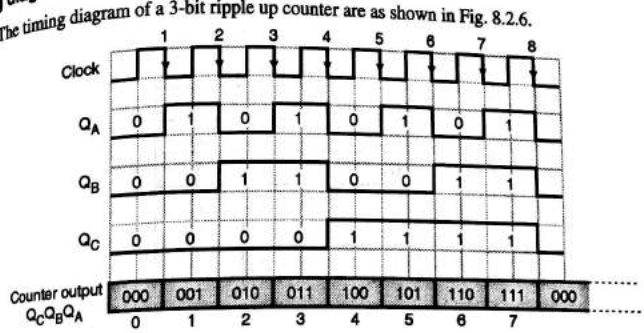
The 3 bit ripple up counter can have 8 distinct states i.e. $Q_C\ Q_B\ Q_A$ can take up values from 000, 001, 010, ...110, 111.

Maximum count :

Maximum count = $2^n - 1 = 8 - 1 = 7$. Refer Table 8.2.2 the maximum count is $Q_C\ Q_B\ Q_A$ = 1 1 1 i.e. decimal 7. Note that Q_C is treated as MSB and Q_A as LSB.

Timing diagram :

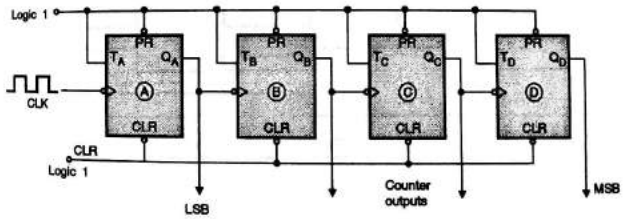
The timing diagram of a 3-bit ripple up counter are as shown in Fig. 8.2.6.



(C-776) **Fig. 8.2.6 : Timing diagram for a 3-bit ripple up counter**

8.2.3 4 Bit Asynchronous up Counter :

- Fig. 8.2.7(a) shows the circuit diagram of a 4 bit asynchronous counter using the T flip flops.



(C-777) **Fig. 8.2.7(a) : 4 bit asynchronous up counter**

- Since it is a 4 bit ripple up counter, we need to use four flip flops.
- Initially all the flip flops have zero output.

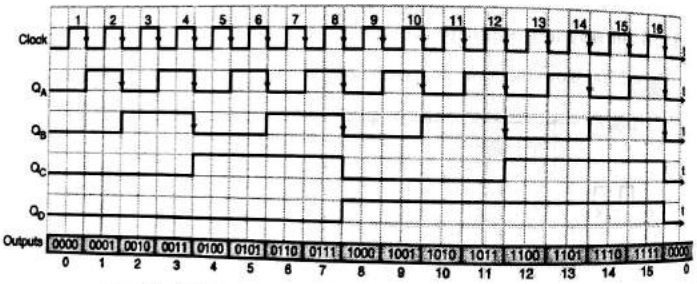
$$\therefore Q_DQ_CQ_BQ_A = 0000.$$

- All the flip flop are negative edge triggered. CLK signal is applied to the clock input of FF-A whereas Q outputs of every F/F is applied to the clock input of next F/F. For example Q_A to CLK of FF-B, Q_B to CLK of FF-C and so on.
- Table 8.2.3 shows the truth table for 4 bit asynchronous up counter. Its output passes through 16 states form 0000 i.e. $(0)_{10}$ to (1111) i.e. $(15)_{10}$.
- After 1111, the output again become 0000 and the operation repeats itself.
- Fig. 8.2.7(b) shows the timing diagram for the 4-bit asynchronous up counter.
- Q_D acts as MSB of the output whereas Q_A acts as the LSB.

Scanned by CamScanner

Table 8.2.3 : Truth table for a 4-bit asynchronous up counter

Clock	FF outputs				Decimal
	Q_D	Q_C	Q_B	Q_A	
Initially	0	0	0	0	0
1 (↓)	0	0	0	1	1
2 (↓)	0	0	1	0	2
3 (↓)	0	0	1	1	3
:		:			:
:		:			:
14 (↓)	1	1	1	0	14
15 (↓)	1	1	1	1	15
16 (↓)	0	0	0	0	0

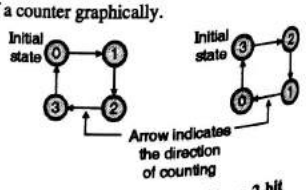


(C-778) **Fig. 8.2.7(b) : Waveforms of a 4 bit asynchronous up counter**

- The number of state through which the output of a 4 bit up counter passes is 16 (from 0 to 15).
- The maximum count is 15 i.e. 1111. Thus a 4-bit ripple up counter will count from 0000 to 1111.

8.2.4 State Diagram of a Counter :

- The state diagram of a counter represents the states of a counter graphically.
- For example for a 2-bit up counter the state diagram is shown in Fig. 8.2.8(a) and for a 2-bit down counter the state diagram is shown in Fig. 8.2.8(b).
- The number written inside a circle represents the state number, whereas the arrow shows the direction of counter (up or down).



(a) For a 2-bit up counter **(b) For a 2-bit down counter**

(C-779) **Fig. 8.2.8 : State diagram**

- Note that in the counters only the state is important. Hence in the state diagram we have not shown any input or output conditions.

8.3 Asynchronous Down Counters :

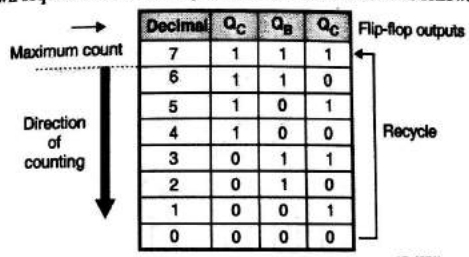
8.3.1 3- Bit Asynchronous Down Counter :

MU : Dec. 15

University Questions

Q. 1 Draw a circuit diagram for 3-bit asynchronous binary down counter using master-slave JK flip-flops. Show the output of each flip-flop with reference to the clock and justify that the down counting action. Also prove from the timing diagram that the counter is "divide by 8" counter. **(Dec. 15, 10 Marks)**

- All the counters discussed so far have counted upwards from zero. So they can be called as **up counters.**
- But the counters which can count in the **downward** direction i.e. from the maximum count to zero are called **down counters.**
- The countdown sequence for a 3-bit asynchronous down counter is as follows :



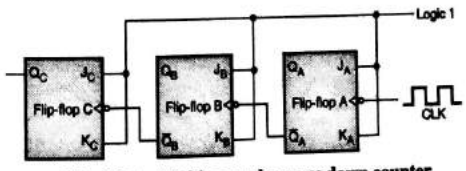
Decimal	Q_C	Q_B	Q_C
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0

(C-4871)

- Thus counting takes place as follows :

$$Q_C\,Q_B\,Q_A = 111, 110, 101, 100, 011, 010, 001, 000.$$

- From this sequence it is evident that FF-A should toggle at every negative going clock edge but FF-B should change its state only at those instants when Q_A changes from LOW (0) to HIGH (1) and Q_C should changes only when Q_B changes from LOW to HIGH.
- Thus in a down counter, each FF except the first one (FF-A) should toggle when the output of its preceding flip-flop changes from LOW to HIGH.
- If all the FFs are negative edge triggered i.e. responding to the negative CLK edge, then we can place an inverter in front of every CLK input or we can drive the CLK input of next FF from the $\overline{Q}$ output of the preceding FF and not from the Q outputs as shown in Fig. 8.3.1.
- A 3-bit asynchronous down counter is shown in Fig. 8.3.1. The clock input is applied directly to the clock input of FF-A. But $\overline{Q_A}$ is connected to clock of FF-B, $\overline{Q_B}$ to clock of FF-C and so on.



(C-781) **Fig. 8.3.1 : A 3-bit asynchronous down counter**

Operation :

- Initially let all the flip-flops be in the reset condition.

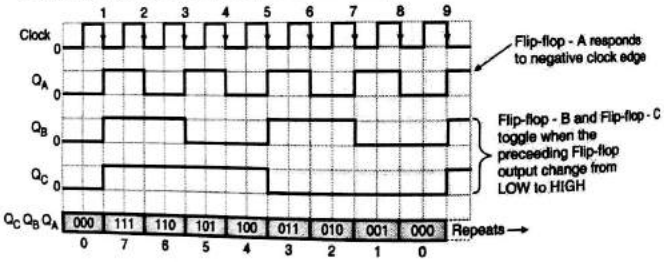
$$\therefore\ Q_C\,Q_B\,Q_A = 000$$

Scanned by CamScanner

- As soon as the first falling clock pulse arrives, FF-A toggles. So Q_A becomes 1 and $\overline{Q_A}$ changes from 1 to 0.
- The negative going change in $\overline{Q_A}$ acts as a clock to FF-B. Hence FF-B will change its state. So Q_B becomes 1 and $\overline{Q_B}$ changes from 1 to 0. This negative going change in $\overline{Q_B}$ acts as a clock to FF-C. Hence FF-C will change its state. So Q_C becomes 1 and $\overline{Q_C}$ becomes 0.
- Thus after the first clock pulse the output of counter are,

  $Q_C\,Q_B\,Q_A = 1\,1\,1$ ..... After the 1st CLK pulse
- Corresponding to the second falling clock edge, FF-A toggles. Q_A becomes 0 and $\overline{Q_A}$ becomes 1. This positive going change in $\overline{Q_A}$ does not alter the state of FF-B. So Q_B remains 1 and $\overline{Q_B}$ remains 0. So there is no change in the state of FF-C. Hence after the second clock pulse the counter outputs are,

  $Q_C\,Q_B\,Q_A = 1\,1\,0$ ..... After the 2nd CLK pulse
- The down counting will thus take place. Similarly the counter will count down to pass through the states 101, 100, 011, 010, 001 and 000. The operation repeats itself thereafter.
- The timing diagram for the down counter is shown in Fig. 8.3.2.

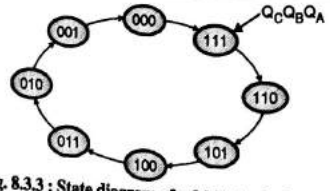


(C-782) **Fig. 8.3.2 : Timing diagram of a 3-bit down counter**

State diagram :

The state diagram of the 3-bit down counter is shown in Fig. 8.3.3.

Note : The numbers inside the circles correspond to the state of $Q_C\,Q_B\,Q_A$.



(C-783) **Fig. 8.3.3 : State diagram of a 3 bit ripple down counter**

Ex. 8.3.1 : Draw 4 bit down counter, explain its working with timing diagram and truth table.

Soln. : The count sequence for a 4 bit down counter is shown in Table P. 8.3.1.

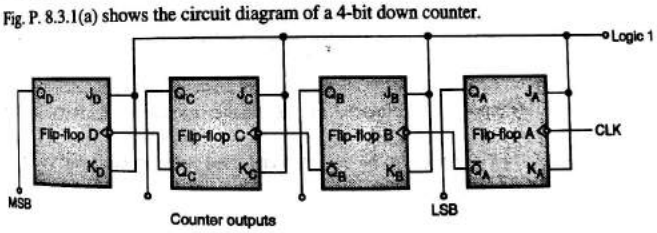
Table P. 8.3.1 : Truth table of a 4 bit down counter

CLK	Flip Flop outputs				State	Decimal Equivalent
	Q_D	Q_C	Q_B	Q_A		
Initially	0	0	0	0	1	0
1(↓)	1	1	1	1	2	15
2(↓)	1	1	1	0	3	14
3(↓)	1	1	0	1	4	13
4(↓)	1	1	0	0	5	12
:	:	:	:	:	:	:
15(↓)	0	0	1	0	15	2
16(↓)	0	0	0	1	16	1
1(↓)	0	0	0	0	1	0
2(↓)	1	1	1	1	2	15

↓ Repeats

Circuit diagram :

Fig. P. 8.3.1(a) shows the circuit diagram of a 4-bit down counter.



(C-784) **Fig. P. 8.3.1(a) : 4 bit asynchronous down counter**

Operation :

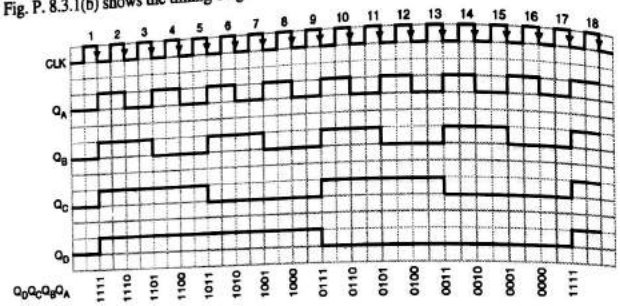
- Initially let $Q_D\,Q_C\,Q_B\,Q_A = 0\,0\,0\,0$
- On the first clock pulse, flip flop A toggles and Q_A becomes 1 from 0. Thus flip flop B will change its state.
- Q_B becomes 1 and $\overline{Q_B}$ becomes 0 from 1. $\overline{Q_B}$ acts as a clock to flip flop C. Thus flip flop C will change its state, $Q_C = 1$ and $\overline{Q_C}$ becomes 0. Similarly Q_D becomes 1 and $\overline{Q_D} = 0$. Thus after first clock pulse.

  $Q_D\,Q_C\,Q_B\,Q_A = 1\,1\,1\,1$
- On the second clock pulse, flip flop A toggles. Q_A becomes 0 and $\overline{Q_A}$ becomes 1. This positive going change in $\overline{Q_A}$ does not alter the state of flip flop B.
- So Q_B remains 1 and $\overline{Q_B}$ remains 0. There is no change in states of flip-flop C and D. Thus after second clock pulse.

Scanned by CamScanner

$Q_D\ Q_C\ Q_B\ Q_A = 1\ 1\ 1\ 0$

- The down counting thus takes place.
- Fig. P. 8.3.1(b) shows the timing diagram.



(C-785) **Fig. P. 8.3.1(b) : Timing diagram of 4 bit down counter**

- The 4-bit down counter undergoes 16 different states (1111 to 0000) and the maximum count is 15 i.e. 1111.

8.4 UP / DOWN Counters :

- We have designed the up counters and the down counters separately.
- But in practice both these modes are generally combined together and an UP/DOWN counter is formed.
- A mode control (M) input is also provided to select either up count or down count mode of operation.
- A combinational circuit is required to be designed and used between each pair of flip-flops in order to achieve the up/down operation.

Types of up/down counters :

- The up/down counters are of two types :
  1. UP/DOWN ripple counters. 2. UP/DOWN synchronous counters.

8.4.1 UP/DOWN Ripple Counters :

- In the up/down ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used.
- The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from Q or $\bar{Q}$ output of the previous FF.

UP counting mode (M = 0) :

- The CLK signal is applied directly to the clock input of the LSB flip-flop. For the remaining flip-flops, the Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M = 0).

DOWN counting mode (M = 1) :

- The clock signal is applied directly to the clock input of the LSB flip-flop. For the remaining flip-flop, the $\bar{Q}$ output of the preceding FF is connected to the clock of the next FF. This will operate the counter in the down counting mode. For down counting mode the mode select input M is kept at logic 1 (M = 1).

8.4.2 3-bit and 4-bit Up Down Ripple Counters :

The design of 3-bit and 4-bit up down ripple counters has been illustrated in the following examples.

Ex. 8.4.1 : Design a 3-bit binary up/down ripple counter. Draw the timing diagram. May 07, Dec. 08, 10 Marks

Soln. :

- The requirements of counter are :
  1. 3-bit : Hence three FFs are required. 2. UP/DOWN : So a mode control input is essential.
- We know that for a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- And for a ripple down counter, the $\bar{Q}$ output of the preceding FF is connected to the clock input of the next one.
- Let the selection of Q or $\bar{Q}$ output of the preceding FF be controlled by the mode control input M such that,

  If M = 0 ..... UP counting. So connect Q to CLK

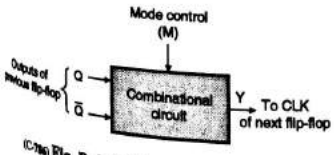
  If M = 1 ..... DOWN counting. So connect $\bar{Q}$ to CLK
- Let us design a combinational logic to satisfy all the requirements stated above.

Truth table of combinational circuit :

The truth table of such a combinational circuit is shown in Table P. 8.4.1.

Table P. 8.4.1 : Truth table

Inputs			Output	
M	Q	$\bar{Q}$	Y	
0	0	0	0	Y = Q for up counting
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$Y = \bar{Q}$ for down counting
1	0	1	1	
1	1	0	0	
1	1	1	1	



(C-786) **Fig. P. 8.4.1(a) : Block diagram of combinational circuit**

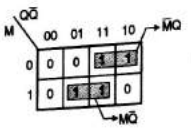
K-map and simplified expression for output :

Fig. P. 8.4.1(b) shows the K-map and simplified expression for Y.

Scanned by CamScanner

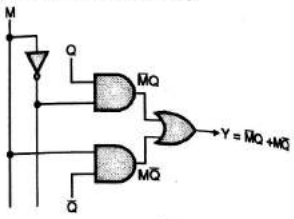
Logic diagram for combinational circuit :

The logic diagram for the combinational circuit is shown in Fig. P. 8.4.1(c).



Expression for Y

$Y = \overline{M}Q + M\overline{Q}$

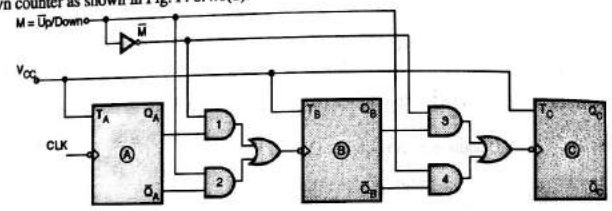


(C-787) **Fig. P. 8.4.1(b) : K-map for Y**

Fig. P. 8.4.1(c)

Logic diagram of a 3-bit up/down counter :

The combinational circuit is connected between every pair of flip-flops to obtain the 3-bit up/down counter as shown in Fig. P. 8.4.1(d).



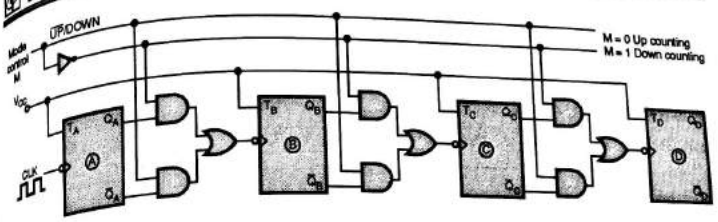
(C-788) **Fig. P. 8.4.1(d) : A 3-bit up / down ripple counter**

Operation of 3-bit up down ripple counter :

1. **With M = 0 (Up counting mode) :**
   - If M = 0 and $\overline{M}$ = 1, then the AND gates 1 and 3 in Fig. P. 8.4.1(d) will be enabled whereas the AND gates 2 and 4 will be disabled.
   - Hence Q_A gets connected to the clock input of FF-B and Q_B gets connected to the clock input of FF-C. These connections are same as those for the normal up counter. Thus with M = 0 the circuit works as an up counter.
2. **With M = 1 (Down counting mode) :**
   - If M = 1, then AND gates 2 and 4 in Fig. P. 8.4.1(d) are enabled whereas the AND gates 1 and 3 are disabled. Hence $\overline{Q_A}$ gets connected to the clock input of FF-B and $\overline{Q_B}$ gets connected to the clock input of FF-C. As discussed earlier, these connections will produce a down counter. Thus with M = 1 the circuit works as a down counter.

Ex. 8.4.2 : Design a 4-bit up down ripple counter.

Soln. : Refer Ex. 8.4.1. The combinational circuit is going to be the same. But we have to use 4 flip-flops. The 4-bit up down counter is as shown in Fig. P. 8.4.2.



(C-789) **Fig. P. 8.4.2 : 4-bit up/down ripple counter**

The circuit will work as up counter with M = 0 whereas it will operate as a down counter when M = 1.

8.5 Modulus of the Counter (MOD-N Counter) :

- The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter where

  MOD number = 2^n
- So we can conclude that **modulus** of a counter represents the **number of states** through which the counter progresses during its operation.
- Can we have a modulo-5 counter using a 3-bit ripple counter ? That means can we restrict the number of states to only 5 instead of 8 under normal conditions ?
- The answer is yes. We can design such modulo counters with the help of the basic ripple counter structure and a combinational logic called reset logic.
- Table 8.5.1 shows the relation between 2, 3 and 4 bit counters and their modulus.

Table 8.5.1

Counter type	Modulus
2 bit up or down	MOD-4
3 bit up or down	MOD-8
4 bit up or down	MOD-16

Ex. 8.5.1 : Design a Modulo-5 ripple counter using a 3-bit ripple counter.

OR

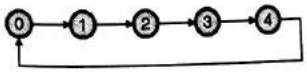
Design MOD-5 asynchronous counter, and also draw the waveforms.

Dec. 02. 4 Marks. May 16. 10 Marks

Soln. :

Step 1 : Draw the state diagram :

The state diagram of MOD-5 ripple counter is as shown in Fig. P. 8.5.1(a).



(C-794) **Fig. P. 8.5.1(a) : State diagram of a MOD-5 ripple counter**

Scanned by CamScanner

Step 2 : Write truth table for the reset logic :

- Table P. 8.5.1 shows the truth table for the reset logic.

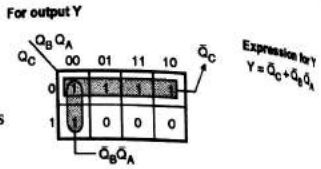
Table P. 8.5.1 : Truth table for the reset logic

State	Flip-flop outputs Q_C	Q_B	Q_A	Output Y of reset logic
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Valid states

Invalid states

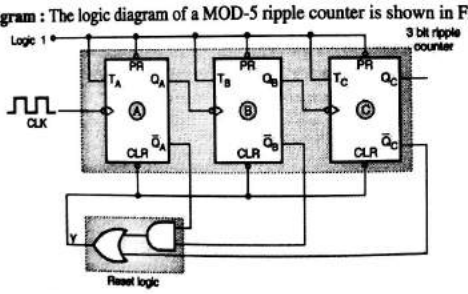
Step 3 : K-map :



(C-795) **Fig. P. 8.5.1(b) : K-map and simplification**

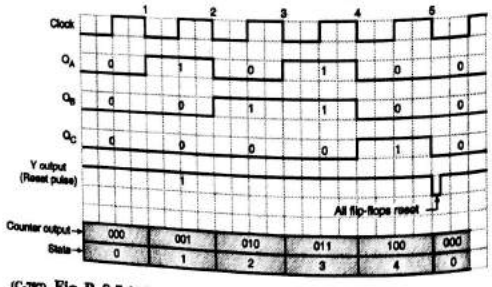
- The K map is as shown in Fig. P. 8.5.1(b).
- The states 0 through 4 are valid states and the output Y of reset logic (Y) is inactive (1) for them.
- The states 5, 6 and 7 are invalid states. If counter enters into any one of these states that Y = 0 (active) and will reset all the flip-flops.

Step 4 : Logic diagram : The logic diagram of a MOD-5 ripple counter is shown in Fig. P. 8.5.1(c).



(C-796) **Fig. P. 8.5.1(c) : Logic diagram of MOD-5 ripple counter**

Step 5 : Timing diagram : The timing diagram is as shown in Fig. P. 8.5.1(d).



(C-797) **Fig. P. 8.5.1(d) : Timing diagram of MOD-5 ripple counter**

Ex. 8.5.2 : Draw the divide by 7 (MOD-7) asynchronous up counter using T flip-flop. Write truth table, draw the timing diagram. **Dec. 04. 6 Marks**

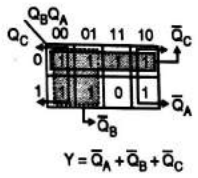
Soln. :

Step 1 : Truth table :

The truth table of divide by 7 (MOD-7) asynchronous up counter is shown in Table P. 8.5.2. Note that Y represents the output of the reset logic used for resetting the flip flops.

Table P. 8.5.2

CLK	Q_C	Q_B	Q_A	Y output
Initially	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0
8	0	0	0	1



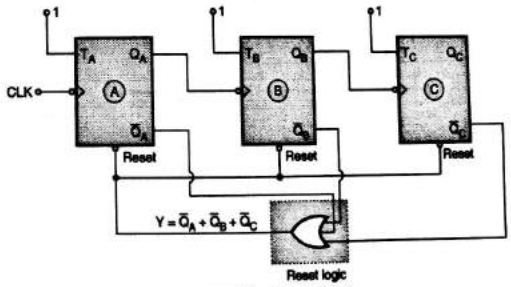
(C-798) **Fig. P. 8.5.2(a)**

Step 2 : K map :

Fig. P. 8.5.2(a) shows the K map and simplified expression.

Step 3 : Draw the circuit :

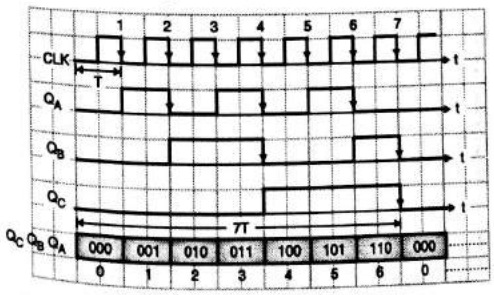
Divide by 7 counter is shown in Fig. P. 8.5.2(b).



(C-799) **Fig. P. 8.5.2(b)**

Step 4 : Timing diagram :

The timing diagram is shown in Fig. P. 8.5.2(c).



(C-800) **Fig. P. 8.5.2(c) : Timing diagram of MOD-7 counter**

Scanned by CamScanner

Ex. 8.5.3 : For MOD-11 asynchronous up counter.
1. Draw circuit diagram. Use T flip-flop.
2. Write truth table.
3. Draw timing diagram. 4. If the output frequency is 11 kHz what is the clock input?

May 04. 6 Marks

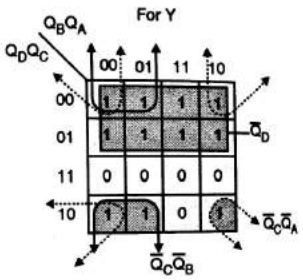
Soln. :

Step 1 : Write the truth table :

Table P. 8.5.3

Q_D	Q_C	Q_B	Q_A	Y output
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Invalid states



$\therefore\ Y = \overline{Q}_D + \overline{Q}_C\overline{Q}_A + \overline{Q}_C\overline{Q}_B$

$\therefore\ Y = \overline{Q}_D + \overline{Q}_C(\overline{Q}_A + \overline{Q}_B)$

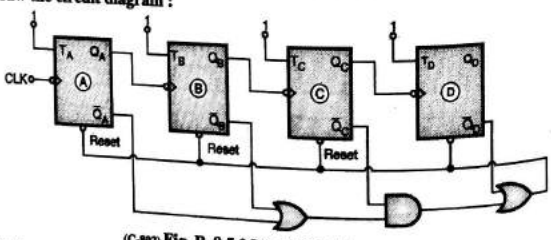
(C-801) **Fig. P. 8.5.3(a) : K map**

Step 2 : Draw the K map :

The K map is shown in Fig. P. 8.5.3(a). The simplified equation for Y output of the reset logic is as follows :

$$Y = \overline{Q}_D + \overline{Q}_C\left(\overline{Q}_A + \overline{Q}_B\right)$$

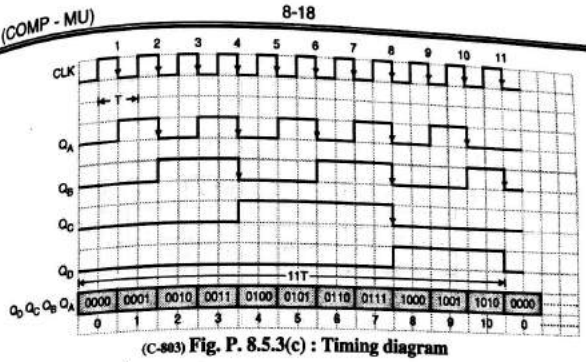
Step 3 : Draw the circuit diagram :



(C-802) **Fig. P. 8.5.3(b) : MOD 11 counter**

Step 4 : Timing diagram :

- The timing diagram is shown in Fig. P. 8.5.3(c).



(C-803) **Fig. P. 8.5.3(c) : Timing diagram**

Output frequency $f_o = \dfrac{f_{CLK}}{11}$ $\quad\therefore\quad f_{CLK} = 11\ f_o = 11 \times 11$ kHz

$\therefore\quad f_{CLK} = 121$ kHz **...Ans.**

Ex. 8.5.4 : Design a mod 11 counter using JK flipflops. May 04. 8 Marks

Soln. :

Step 1 : Decide number of flip-flops : Since the number states are 11 we need 4 flip-flops.

Step 2 : Excitation tables : The excitation table for J-K flip-flops.

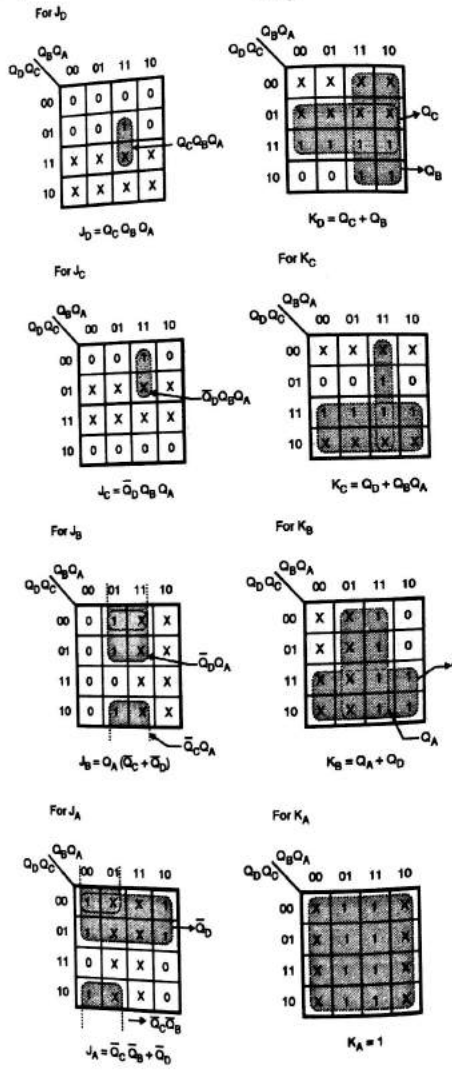
Present state Q	Next state Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

Circuit excitation table :

Present state				Next state				Flip flop inputs							
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	1	0	0	×	0	×	1	×	×	1
0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	×
0	0	1	1	0	1	0	0	0	×	1	×	×	1	×	1
0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	×
0	1	0	1	0	1	1	0	0	×	×	0	1	×	×	1
0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	×
0	1	1	1	1	0	0	0	1	×	×	1	×	1	×	1
1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	×
1	0	0	1	1	0	1	0	×	0	0	×	1	×	×	1
1	0	1	0	0	0	0	0	×	1	0	×	×	1	0	×
1	0	1	1	0	0	0	0	×	1	0	×	×	1	×	1
1	1	0	0	0	0	0	0	×	1	×	1	0	×	0	×
1	1	0	1	0	0	0	0	×	1	×	1	0	×	×	1
1	1	1	0	0	0	0	0	×	1	×	1	×	1	0	×
1	1	1	1	0	0	0	0	×	1	×	1	×	1	×	1

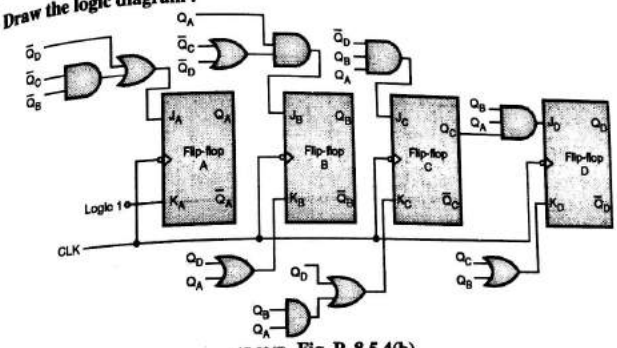
Scanned by CamScanner

Step 3 : K-map and simplification :



(C-3346) **Fig. P. 8.5.4(a)**

Step 4 : Draw the logic diagram :



(C-3347) **Fig. P. 8.5.4(b)**

Ex. 8.5.5 : Design an asynchronous counter for the following counting sequence.
– 12 – 11 – 10 – 9 – 8 – 7 – 6 – 5 – 12 – **Dec. 03, 8 Marks**

Soln. : From the given counting sequence the highest state is 12 and the counter is down counter. It is observed that ;

1. 12, 11, 10, 9, 8, 7, 6, 5 are the valid states.
2. 4, 3, 2, 1, 0, 13, 14, 15 are invalid states.
3. As soon as the counter reaches 5 i.e. 0101, all the flip should be set and counter should return back to state 9.

Step 1 : Decide the number of flip flops (n) :

Highest state = 12 $\therefore 12 \le 2^n$ $\therefore n = 4$

So 4 flip flops should be used.

Step 2 : Truth table :

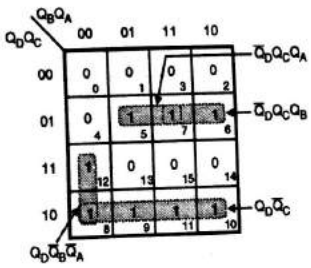
State	Flip Flop Outputs				Output Y
	Q_D	Q_C	Q_B	Q_A	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Invalid States

Valid States

Invalid States

Step 3 : K-map and simplification for output Y :

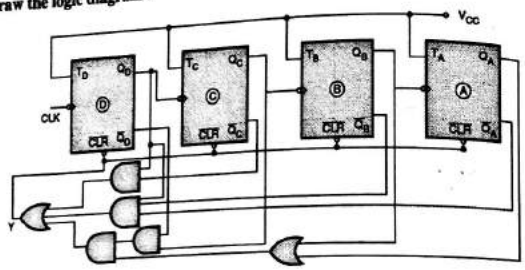


(C-3712) **Fig. P. 8.5.5**

Scanned by CamScanner

$$\therefore Y = Q_D\,\overline{Q_C} + Q_D\,\overline{Q_B}\,\overline{Q_A} + \overline{Q_D}\,Q_C\,(Q_B + Q_A)$$

Step 4 : Draw the logic diagram :



(C-3713) **Fig. P. 8.5.5(a)**

Ex. 8.5.6 : Design a mod-12 asynchronous down counter. **May 03. 8 Marks**

Soln. :

Step 1 : Decide the number of flip-flops :

Since we have to design MOD-12 counter, 4 flip-flops are required.

Step 2 : Truth table for the result logic :

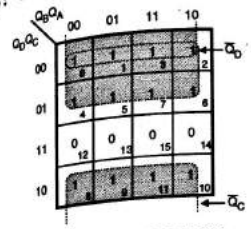
State	Flip Flop Outputs				Output of reset logic Y
	Q_D	Q_C	Q_B	Q_A	
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Valid States (0 – 11)

Invalid States (12 – 15)

1. The states 0 through 11 are valid states and the output y of reset logic (y) is inactive (1) for them.
2. The states 12, 13, 14 and 15 are invalid state.
   If counter enters into any one of these states then y = 0 (active) and will reset all the flip-flops.

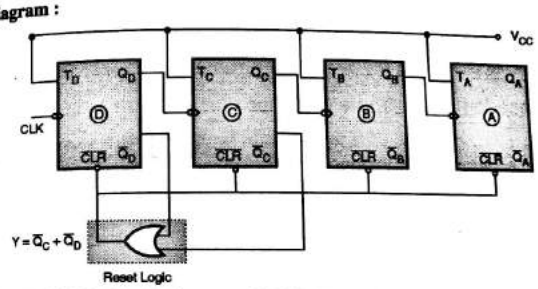
Step 3 : K-map and simplification for y output :



$$Y = \overline{Q_C} + \overline{Q_D}$$

(C-3714) **Fig. P. 8.5.6(a)**

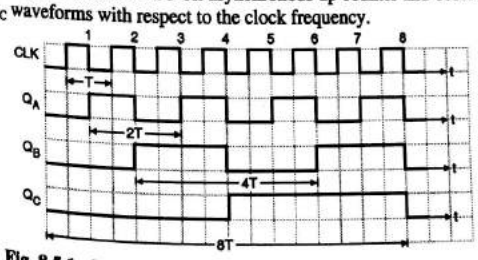
Step 4 : Logic diagram :



(C-3715) **Fig. P. 8.5.6(b) : Logic diagram of MOD-12 asynchronous down counter**

8.5.1 Frequency Division Taking Place in Asynchronous Counters :

- In the chapter on flip-flops, we have seen that a flip-flop in toggle mode divides the clock frequency by 2.
- That means the frequency of Q or $\overline{Q}$ output waveform of a toggle flip-flop is exactly half of the clock frequency.
- The concept of frequency division is applicable to the counters as well because we use flip-flops in the toggle mode for counters.
- Refer to the timing waveforms of a 3-bit asynchronous up counter and observe the frequencies of Q_A, Q_B and Q_C waveforms with respect to the clock frequency.



(C-897) **Fig. 8.5.1 : Timing waveforms of a 3-bit asynchronous up counter**

Scanned by CamScanner

Conclusions :

- Let one cycle period of the clock signal be T sec. Hence the clock frequency $f_{CLK} = (1/T)$ Hz.
- Output of the least significant flip-flop (i.e. FF-A) has a one cycle period of (2T) as shown in Fig. 8.5.1. Hence

$$\text{Frequency of } Q_A \text{ output} = f_A = \frac{1}{2T} = \frac{f_{CLK}}{2}$$

- The one cycle period of Q_B is 4T. Hence the frequency of Q_B is given by,

$$f_B = \frac{1}{4T} = \frac{f_{CLK}}{4} \qquad \text{..... since } \frac{1}{T} = f_{CLK}$$

- Similarly the frequency of Q_C output is given by,

$$f_C = \frac{1}{8T} = \frac{f_{CLK}}{8}$$

Note : In any counter, the signal at the output of last FF (i.e. MSB) will have a frequency equal to the input clock frequency divided by the MOD number of the counter.

8.6 Decade (BCD) Ripple Counter :

MU : Dec. 05, May 08, Dec. 11, Dec. 12, Dec. 13, Dec. 14, May 16

University Questions

Q. 1 Design a decade counter using 4 JKFF. **(Dec. 05, 10 Marks)**

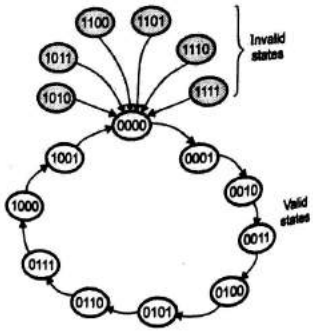
Q. 2 1. Design mod 10 asynchronous up counter with the help of necessary decoding logic.
2. Give the timing diagram for the counter of part. **(May 08, 8 Marks)**

Q. 3 Write short notes on : Decade counters. **(Dec. 11, Dec. 14, 7 Marks)**

Q. 4 Write short notes on : Decade counters. **(Dec. 12, May 16, 5 Marks)**

Q. 5 Design mod-10 synchronous counter using JK flip flops. Check for the lock out condition. If so, how the lock-out condition can be avoided ? Draw the neat state diagram and circuit diagram with flip flops. **(Dec. 13, 15 Marks)**

- The state diagram of an BCD ripple counter is shown in Fig. 8.6.1. It shows that a BCD counter counts from 0000 to 1001 i.e. from 0 to 9.
- Four JK flip-flops with a clear input are to be used. The output of the **reset logic** is to be connected to the clear input of all the FFs.
- The output of the reset logic Y = 1 for all the valid states from 0 to 9 and Y = 0 for all the invalid outputs of the counter i.e. from 10 to 15.
- If the counter enters into any invalid state then it will be reset automatically.



(C-811) **Fig. 8.6.1 : State diagram of a BCD counter**

The truth table of the reset logic is given in Table 8.6.1.

Table 8.6.1 : Truth table of a BCD counter

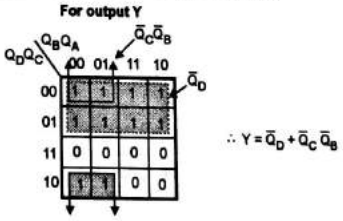
CLK	Q_D	Q_C	Q_B	Q_A	Output of reset logic
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
–	1	0	1	0	0
–	1	0	1	1	0
–	1	1	0	0	0
–	1	1	0	1	0
–	1	1	1	0	0
–	1	1	1	1	0

Valid states

Invalid states

K-map and simplifications :

The K-map for Y output is shown in Fig. 8.6.2.

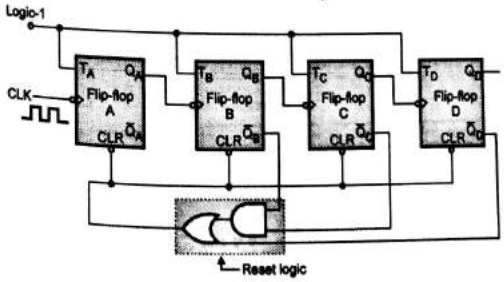


$\therefore Y = \bar{Q}_D + \bar{Q}_C \bar{Q}_B$

(C-812) **Fig. 8.6.2**

Logic diagram :

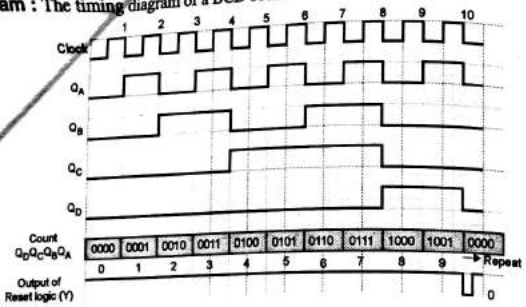
The logic diagram of a BCD counter are as shown in Fig. 8.6.3.



(C-813) **Fig. 8.6.3 : Logic diagram of a BCD counter**

Scanned by CamScanner

Timing diagram : The timing diagram of a BCD counter is shown in Fig. 8.6.4.



(C-814) **Fig. 8.6.4 : Timing diagram of a BCD ripple counter**

8.7 Problems Faced by Ripple Counters :

The two major problems associated with the ripple counters are as follows :

1. Generation of unwanted short duration pulses called glitch. 2. Propagation delay.

8.7.1 Glitch :

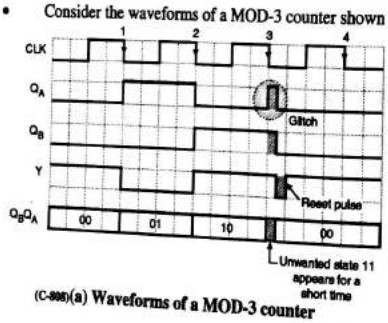
MU : Dec. 02, May 03, Dec. 03, May 04, Dec. 04, May 05, Dec. 05, May 06, Dec. 07, May 08

University Questions

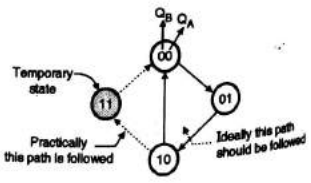
Q. 1 What is glitch problem ?
(Dec. 02, May 03, Dec. 03, May 04, Dec. 04, May 05, Dec. 05, May 06, Dec. 07, May 08, 6 Marks)

Q. 2 What are decoding glitches and how can they be avoided ? **(Dec. 07, 5 Marks)**

- Glitch is a short duration pulse or spike that appears in the outputs of a counter with mod number $< 2^n$.
- Consider the waveforms of a MOD-3 counter shown in Fig. 8.7.1(a).



(C-808)(a) **Waveforms of a MOD-3 counter**



(C-809) **(b) State diagram**

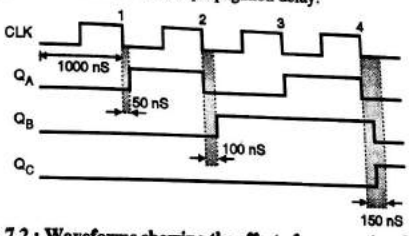
Fig. 8.7.1

- The counter is supposed to pass through the states 00, 01 and 10 only and return to 00 state as shown in the state diagram of Fig. 8.7.1(b).

- But practically what happens is something different. At the third falling edge of CLK, the Q_B and Q_A become 11.
- Hence output of reset logic goes low but after its own propagation delay. Therefore the Q_BQ_A outputs are allowed to be 11 for a short period of time as shown in Fig. 8.7.1(a).
- Note that 11 is an unwanted state. In the Q_A output waveform a short pulse called "Glitch" will appear as shown in Fig. 8.7.1(a).

8.7.2 Problem of Propagation Delay :

- The other major drawback of the ripple counters is that of the propagation delay.
- Each FF is clocked by the transition at the output of the preceding FF.
- Due to the inherent propagation delay (t_{pd}) of each flip-flop, the outputs of FFs do not change simultaneously.
- This problem is illustrated in Fig. 8.7.2.



(C-810) **Fig. 8.7.2 : Waveforms showing the effect of propagation delay**

- Each FF is assumed to have a propagation delay of 50 nS. Then as seen from Fig. 8.7.2 the last FF will take 3 × 50 i.e. 150 nS to change its state after the application of the 4th negative going edge of the clock.
- In Fig. 8.7.2 the clock period is assumed to be 1000 nS. But its clock frequency is raised so that clock period reduces to 100 nS then the $Q_C\ Q_B\ Q_A = 011$ state will never appear as the propagation delay of 150 nS is longer than 1 cycle period of clock.

8.7.3 Maximum Operating Frequency :

- Every flip-flop has its own propagation delay. In ripple counter the output of the previous FF is used as clock for the next FF.
- Hence the propagation delay goes on accumulating. For a 3-bit ripple counter the propagation delay of the first FF gets added to that of the second FF to decide the transition time for the third stage.
- This accumulated time delay is the main problem with the ripple counters, because the propagation delay goes on increasing with increase in number of flip-flops.
- This will put a limitation on the maximum clock frequency.
- The frequency 'f' of a clock pulse for reliable operation of the counter is given by,

$$f \le \frac{1}{n(t_d) + T_s} \qquad \text{... (8.7.1)}$$

where n = Number of flip-flops, T_s = Width of strobe pulse (if used)

And t_d = Propagation delay of one flip-flop.

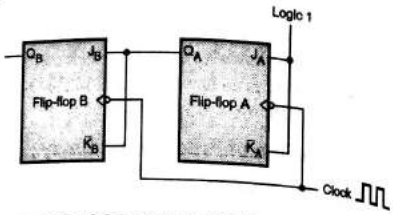
8.8 Synchronous Counters :

If the "clock" pulses are applied to all the flip-flops connected in a counter simultaneously, then such a counter is called as synchronous counter.

Scanned by CamScanner

8.8.1 2-Bit Synchronous up Counter :

- A 2-bit or MOD-4 synchronous counter is shown in Fig. 8.8.1.
- The J_A and K_A inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A. Hence FF-B will toggle if $Q_A = 1$ and there won't be any state change if $Q_A = 0$, at the instant when the negative clock edge is applied.



(C-815) **Fig. 8.8.1 : A 2-bit (MOD-4) synchronous counter**

Operation :

- Initially let both the FFs be in the reset state. Let FF-A be the LSB flip-flop and FF-B be the MSB flip-flop.

$$\therefore Q_B Q_A = 0\,0 \text{ ..... Initially}$$

At the 1st negative clock edge :

- As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1.
- But at the instant of application of negative clock edge, $Q_A = 0 \therefore J_B = K_B = 0$. Therefore FF-B will not change its state. So Q_B will remain 0.

$$\therefore Q_B Q_A = 0\,1 \text{ ..... After the first clock pulse}$$

At the 2nd negative clock edge :

- At the instant when we apply the second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0.
- But at this instant Q_A was 1. So $J_B = K_B = 1$ and FF-B also will toggle. Hence Q_B changes from 0 to 1.

$$\therefore Q_B Q_A = 1\,0 \text{ ..... After the second clock pulse}$$

Next negative clock edges :

- Similarly on application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B.

$$\therefore Q_B Q_A = 1\,1 \text{ ..... After the third clock pulse}$$

- On application of the next clock pulse, Q_A will change from 1 to 0 as Q_B will also change from 1 to 0. Hence

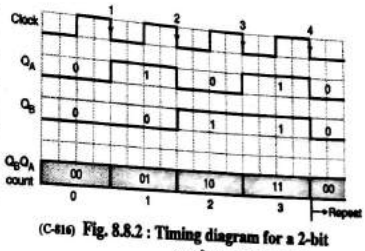
$$\therefore Q_B Q_A = 0\,0 \text{ ..... After the fourth clock pulse}$$

- This is the original state. The operation of counter will repeat after this. The operation is summarised in Table 8.8.1 and the timing diagram is shown in Fig. 8.8.2.

- In this way the 2-bit synchronous counter has four distinct states namely $Q_B Q_A$ = 00, 01, 10 and 11.
- The maximum count of a 2-bit counter is $(11)_2$ or $(3)_{10}$.

Table 8.8.1 : Summary of operation of a 2-bit synchronous counter

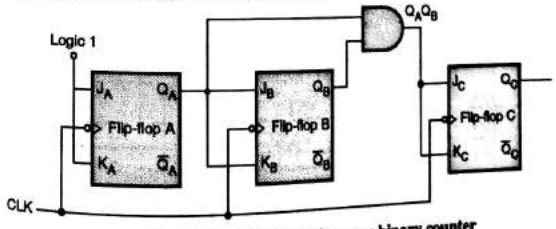
Clock	Counter outputs	
	Q_B (MSB)	Q_A (LSB)
Initially	0	0
1st (↓)	0	1
2nd (↓)	1	0
3rd (↓)	1	1
4th (↓)	0	0



(C-816) **Fig. 8.8.2 : Timing diagram for a 2-bit synchronous counter**

8.8.2 3-Bit Synchronous Binary up Counter :

- We can extend the principle of operation of a 2-bit synchronous counter to a 3-bit counter shown in Fig. 8.8.3.
- FF-A acts as a toggle FF since $J_A = K_A = 1$.
- Q_A output of FF-A is applied to J_B as well as K_B. Hence if $Q_A = 1$ at the instant of triggering, then FF-B will toggle but if $Q_A = 0$ then FF-B will not change its state.
- Q_A and Q_B are ANDed and the output of AND gate is applied to J_C and K_C.
- Hence when Q_A and Q_B both are simultaneously high, then $J_C = K_C = 1$ and FF-C will toggle. Otherwise there is no change in the state of FF-3.



(C-817) **Fig. 8.8.3 : A 3-bit synchronous binary counter**

Operation : Initially all the FFs are in their reset state. $\therefore Q_C Q_B Q_A = 0\,0\,0$

1st clock pulse :

- FF-A toggles and Q_A changes to 1 from 0. But since $Q_A = 0$ at the instant of application of 1st falling clock edge, $J_B = K_B = 0$ and Q_B does not change state. $\therefore Q_B$ remains 0.
- Similarly Q_C also does not change state. $\therefore Q_C = 0$.

$$\therefore Q_C Q_B Q_A = 0\,0\,1 \text{ ..... After 1st clock pulse}$$

Scanned by CamScanner

2nd clock pulse :

- FF-A toggles and Q_A becomes 0.
- But at the instant of application of 2nd falling clock edge Q_A was equal to 1. Hence $J_B = K_B = 1$. Hence FF-B will toggle and Q_B becomes 1.
- Output of AND gate is 0 at the instant of negative clock edge. So $J_C = K_C = 0$. Hence Q_C remains 0.

$$\therefore\ Q_C Q_B Q_A = 0\,1\,0 \qquad \text{...After the 2nd clock pulse}$$

3rd clock pulse :

After the 3rd clock pulse, the outputs are $Q_C Q_B Q_A = 0\,1\,1$.

4th clock pulse :

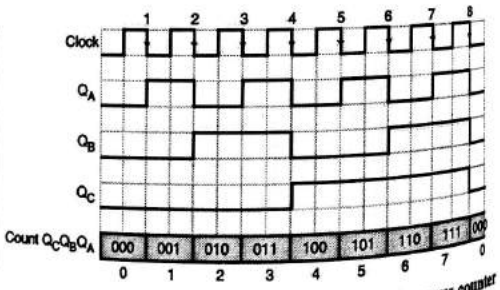
- Note that $Q_B = Q_A = 1$. Hence output of AND gate = 1 and $J_C = K_C = 1$, at the instant of application of 4th negative edge of the clock.
- Hence on application of this clock pulse, FF-C will toggle and Q_C changes from 0 to 1.
- FF-A toggles as usual and Q_A becomes 0.
- Since Q_A was equal to 1 earlier, FF-B will also toggle to make $Q_B = 0$.

$$\therefore\ Q_C Q_B Q_A = 1\,0\,0 \qquad \text{...After the 4th clock pulse}$$

- Thus the counting progresses.
- After the 7th clock pulse the output is 111 and after the 8th clock pulse, all the flip-flops toggle and change their outputs to 0. Hence $Q_C Q_B Q_A = 0\,0\,0$ after the 8th pulse and the operation repeats.
- Table 8.8.2 summarizes operation of the three bit synchronous counter.

Table 8.8.2 : Summary of operation of a 3-bit synchronous counter

Clock	Q_C	Q_B	Q_A
0	0	0	0
1st (↓)	0	0	1
2nd (↓)	0	1	0
3rd (↓)	0	1	1
4th (↓)	1	0	0
5th (↓)	1	0	1
6th (↓)	1	1	0
7th (↓)	1	1	1



(C-818) **Fig. 8.8.4 : Timing diagram for a 3-bit synchronous counter**

Timing diagram :

- Timing diagram for a 3-bit synchronous counter is shown in Fig. 8.8.4. The number of states through which this counter progresses is 8 namely $Q_C Q_B Q_A$ = 000, 001, 010, 011, 100, 101, 110, 111. The maximum count is $(111)_2$ or $(7)_{10}$.

Note that the waveforms of synchronous counter are exactly same as those of an asynchronous counter.

8.8.3 Design of the 3 Bit Synchronous Counter :

Let us now design a 3 bit synchronous counter first using the T flip flops and then using JK flip flops.

Design using T flip flops :

Steps to be followed :

Step 1 : Decide the number of flip flops.
Step 2 : Write the excitation table of T flip flop.
Step 3 : Write the excitation table of the counter.
Step 4 : From the circuit excitation table write K-maps and obtain simplified equations.
Step 5 : Draw the logic diagram.

Step 1 : Decide number of FFs : A 3 bit counter goes through 8 states. So it needs three flip flops.

Step 2 : Excitation table of T FFs : Table 8.8.3(a) shown the excitation table of T FF.

Table 8.8.3(a) : Excitation table of a T FF

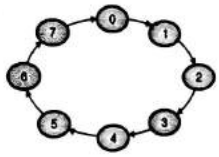
Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Step 3 : State diagram and circuit excitation table :

- Count sequence for a 3 bit up counter is given in Table 8.8.3(b) and Fig. 8.8.5 shows the corresponding state diagram.

Table 8.8.3(b)

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



(C-819) **Fig. 8.8.5 : State diagram**

Table 8.8.3(c) shows the circuit excitation table.

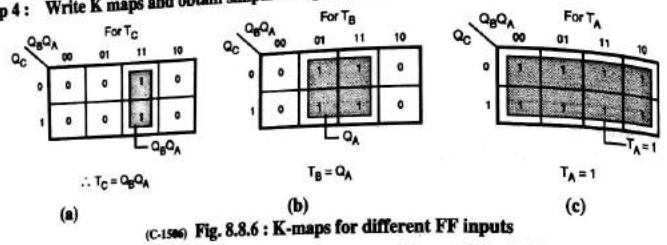
Table 8.8.3(c) : Circuit excitation table

Present state			Next state			Flip flop input		
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Scanned by CamScanner

- Table 8.8.3(c) has been written by referring to the present and next stat of an output and the required value to input is written as per the excitation table of T FF.
- For example consider the shaded columns of Table 8.8.3(c). i.e. Q_C, Q_{C+1} and T_C
- Consider the first row. $Q_C = 0$, $Q_{C+1} = 0$. So as per the excitation table of a T FF T_C should be 0. Similarly all other entries are made.

Step 4 : Write K maps and obtain simplified equations :

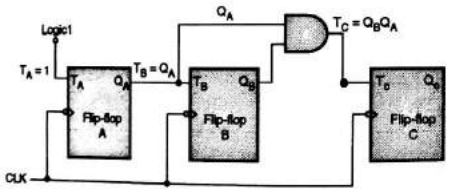


(C-1506) **Fig. 8.8.6 : K-maps for different FF inputs**

Refer Figs. 8.8.6(a), (b), (c) for the K-maps corresponding to all the FF inputs. The simplified equations for T_A, T_B and T_C also are shown.

Step 5 : Draw the logic diagram :

By using the simplified equations for T_A, T_B and T_C we can draw the logic diagram for the 3 bit synchronous shown in Fig. 8.8.7.



(C-1507) **Fig. 8.8.7 : Logic diagram of a 3-bit synchronous counter**

Design using JK flip flops :

Step 1 : Number of flip flops :

For a 3 bit counter, we need 3 flip-flops.

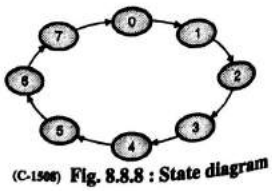
Step 2 : Excitation table of JK FF :

Table excitation table of JK FF

Table 8.8.4 : Excitation table of JK FF

Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Step 3 : State diagram and circuit excitation table

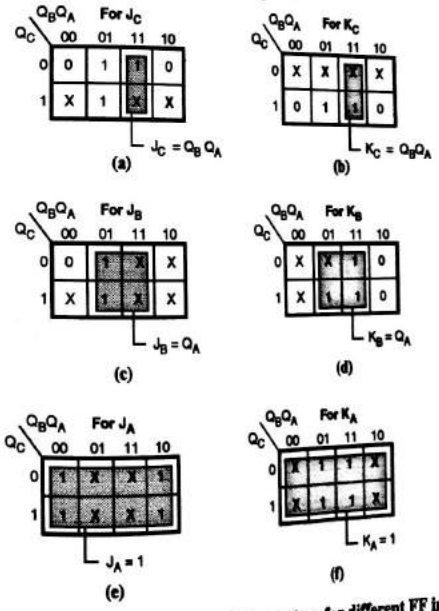


(C-1508) **Fig. 8.8.8 : State diagram**

Table 8.8.5 : Circuit excitation table

Present state			Next state			Flip flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Step 4 : K maps and simplified expressions for all FF inputs :



(C-1509) **Fig. 8.8.9 : K-maps and simplified equations for different FF inputs**

Thus the simplified equations are :

$J_C = Q_B Q_A$ $\quad$ $K_C = Q_B Q_A$

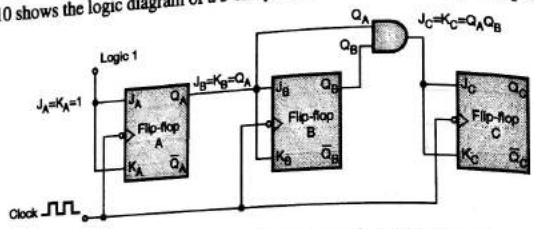
$J_B = Q_A$ $\quad$ $K_B = Q_A$

$J_A = 1$ $\quad$ $K_A = 1$

Scanned by CamScanner

Step 5 : Logic diagram :

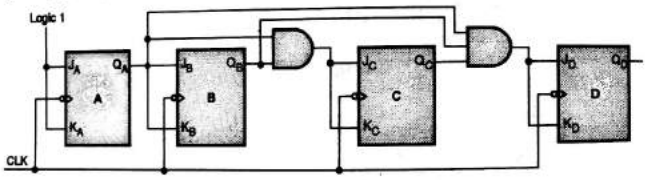
- Fig. 8.8.10 shows the logic diagram of a 3 bit synchronous counter using JK flip-flops.



(C-1510) **Fig. 8.8.10 : 3 bit synchronous counter using JK FFs**

8.8.4 Four Bit Synchronous Up Counter :

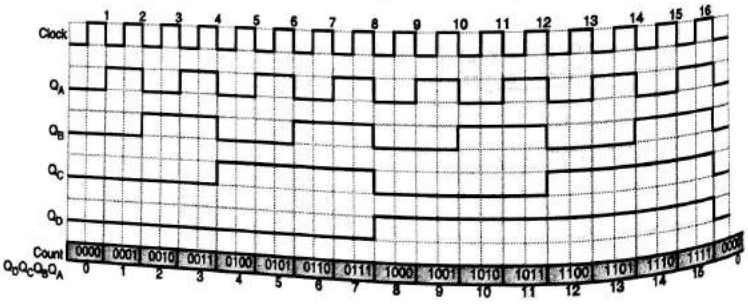
- All the concepts of a synchronous counter are extended to design a 4-bit synchronous counter shown in Fig. 8.8.11.
- Note that J_D and K_D of the FF-D are connected to the output of an AND gate and the inputs of this AND gate come from the outputs of the three preceding FFs.
- Operating principle of this counter is same as that of the 3-bit counter discussed earlier.



(C-1391) **Fig. 8.8.11 : A four bit synchronous counter**

Timing diagram :

- The timing diagram of a four bit synchronous counter is as shown in Fig. 8.8.12.
- Note that the principle of frequency division is applicable to the synchronous counters as well.



(C-1392) **Fig. 8.8.12 : Timing diagram of a 4-bit synchronous counter**

8.9 Modulo – N Synchronous Counters :

University Questions MU : May 04

Q. 1 Design a mod of synchronous counter using JK flip flop. **(May 04, 10 Marks)**

- We have already discussed the MOD-N asynchronous counters. Now let us discuss the Modulo-N synchronous counters.
- The steps to be followed to design a MOD-N synchronous counter are as follows.

Steps to be followed :

Step 1 : Decide the number of FFs and type of FF to be used.
Step 2 : Write the circuit excitation table.
Step 3 : From the circuit excitation table write down the K-maps and obtain simplified expressions for the outputs.
Step 4 : Draw the logic diagram and timing diagram.

8.9.1 Synchronous Decade Counter :

For the design of a synchronous decade counter follow the steps given below :

Step 1 : Write the excitation table for T FF and circuit excitation table :

- Excitation table for T FF is shown in Table 8.9.1.

Table 8.9.1 : Excitation table for T flip-flop

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

- The circuit excitation table is shown in Table 8.9.2.

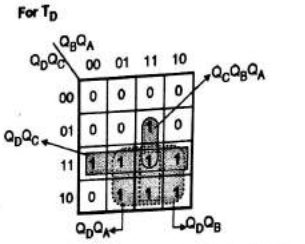
Table 8.9.2 : Circuit excitation table

Present state				Next state				Flip-flop inputs			
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	0	0	0	1	0	1	0
1	0	1	1	0	0	0	0	1	0	1	1
1	1	0	0	0	0	0	0	1	1	0	0
1	1	0	1	0	0	0	0	1	1	0	1
1	1	1	0	0	0	0	0	1	1	1	0
1	1	1	1	0	0	0	0	1	1	1	1

Step 2 : K-maps and simplifications :

K-maps for T_D, T_C, T_B, T_A and their simplified expressions are given in Figs. 8.9.1(a), (b), (c) and (d).

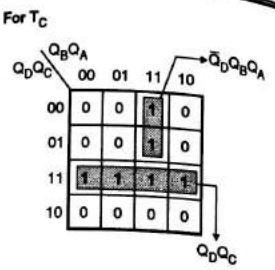
Scanned by CamScanner



$T_D = Q_C Q_B Q_A + Q_D Q_C + Q_D Q_B + Q_D Q_A$

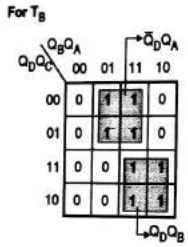
$\therefore T_D = Q_C Q_B Q_A + Q_D (Q_C + Q_B + Q_A)$

(a) K map for T_D



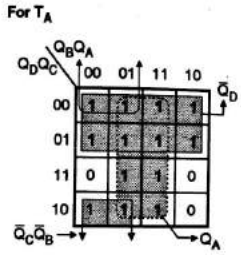
$T_C = \bar{Q}_D Q_B Q_A + Q_D Q_C$

(b) K map for T_C



$\therefore T_B = Q_D Q_B + \bar{Q}_D Q_A$

(c) K map for T_B



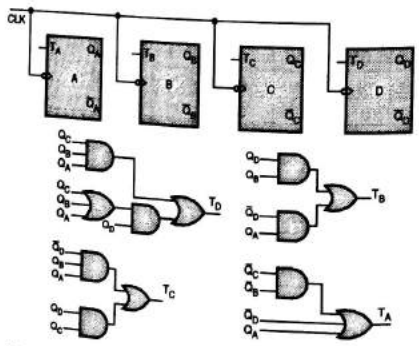
$T_A = \bar{Q}_D + \bar{Q}_C \bar{Q}_B + Q_A$

(d) K map for T_A

(C-847) **Fig. 8.9.1**

Step 3 : Draw the logic diagram :

Fig. 8.9.2 shows the logic diagram of a synchronous decade counter.



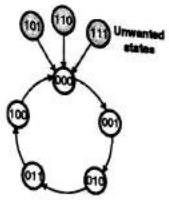
(C-848) **Fig. 8.9.2 : Logic diagram of a synchronous decade counter**

Ex. 8.9.1 : Design a synchronous counter for the sequence shown in Fig. P. 8.9.1(a).

May 15, Dec. 15, 10 Marks

Table P. 8.9.1(a) : Desired sequence

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0



(C-849) **Fig. P. 8.9.1(a) : State diagram**

Soln. :

Step 1 : Determine the desired number of FFs :

- From the given sequence the number of FFs is equal to 3. This is a MOD-5 synchronous counter since the number of states is 5.
- The state diagram is shown in Fig. P. 8.9.1(a) which shows that 101, 110, 111 are unwanted states.

Excitation table of JK FF :

Table P. 8.9.1(b) exhibits the excitation table of a JK FF.

Circuit excitation table :

The circuit excitation table is as shown in Table P. 8.9.1(c).

Step 2 : Write the excitation table and state table :

- The type of FF used is JK flip-flop. The excitation table for a JK FF is as shown in Table P. 8.9.1(b).
- We have already seen how to write the excitation table for JK FF.

Table P. 8.9.1(b) : Excitation table of JK FF

Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table P. 8.9.1(c) : Circuit excitation table

Present state			Next state			Flip-flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	0	0	0	×	1	0	×	0	×
1	0	1	0	0	0	×	1	0	×	×	1
1	1	0	0	0	0	×	1	×	1	0	×
1	1	1	0	0	0	×	1	×	1	×	1

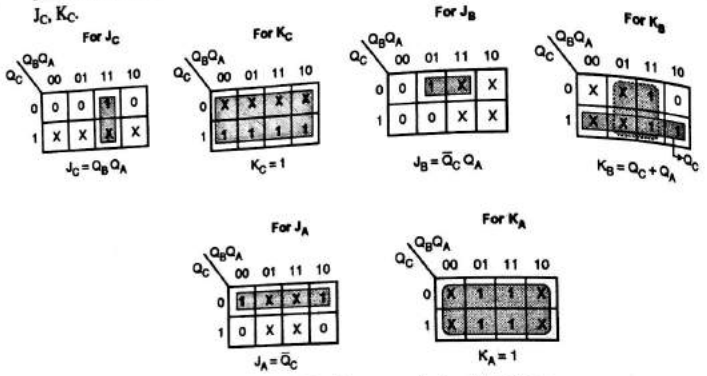
- Refer to the shaded portion of the circuit excitation table. This is nothing but the excitation table of FF-C. The J_C and K_C values have been decided based on Q_C and Q_{C+1}.
- Similarly the entries for J_B and K_B are based on Q_B and Q_{B+1} whereas those for J_A and K_A are based on Q_A and Q_{A+1}.

Step 3 : K-maps and simplifications :

- K-maps for the J and K inputs of all the FFs and the corresponding simplified equations are shown in Fig. P. 8.9.1(b).

Scanned by CamScanner

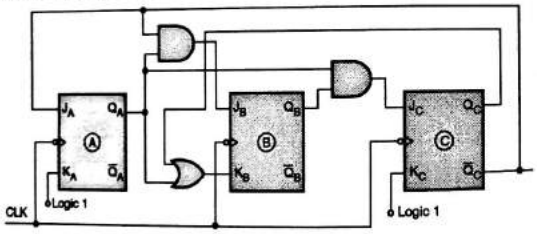
- Note that the inputs to this combinational circuit are Q_A, Q_B, Q_C and outputs are J_A, K_A through J_C, K_C.



(C-850) **Fig. P. 8.9.1(b) : K-maps and simplifications**

Step 4 : Logic diagram :

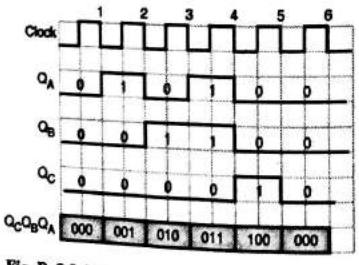
The logic diagram of MOD-5 synchronous counter is shown in Fig. P. 8.9.1(c).



(C-851) **Fig. P. 8.9.1(c) : Logic diagram of MOD-5 synchronous counter**

Step 5 : Draw the timing diagram :

The timing diagram of MOD-5 synchronous counter is shown in Fig. P. 8.9.1(d).



(C-852) **Fig. P. 8.9.1(d) : Timing diagram of MOD-5 counter**

Ex. 8.9.2 : Design MOD 13 synchronous counter using T flip-flop.

Soln. : For the design of a synchronous decade counter follow the steps given below :

Step 1 : Write the excitation table for T FF and circuit excitation table :

- Excitation table for T FF is shown in Table P. 8.9.2.

Table P. 8.9.2 : Excitation table for T flip-flop

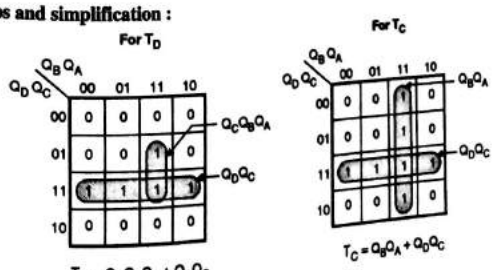
Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

- The circuit excitation table is shown in Table P. 8.9.2(a).

Table P. 8.9.2(a) : Circuit excitation table

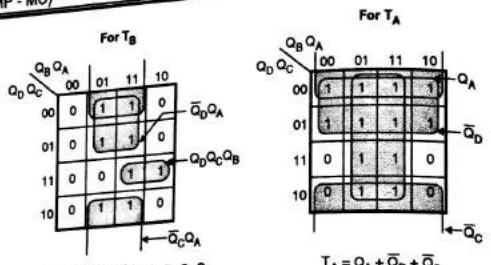
Present state				Next state				Flip-flop inputs			
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	1	0	1	1	0	0	0	1
1	0	1	1	1	1	0	0	0	1	1	1
1	1	0	0	0	0	0	0	1	1	0	0
1	1	0	1	0	0	0	0	1	1	0	1
1	1	1	0	0	0	0	0	1	1	1	0
1	1	1	1	0	0	0	0	1	1	1	1

Step 2 : K-maps and simplification :



(C-3454) **Fig. P. 8.9.2 (Contd..)**

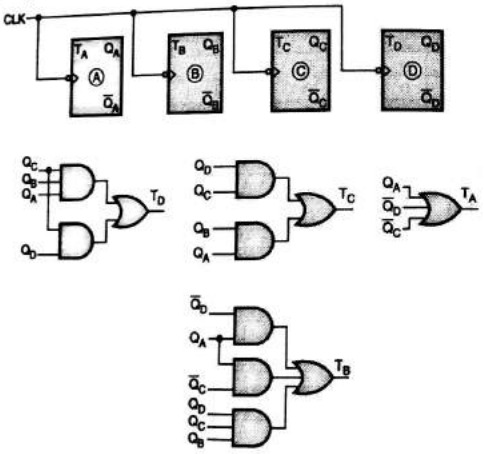
Scanned by CamScanner



$T_B = \overline{Q}_D Q_A + \overline{Q}_C Q_A + Q_D Q_C Q_B$ $\quad$ $T_A = Q_A + \overline{Q}_D + \overline{Q}_C$

(C-3454) **Fig. P. 8.9.2**

Step 3 : Draw the logic diagram :



(C-3455) **Fig. P. 8.9.2(a)**

Ex. 8.9.3 : Design MOD-6 synchronous counter and explain its operation. Dec. 08, May 09, Dec. 10, Dec. 11, 10 Marks

Soln. :

Step 1 : Decide the number of FFs : Since the number of state is 6 we need to use 3 flip-flops

Step 2 : Excitation tables : Let us use T flip flop

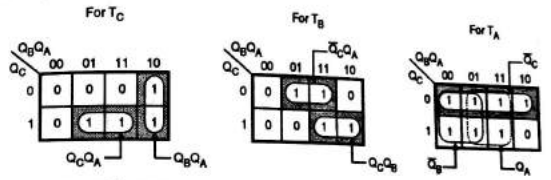
Table P. 8.9.3(a) : Excitation table of a T flip-flop

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table P. 8.9.3(b) : Circuit excitation table :

Present state			Next state			Flip-flop input		
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	0	1
1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	1	1	1

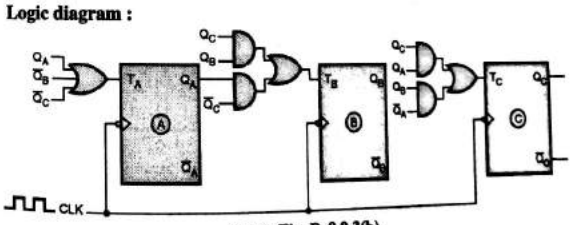
Step 3 : K-map and simplification :



$T_C = Q_B \overline{Q}_A + Q_C Q_A$ $\quad$ $T_B = Q_C Q_B + \overline{Q}_C Q_A$ $\quad$ $T_A = Q_A + \overline{Q}_B + \overline{Q}_C$

(C-3280) **Fig. P. 8.9.3(a)**

Step 4 : Logic diagram :



(C-3281) **Fig. P. 8.9.3(b)**

Ex. 8.9.4 : Design a mod-6 synchronous counter using JK FF. May 12, 10 Marks

Soln. :

Step 1 : Decide number of flip flops :

Since the number of states is 6 we need to use 3 flip-flops

Step 2 : Excitation tables :

Table P. 8.9.4(a) : Excitation table of JK FF

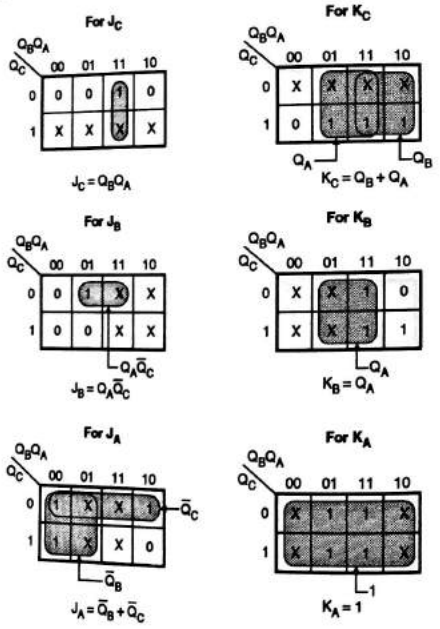
Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Scanned by CamScanner

Table P. 8.9.4(b) : Circuit excitation table

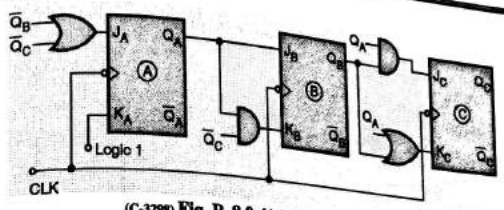
Present state			Next state			Flip-Flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	0	0	0	X	1	X	1	X	1

Step 3 : K-map simplification :



(C-3297) **Fig. P. 8.9.4 : K-maps**

Step 4 : Logic diagram :



(C-3298) **Fig. P. 8.9.4(a) : Implementation**

8.10 UP / DOWN Synchronous Counter :

- The operating principle of a synchronous up down counter is same as that of the up/down ripple counters. But the design procedure and realization are different.
- The mode control (M) is used for selecting the mode of operation.
- The steps to be followed for the design are as follows :

Steps to be followed :

Step 1 : Write the circuit excitation table.
Step 2 : Write K-maps and obtain the simplified expressions.
Step 3 : Draw the logic diagram.

8.10.1 3-bit Up/Down Synchronous Counter :

The number of FFs to be used is three. We will use three toggle flip-flops. Let upcounting takes place with M = 0 and down counting takes place for M = 1.

Step 1 : Write the circuit excitation table :

The circuit excitation table is shown in Table 8.10.1.

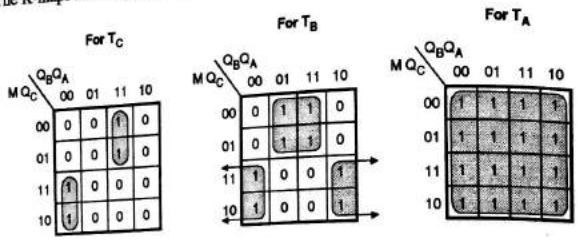
Table 8.10.1 : Excitation table for a 3-bit up/down synchronous counter

Mode control	Present state			Next state			Flip-flop inputs			
M	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A	
0	0	0	0	0	0	1	0	0	1	Up counting
0	0	0	1	0	1	0	0	1	1	
0	0	1	0	0	1	1	0	0	1	
0	0	1	1	1	0	0	1	1	1	
0	1	0	0	1	0	1	0	0	1	
0	1	0	1	1	1	0	0	1	1	
0	1	1	0	1	1	1	0	0	1	
0	1	1	1	0	0	0	1	1	1	
1	0	0	0	1	1	1	1	1	1	Down counting
1	0	0	1	0	0	0	0	0	1	
1	0	1	0	0	0	1	0	1	1	
1	0	1	1	0	1	0	0	0	1	
1	1	0	0	0	1	1	1	1	1	
1	1	0	1	1	0	0	0	0	1	
1	1	1	0	1	0	1	0	1	1	
1	1	1	1	1	1	0	0	0	1	

Scanned by CamScanner

Step 2 : K-maps and simplified equations :

The K-maps and simplified equations for T_C, T_B and T_A are shown in Fig. 8.10.1.

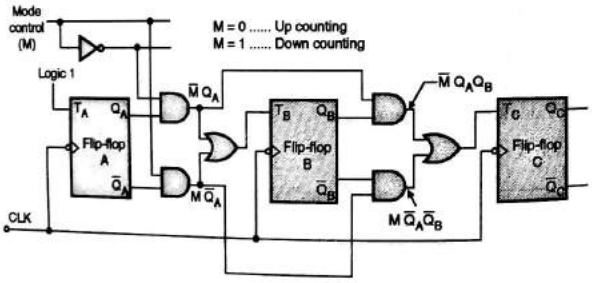


$\therefore T_C = \overline{M} Q_B Q_A + M \overline{Q}_B \overline{Q}_A$ $\quad T_B = \overline{M} Q_A + M \overline{Q}_A = M \oplus Q_A$ $\quad \therefore T_A = 1$

(C-855) **Fig. 8.10.1 : K-maps and simplified expressions**

Step 3 : Draw the logic diagram :

The logic diagram for a 3-bit synchronous up down counter is shown in Fig. 8.10.2.



(C-856) **Fig. 8.10.2 : Logic diagram of a 3-bit synchronous up down counter**

8.10.2 Advantages of Synchronous Counter :

1. All the FFs receive the clock signal simultaneously i.e. at the same instant, the propagation delay problem is reduced to a great extent. Recall that the propagation delay is one of the problems of the ripple counter.
2. The total propagation delay between the instant of application of clock edge and the instant at which the MSB output changes is equal to sum of propagation delay of one FF and that of one AND gate.

$\therefore$ Total propagation delay = t_d of one FF + t_d of one gate

3. This is much less than the propagation delay of an asynchronous (ripple) counter.

Due to reduced propagation delay, the synchronous counters can operate at a much higher clock frequency than the asynchronous counters.

8.10.3 Comparison of Synchronous and Asynchronous Counters : MU : Dec. 08

University Questions

Q. 1 Write short note on : Asynchronous versus synchronous counter. **(Dec. 08, 5 Marks)**

Comparison of synchronous and asynchronous counters is given in Table 8.10.2.

Table 8.10.2 : Comparison of synchronous and asynchronous counters

Sr. No.	Parameter	Asynchronous counter	Synchronous counter
1.	Circuit complexity	Logic circuit is simple	With increase in number of states, the logic circuit becomes complicated.
2.	Connection pattern	Output of the preceding FF, is connected to clock of the next FF.	There is no connection between output of preceding FF and CLK of next one.
3.	Clock input	All the FFs are not clocked simultaneously.	All FFs receive clock signal simultaneously.
4.	Propagation delay	P.D. = n × (t_d) where n is number of FFs and t_d is p.d. per FF.	P.D. = $(t_d)_{FF} + (t_d)_{gate}$. It is much shorter than that of asynchronous counter.
5.	Maximum frequency of operation	Low because of the long propagation delay.	High due to shorter propagation delay.

Ex. 8.10.1 : Design a mod 6 synchronous up/down counter using SR flip flop.

Dec. 02. 10 Marks. May 05. 8 Marks

Soln. :

Step 1 : Determine the desired number of FFs :

As it is a mod-6 synchronous counter, the number of states is 6. So the number of FFs required is equal to 3.

Scanned by CamScanner

Step 2 : Write the excitation table and state table :

Excitation table of SR FF :

Present state Q	Next state Q_{n+1}	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

Circuit excitation table :

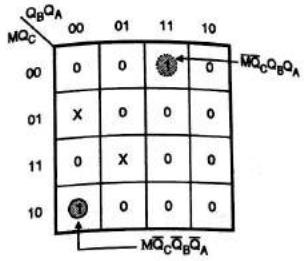
	Mode control	Present state			Next state			Flip-flop inputs					
	M	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	S_C	R_C	S_B	R_B	S_A	R_A
Up	0	0	0	0	0	0	1	0	×	0	×	1	0
	0	0	0	1	0	1	0	0	×	1	0	0	1
	0	0	1	0	0	1	1	0	×	×	0	1	0
	0	0	1	1	1	0	0	1	0	0	1	0	1
	0	1	0	0	1	0	1	×	0	0	×	1	0
	0	1	0	1	0	0	0	0	1	0	×	0	1
Invalid states	0	1	1	0	0	0	0	0	1	0	1	0	×
	0	1	1	1	0	0	0	0	1	0	1	0	1

Circuit excitation table :

	Mode control	Present state			Next state			Flip-flop inputs					
	M	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	S_C	R_C	S_B	R_B	S_A	R_A
Down	1	0	0	0	1	0	1	1	0	0	×	1	0
	1	0	0	1	0	0	0	0	×	0	×	0	1
	1	0	1	0	0	0	1	0	×	0	1	1	0
	1	0	1	1	0	1	0	0	×	×	0	0	1
	1	1	0	0	0	1	1	0	1	1	0	1	0
	1	1	0	1	1	0	0	×	0	0	×	0	1
Invalid states	1	1	1	0	0	0	0	0	1	0	1	0	×
	1	1	1	1	0	0	0	0	1	0	1	0	1

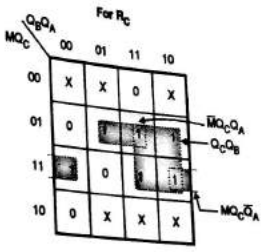
Step 3 : K-maps and simplification :

For S_C



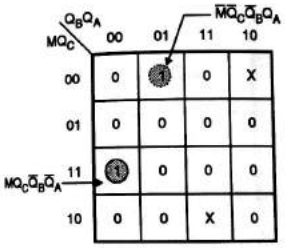
$S_C = \overline{M}\overline{Q}_C Q_B Q_A + M\overline{Q}_C\overline{Q}_B\overline{Q}_A$

For R_C



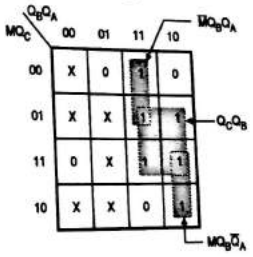
$R_C = Q_C Q_B + Q_C(\overline{M}Q_A + M\overline{Q}_A)$

For S_B



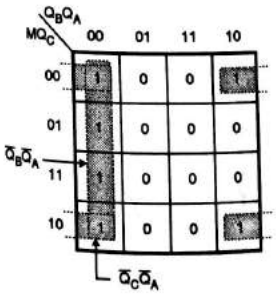
$S_B = MQ_C\overline{Q}_B\overline{Q}_A + M\overline{Q}_C\overline{Q}_B Q_A$

For R_B



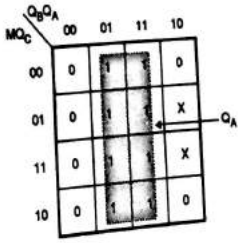
$R_B = \overline{M}Q_B Q_A + Q_C Q_B + MQ_B\overline{Q}_A$

For S_A



$S_A = \overline{Q}_A(\overline{Q}_B + \overline{Q}_C)$

For R_A

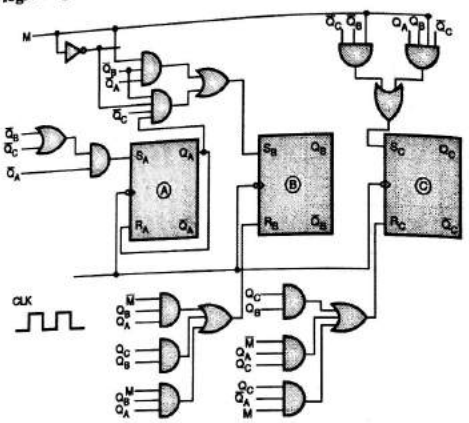


$R_A = Q_A$

(C-3716)

Scanned by CamScanner

Step 4 : Draw the logic diagram :



(C-3717) **Fig. P. 8.10.1**

8.11 Lock Out Condition :

MU : Dec. 03, May 04, Dec. 05, Dec. 07

University Questions

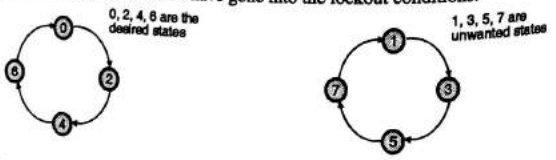
Q. 1 What is lock out in synchronous counters ? Does the above counter go in lock out ? How can it be avoided **(Dec. 03, 6 Marks)**

Q. 2 What is lockout state? Does the counter you have designed go in a locked out state ? **(May 04, 10 Marks)**

Q. 3 What is lockout in synchronous counters? **(Dec. 05, 3 Marks)**

Q. 4 If the counter enters any unused state will it go to lockout condition. Justify. **(Dec. 07, 2 Marks)**

- A counter is supposed to follow the sequence of only the desired states as shown in Fig. 8.11.1(a).
- If it enters into an unused or unwanted state, then it is expected to return back to a desired state.
- Instead if the next state of an unwanted state is again an unwanted state as shown in Fig. 8.11.1(b) then the counter is said to have gone into the lockout conditions.



(a) State diagram showing the desired sequence

(b) State diagram showing the lockout condition

(C-831) **Fig. 8.11.1**

8.11.1 Bushless Circuit :

MU : Dec. 03, Dec. 05

University Questions

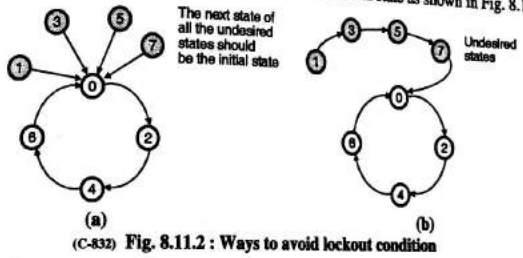
Q. 1 What is lock out in synchronous counters ? Does the above counter go in lock out ? How can it be avoided **(Dec. 03, 6 Marks)**

Q. 2 What is lockout in synchronous counters ? How can it be avoided ? **(Dec. 05, 3 Marks)**

The sequential circuit which enters into the lockout condition is called as the bushless circuit.

How to avoid lockout situation ?

- In order to avoid the lockout condition, we have to use some additional logic circuit to ensure that the next state of the counter is its initial state, if it enters into an undesired state.
- Thus the next state of each unwanted state should be the initial state as shown in Fig. 8.11.2(a).



(C-832) **Fig. 8.11.2 : Ways to avoid lockout condition**

- However it is not necessary to terminate all the undesired states into the initial state as shown in Fig. 8.11.2(a). It will be sufficient to terminate only one unused state into the initial state as shown in Fig. 8.11.2(b).
- If the circuit enters into say state "3" then its next states will be "5" and "7". As "7" has next state of "0" which is the desired state, the lockout condition is avoided.

Ex. 8.11.1 : By using suitable flip-flops design a counter to go through the following states.

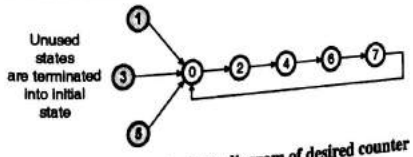
0 – 2 – 4 – 6 – 7 – 0

Avoid the lockout condition.

Soln. :

Step 1 : Draw the state diagram :

- The state diagram is shown in Fig. P. 8.11.1(a). It shows that the next state of all the unused states (1, 3 and 5) is forced to be the initial state (0).
- This is essential in order to avoid the lockout condition.



(C-833) **Fig. P. 8.11.1(a) : State diagram of desired counter**

Scanned by CamScanner

Step 2 : Number of flip-flops and type of flip-flop :

- Since the highest state is 7 i.e. 111 we need to use three flip-flops. Let us use JK flip-flops.

Step 3 : Write the state table :

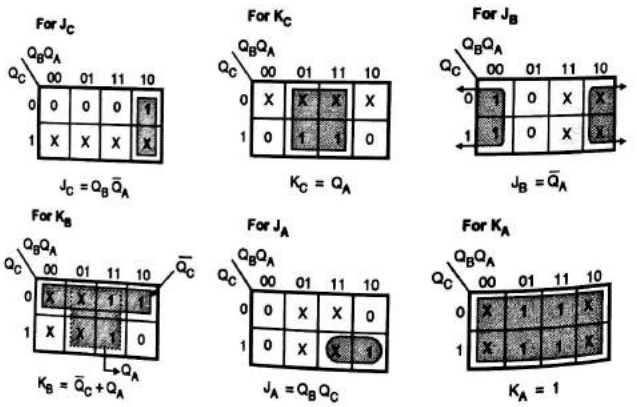
- Table P. 8.11.1(a) shows the state table for the required counter.

Table P. 8.11.1(a)

Present state			Next state			Flip-flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	1	1	0	X	0	1	X	0	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Step 4 : K maps and simplification :

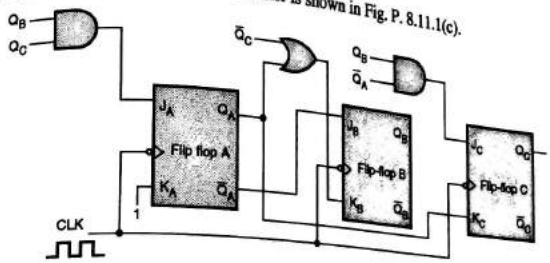
Fig. P. 8.11.1(b) illustrates the K-maps for all the FF inputs and the corresponding simplified expressions.



(C-834) **Fig. P. 8.11.1(b) : K-maps and simplification**

Step 5 : Draw the logic diagram :

The logic diagram for the required counter is shown in Fig. P. 8.11.1(c).



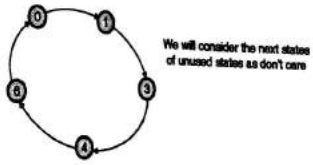
(C-835) **Fig. P. 8.11.1(c) : Logic circuit of the required counter**

Ex. 8.11.2 : By using suitable FFs design a counter to go through the states 0-1-3-4-6-0. Draw the logic diagram. Examine the action of counter for the unused states.

Soln. :

Step 1 : Draw the state diagram :

The state diagram is as shown in Fig. P. 8.11.2(a). Let us use three T flip-flops.



(C-836) **Fig. P. 8.11.2(a) : State diagram**

Step 2 : Write the state table :

- Assuming the next state of unused states to be don't care conditions the state table is written as shown in Table P. 8.11.2(a).

Table P. 8.11.2(a) : State table

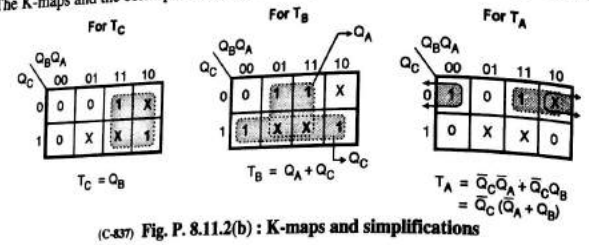
Present states			Next states			Flip-flop inputs		
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	0	X	X	X	X	X	X
0	1	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1	0
1	0	1	X	X	X	X	X	X
1	1	0	0	0	0	1	1	0
1	1	1	X	X	X	X	X	X

Next state of the unused state is considered as don't care

Scanned by CamScanner

Step 3 : K-maps and simplifications :

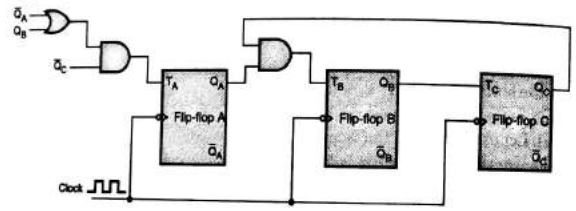
The K-maps and the corresponding simplified expressions are as shown in Fig. P. 8.11.2(b).



(C-837) **Fig. P. 8.11.2(b) : K-maps and simplifications**

Step 4 : Draw the logic diagram :

The logic diagram of the required counter is shown in Fig. P. 8.11.2(c).



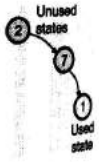
(C-838) **Fig. P. 8.11.2(c) : Logic circuit of the required counter**

Step 5 : Examine the counter action for the unused states :

For state 2 : $Q_C = 0$, $Q_B = 1$ and $Q_A = 0$.

Table P. 8.11.2(b) (C-839)

Present state			FF inputs			Next state		
Q_C	Q_B	Q_A	$T_C = Q_B$	$T_B = Q_A + Q_C$	$T_A = \overline{Q}_C(\overline{Q}_A + Q_B)$	Q_{C+1}	Q_{B+1}	Q_{A+1}
0	1	0	1	0	1	1	1	1
1	1	1	1	1	0	0	0	1

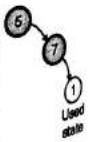


001 i.e. 1 is a used state so no lockout.

For state 101 i.e. 5 : Refer Table P. 8.11.2(c) to know the behaviour of the counter for state 5.

Table P. 8.11.2(c)

Present state			Flip flop			Next state		
Q_C	Q_B	Q_A	$T_C = Q_B$	$T_B = Q_A + Q_C$	$T_A = \overline{Q}_C(\overline{Q}_A + Q_B)$	Q_{C+1}	Q_{B+1}	Q_{A+1}
1	0	1	0	1	0	1	1	1



In Table P. 8.11.2(b) we have already proved that the counter returns to state "1" from state "7", which is used state.

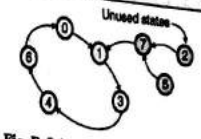
(C-840)

For state 111 i.e. 7 : In Table P. 8.11.2(b) we have already proved that the counter returns to state 1 from "7".

Conclusion : The counter returns to a used state 1 if by chance it enters into one of the unused states 2, 5 or 7. Hence lockout condition is automatically avoided.

Step 6 : Draw the modified state diagram :

The modified state diagram showing the used as well as the unused states is shown in Fig. P. 8.11.2(d).



(C-841) **Fig. P. 8.11.2(d) : State diagram showing used and unused states**

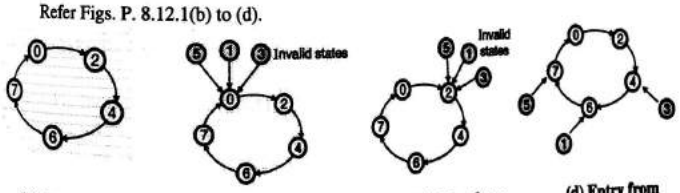
8.12 Bush Diagram :

- In the previous section we have discussed the meaning of lock-out condition.
- The lock-out condition can be avoided by taking a precaution called **"bushing"**. Bushing means adding branches in the state diagram.
- The invalid states of a counter are branched in the bush diagram in such a way that even if the counter enters into an invalid state, it will return to one of its valid states after one, two or three clock cycles.
- The concept of bushing can be well understood by referring to the following example.

Ex. 8.12.1 : For the given state diagram in Fig. P. 8.12.1(a) of a counter draw the bush diagrams required to bring the counter from invalid to a valid state in one, two or three clock cycles.

Soln. :

Refer Figs. P. 8.12.1(b) to (d).



(a) Given state diagram **(b) Entry from invalid to valid state after only one clock pulse** **(c) Entry from invalid to valid state after only one clock pulse** **(d) Entry from invalid to valid state after 1, 2 or 3 clock pulses**

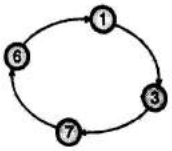
(C-842) **Fig. P. 8.12.1**

- Fig. P. 8.12.1(b) shows that the next state of each invalid state 1, 3 or 5 is the initial state 0. Hence even if the counter enters into an unused state, it will return to a valid state after only one clock cycle.
- Similarly for the state diagram shown in Fig. P. 8.12.1(c), it will take only one clock pulse to bring the counter into a used state if it enters into an unused one.

Scanned by CamScanner

- Refer the state diagram of Fig. P. 8.12.1(d). If the counter enters into state "7" then it will return to "0" after one clock pulse. If it is in the state "5" then it will return to 0 state after two clock pulses and if it is in the state "1" then it will return to "0" state after three clock pulses.

Ex. 8.12.2 : Design a synchronous counter from the state diagram shown in Fig. P. 8.12.2(a). Avoid lockout condition. Draw the bush diagram to avoid the lockout condition.

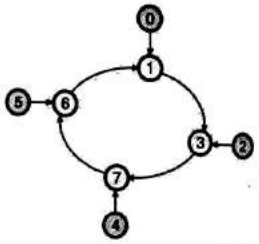


(C-843) **Fig. P. 8.12.2(a) : Given state diagram**

Soln. :

So as to avoid the lockout condition, we have to ensure that the counter should return back to a valid state, as soon as it enters into an unused state. The bush diagram is shown in Fig. P. 8.12.2(b).

0, 2, 4 and 5 are the unused states. They are forcibly terminated into 1, 3, 7 and 6 respectively.



(C-844) **Fig. P. 8.12.2(b) : Bush diagram**

Step 1 : Number of FFs and the type of FF :

- Let us use T-type FF.
- Since the highest state in the state diagram is 7, we have to use three T flip-flops.

Step 2 : Write the circuit excitation table :

- Let us obtain the circuit excitation table from the bush diagram of Fig. P. 8.12.2(b).
- The excitation table for a T flip-flop is shown in Table P. 8.12.2(a) and the circuit excitation table is shown in Table P. 8.12.2(b).

Table P. 8.12.2(a) : Excitation table for a toggle FF

Present state A	Next state A_{n+1}	T input
0	0	0
0	1	1
1	0	1
1	1	0

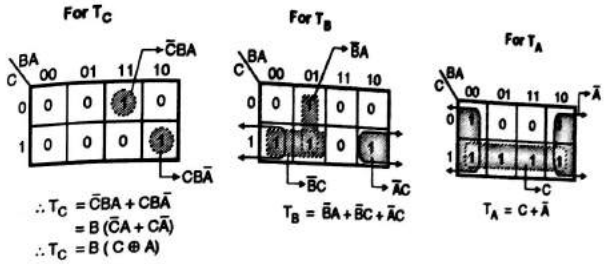
Table P. 8.12.2(b) : Circuit excitation table

Present state			Next state			FF inputs		
C	B	A	C_n +1	B_n +1	A_n +1	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	0	0	1	1	0	0	1
0	1	1	1	1	1	1	0	0
1	0	0	1	1	1	0	1	1
1	0	1	1	1	0	0	1	1
1	1	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1

Invalid states

Step 3 : Draw the K-maps and obtain simplified expressions :

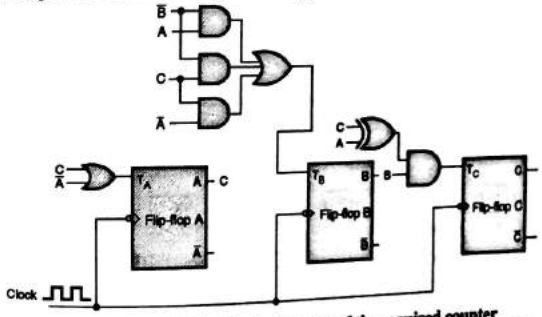
- Fig. P. 8.12.2(c) shows the K-maps for various T inputs. The simplified Boolean expressions also are given alongwith the K-maps.



(C-845) **Fig. P. 8.12.2(c) : K-maps and simplifications**

Step 4 : Logic diagram :

The logic diagram is as shown in Fig. P. 8.12.2(d).



(C-846) **Fig. P. 8.12.2(d) : Logic diagram of the required counter**

Ex. 8.12.3 : Design a synchronous mod 12 down counter using JK flip flops. Does the counter go in locked out state ? If it does, redesign the circuit so that no lock out state occurs. Sketch the waveforms. **May 04. 10 Marks**

Soln. :

Step 1 : Determine the desired number of FF's :

From the given sequence the number of FFs is equal to 4. This is MOD-12 synchronous counter since the number of state is 12.

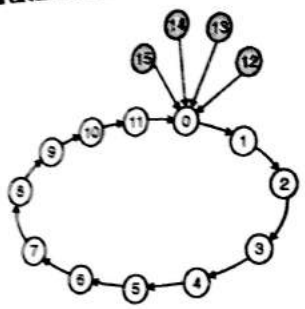
Step 2 : Write the excitation table : The type of FF used is J-K flip-flop.

Excitation table of J-K FF

Present state Q	Next state Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

Scanned by CamScanner

Step 3 : State diagram and circuit excitation table :

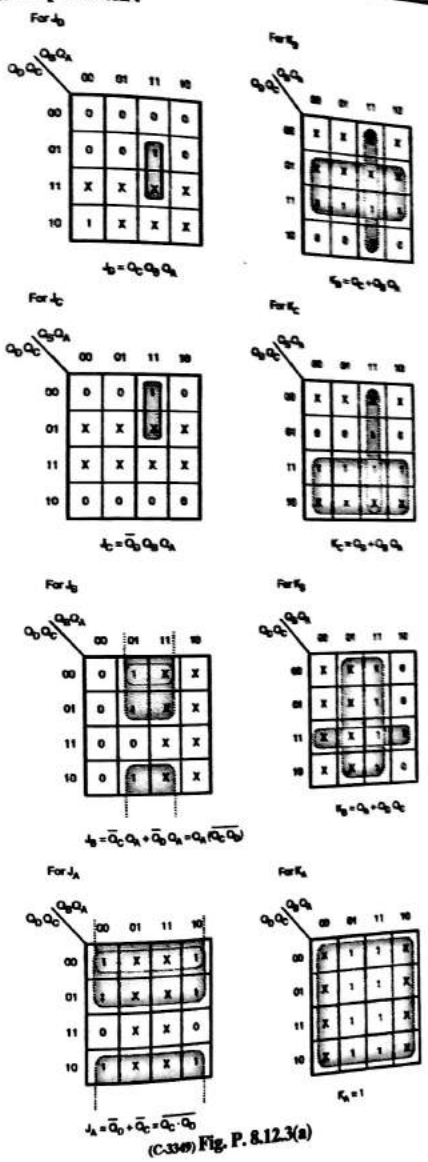


(C-3348) **Fig. P. 8.12.3 : State diagram**

Circuit excitation table :

Present state				Next state				Flip flop inputs							
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	1	0	0	×	0	×	1	×	×	1
0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	×
0	0	1	1	0	1	0	0	0	×	1	×	×	1	×	1
0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	×
0	1	0	1	0	1	1	0	0	×	×	0	1	×	×	1
0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	×
0	1	1	1	1	0	0	0	1	×	×	1	×	1	×	1
1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	×
1	0	0	1	1	0	1	0	×	0	0	×	1	×	×	1
1	0	1	0	1	0	1	1	×	0	0	×	×	0	1	×
1	0	1	1	0	0	0	0	×	1	0	×	×	1	×	1
1	1	0	0	0	0	0	0	×	1	×	1	0	×	0	×
1	1	0	1	0	0	0	0	×	1	×	1	0	×	×	1
1	1	1	0	0	0	0	0	×	1	×	1	×	1	0	×
1	1	1	1	0	0	0	0	×	1	×	1	×	1	×	1

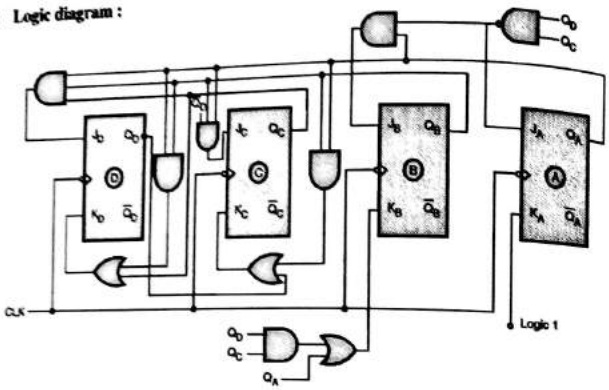
Step 4 : K-maps and simplification :



(C-3349) **Fig. P. 8.12.3(a)**

Scanned by CamScanner

Step 5 : Logic diagram :



(C-3350) **Fig. P. 8.12.3(b)**

8.13 Applications of Counters :

Some of the applications of counters are :

1. In digital clock.
2. In the frequency counters.
3. In time measurement.
4. In digital voltmeters.
5. In the counter type A to D converter.
6. In the digital triangular wave generator.
7. In the frequency divider circuits.

8.14 State Reduction and Assignments :

- In the analysis of a sequential circuit we always starts from a circuit diagram which has been given to us and derive the state table and then draw the state diagram as we did in the solved examples of previous section.
- In order to design a sequential circuit a set of specifications is given to us. We design the required circuit and finally draw the logic diagram of the designed logic circuit.
- In this section we are going to discuss some properties of synchronous sequential circuits which are useful in reducing the number of gates and flip-flops while implementing the designed circuits.

State reduction techniques :

- In order to minimize the number of components required to implement the designed circuit, we need to reduce the number of states the circuit is going through.
- There are two state reduction techniques :
  1. Row elimination method
  2. Implication table method.

8.14.1 Row Elimination Method :

- In the design of synchronous sequential circuits the first step is to draw the state diagram and then write the state table.

- Some of the states in the state diagram/state table could be redundant states.
- These redundant states can be avoided by the technique called state reduction technique.
- Thus the state reduction helps us to eliminate the redundant states which ultimately results in reducing the number of flip-flops and gates to be used in a sequential circuit.

Identification of a redundant state :

- If two states in a state diagram are equivalent, then one of them is treated as redundant and can be eliminated.
- Those two states are said to be equivalent if every same output and same next state is produced for every possible set of inputs.
- Reducing the number of states will not alter the input output relation of the circuit being designed.

8.14.2 Example on State Reduction :

- The specifications of a sequential circuit are given in the form of a state diagram.
- The steps to followed for state reduction are as follows.

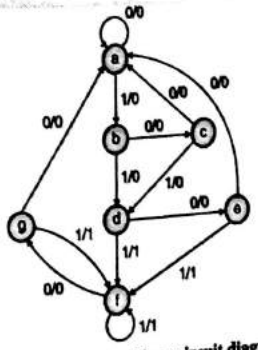
Steps to be followed :

Step 1 :	Derive the state table from the given state diagram.
Step 2 :	Find out the equivalent states.
Step 3 :	Eliminate one of the every pair of equivalent states and write a new state table.
Step 4 :	In the new state table, check for equivalent states and continue till all the redundant states are eliminated.
Step 5 :	Draw the state diagram with reduced number of states.

The concept of state reduction using the row elimination method has been illustrated in the following example.

Ex. 8.14.1 : How will you perform state reductions in Sequential Logic Design ? Reduce the number of states from the following State Diagram. Starting with initial state table, clearly justify the reduced state and reduced state table. Also draw the reduced state diagram.

SPPU : Dec. 07, 10 Marks



(C-680) **Fig. P. 8.14.1(a) : Given circuit diagram**

Scanned by CamScanner

Soln. :

Step 1 : Write the state table :

- Table P. 8.14.1(a) represents the state table derived from the given state diagram.

Table P. 8.14.1(a) : State table

Present state	Next state		Output Y	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	**a**	**f**	**0**	**1**
f	g	f	0	1
g	**a**	**f**	**0**	**1**

States "e" and "g" are equivalent states

Step 2 : Identify the equivalent states :

- Two states are said to be equivalent if, for all the possible input combinations, they give exactly the same output and send the circuit either to the same state or to an equivalent state.
- Apply this principle to the state table of Table P. 8.14.1(a). Look at the state table for two present states which go the same next states and have same output (Y) for both the input conditions (X = 0 and X = 1).
- The states "e" and "g" in Table P. 8.14.1(a) are found to be equivalent states, because both go to the next states "a" and "f" for X = 0 and 1 respectively with Y = 0 and 1 for X = 0 and 1 respectively.

Step 3 : Eliminate one of the equivalent states and write a new state table :

- As the states "e" and "g" are equivalent, we can remove one of them.
- Let us remove state "g" i.e. the last row containing the present state "g", and then replace "g" by "e" each time it appears in the next state column shown in the new state table of Table P. 8.14.1(b).

Table P. 8.14.1(b) : Modified state table with state reduction

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	**e**	**f**	**0**	**1**
e	a	f	0	1
f	**g e**	**f**	**0**	**1**

State "g" has been eliminated

After replacing "g" by "e" the states "d" and "f" have become equivalent.

"g" is replaced by "e"

- Note that in the last row, we have replaced "g" by "e".
- After doing so, the states "d" and "f" become equivalent states. So eliminate "f".

Step 4 : Replace "f" by "d" and prepare a new state table :

The reduced state table is as follows (Table P. 8.14.1(c)). Note that each "f" in the next state column is replaced by "d" and the last row for present state "f" has been removed completely.

Table P. 8.14.1(c) : Reduced state table

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	~~f~~ d	0	1
e	a	~~f~~ d	0	1

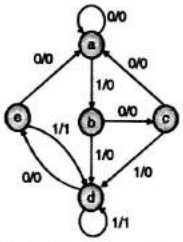
State "f" has been eliminated

"f" is replaced by "d"

In Table P. 8.14.1(c) all the 5 states are different from each other. Therefore the process of state reduction is complete and this table represents the reduced state table, after state reduction.

Step 5 : Draw the state diagram :

The state diagram for reduced state table is shown in Fig. P. 8.14.1(b).



(C-681) **Fig. P. 8.14.1(b) : Reduced state diagram**

Reduction in number of FFs due to state reduction :

- By using the state reduction technique, we reduce the number of states and this will reduce the number of flip flops required to be used to realize the circuit.
- If 2 flip flops are used then $2^2 = 4$ states are produced or with 3 flip flops $2^3 = 8$ states will be produced.
- Thus in general "m" flip-flops produce 2^m states and a reduction in number of states may (or may not) result in a reduction in number of flip-flops.

8.15 State Assignment :

- In the previous section we have learnt the state reduction technique to minimize the number of flip-flops. Now we will learn another technique which is used to reduce the cost of the combinational circuit.
- Such a technique is called as state assignment technique. It plays an important role in minimizing the number of gates used in the combinational circuit which drives the flip flops.
- Thus far we were using alphabets a, b, ... for representing states of a synchronous sequential circuits.
- But using the state assignment technique we can assign binary values to these states, in such a way that it will reduce the cost of the combinational circuit which drives the flip flops.
- The concept of state assignment will be clear after solving the following example.

Ex. 8.15.1 : For the given state table obtain the reduced state table using state assignment procedure.

Scanned by CamScanner

Table P. 8.15.1(a) : Given state table

Present state	Next state		Output (Y)	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Soln. :

Step 1 : Assign various possible binary values to different states :

- The assignment of various possible binary values to different states is demonstrated in Table P. 8.15.1(b).
- Note that the binary values of the state are not important as long as their sequence maintains the proper input output relationship.
- Hence assignment of any binary number to any state is satisfactory if we assign a unique number to each state.
- Table P. 8.15.1(b) shows three examples of possible binary assignment.

Table P. 8.15.1(b) : Three possible binary state assignments

State	Assignment 1	Assignment 2	Assignment 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

- Assignment-1 in Table P. 8.15.1(b) is a straightforward binary assignment, whereas the remaining two are done arbitrarily.

Step 2 : Obtain the reduced state table :

- Table P. 8.15.1(c) represents the reduced state table with binary assignment 1 substituted in place of letter symbols of Table P. 8.15.1(a).
- Such a state table with binary assignment is sometimes called as a **transition table.**
- Note that if we substitute binary assignment 2 or 3, then we would get a completely new transition table, with the same input output relationship.

Table P. 8.15.1(c) : Reduced state table or transition table

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
001 a	001	010	0	0
010 b	011	100	0	0
011 c	001	100	0	0
100 d	101	100	0	1
101 e	001	100	0	1

- This type of state table is used to derive the combinational part of the sequential circuit.
- The complexity of the combinational circuit obtained depends on the binary state assignment chosen.
- Therefore various procedures have been tried to select a particular binary assignment from so many possible ones.
- The most important criterion is that the selected binary assignment should result in a simpler combinational circuit.
- But it is surprising to know that there is no state assignment procedure that will guarantee a minimal-cost combinational circuit.

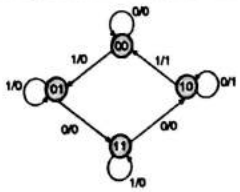
8.16 Design of Clocked Synchronous State Machine using State Diagram :

- Design of a clocked sequential circuits will start from the set of given specifications and it will end with drawing a logic diagram.
- The stepwise design procedure is as follows.

Step 1 : A state diagram or timing diagram or some other information is given, which describes the behaviour of the circuit that is to be designed.
Step 2 : Draw the state table.
Step 3 : The number of states can be reduced by state reduction methods.
Step 4 : Assign binary values to each state for the states in steps 2 and 3 using state assignment technique.
Step 5 : Determine the number of flip-flops required and assign a letter symbol to each one.
Step 6 : Decide the type of FF to be used.
Step 7 : Derive the circuit excitation table and output table from the state table.
Step 8 : Obtain the expressions for the circuit output and flip-flop inputs.
Step 9 : Draw the logic diagram.

The design procedure of clocked sequential circuits will be clear by solving the following example.

Ex. 8.16.1 : For the state diagram given in Fig. P. 8.16.1(a) draw the clocked sequential circuit using T flip-flops.



(C-485) **Fig. P. 8.16.1(a) : Given state diagram**

Soln. :

Step 1 : Write the state table :

The state table is written from the given state diagram as shown in Table P. 8.16.1(a).

Scanned by CamScanner

Table P. 8.16.1(a) : State table

Present state		Next state		Output	
A	B	X = 0	X = 1	X = 0	X = 1
		A_{n+1} B_{n+1}	A_{n+1} B_{n+1}		
0	0	0 0	0 1	0	0
0	1	1 1	0 1	0	0
1	0	1 0	0 0	1	1
1	1	1 0	1 1	0	0

Step 2 : Number of flip-flops :

- As seen from the state table, there are no equivalent states. So there won't be any reduction in the state diagram.
- The circuit goes through four states. Hence we need to use 2 flip-flops.

Step 3 : Write the circuit excitation table :

The circuit excitation table is shown in Table P. 8.16.1(b). The type of FF used is T type FF.

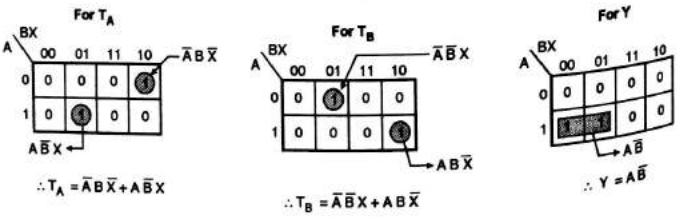
Table P. 8.16.1(b) : Circuit excitation table

Present state		Input	Next state		FF inputs		Output
A	B	X	A_{n+1}	B_{n+1}	T_A	T_B	Y
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	1	0	0	0
1	0	0	1	0	0	0	1
1	0	1	0	0	1	0	1
1	1	0	1	0	0	1	0
1	1	1	1	1	0	0	0

- The shaded portion of the circuit excitation table corresponds to the excitation table of a T FF.

Step 4 : K maps and simplifications :

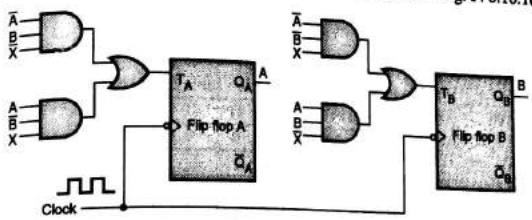
- Fig. P. 8.16.1(b) shows the K-maps and corresponding simplified expressions for T_A T_B i.e. inputs of the two T flip-flops and the Y output.



(C-686) **Fig. P. 8.16.1(b) : K-maps and simplifications**

Step 5 : Draw the logic diagram :

The logic diagram for required clocked sequential circuit is shown in Fig. P. 8.16.1(c).



(C-687) **Fig. P. 8.16.1(c)**

8.16.1 Design using Unused States :

- Sometimes a sequential circuit may use less number of states than the available maximum number of states.
- While designing such a sequential circuit, we may treat the unused states as don't care conditions.
- This is called as design with unused state and it leads to a more reliable design as well as reduction in the number of components.
- Solve the following example to get the idea about designing with unused states.

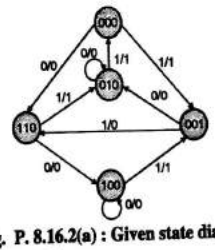
Ex. 8.16.2 : Implement the following state diagram using D flipflops. May 05. 10 Marks

Soln. :

- From the given state table it is evident that 3 flip flops are required to be used.
- The unused states are 011, 101 and 111.

Step 1 : Write the circuit excitation table :

- The required circuit excitation table is shown in Table P. 8.16.2(a).
- This state table has been derived from the given state diagram.



(C-688) **Fig. P. 8.16.2(a) : Given state diagram**

Table P. 8.16.2(a) : Circuit excitation table

Present state			Input	Next state			Output	Flip-flop inputs		
A	B	C	X	A_{n+1}	B_{n+1}	C_{n+1}	Y	D_A	D_B	D_C
0	0	0	0	1	1	0	0	1	1	0
0	0	0	1	0	0	1	1	0	0	1
0	0	1	0	0	1	0	0	0	1	0
0	0	1	1	1	1	0	0	1	1	0
0	1	0	0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	1	0	0
1	0	0	1	0	0	1	1	0	0	1
1	1	0	0	1	0	0	0	1	0	0
1	1	0	1	0	1	0	1	0	1	0

Scanned by CamScanner

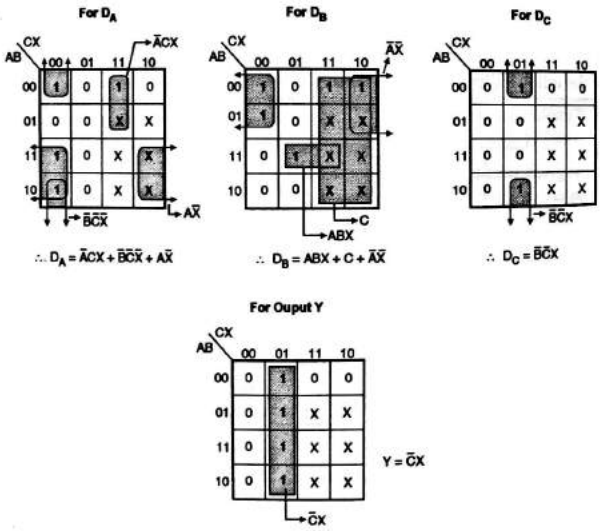
- Note that the entries in the column D_A have been made by taking into consideration the entries in the columns A and A_{n+1}.
- Similarly the entries in the columns of D_B and D_C have been made by considering B, B_{n+1} and C, C_{n+1} respectively.
- The excitation table for a general D FF is shown in Table P. 8.16.2(b).

Table P. 8.16.2(b) : Excitation table for a D FF

Present state	Next state	D
0	0	0
0	1	1
1	0	0
1	1	1

Step 2 : K maps and simplifications :

- The K maps for D_A, D_B and D_C and output Y are shown in Fig. P. 8.16.2(b).
- The simplified expression are also derived.

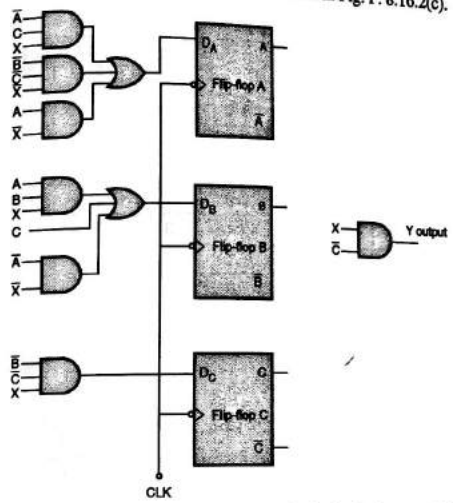


(C-689) **Fig. P. 8.16.2(b) : K-maps and simplifications**

- Note that these K-maps have been written from the circuit excitation table shown in Table P. 8.16.2(a). The inputs are A, B, C and X.
- But the excitation Table P. 8.16.2(a) does not contain all the possible combinations of A, B, C and X. So the missing combinations are treated as don't care conditions and denoted by X in the K-maps. They correspond to 011, 101 and 111.

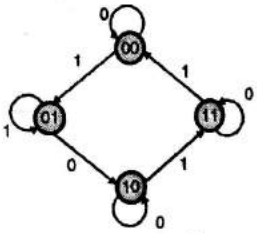
Step 3 : Logic diagram :

The logic diagram of the required sequential circuit is shown in Fig. P. 8.16.2(c).



(C-690) **Fig. P. 8.16.2(c) : Logic diagram of the required clocked sequential circuit**

Ex. 8.16.3 : For the state diagram shown in Fig. P. 8.16.3, obtain the state table and design circuit using minimum number of J-K flipflops. **May 04, 10 Marks**



(C-3762) **Fig. P. 8.16.3**

Soln. :

Step 1 : Write the state table :

Present state		Next state			
A	B	X = 0		X = 1	
		A_{n+1}	B_{n+1}	A_{n+1}	B_{n+1}
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

Scanned by CamScanner

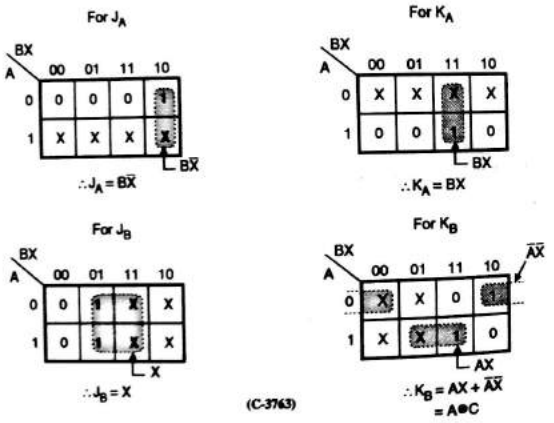
Step 2 : Number of flip-flops :

The number of states are four. Hence we need to use 2 flip-flops.

Step 3 : Write the circuit excitation table :

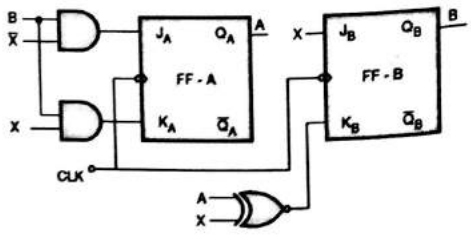
Present state		Input	Next state		FF inputs			
A	B	X	A_{n+1}	B_{n+1}	J_A	K_A	J_B	K_B
0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	×	1	×
0	1	0	1	0	1	×	×	1
0	1	1	0	1	0	×	×	0
1	0	0	1	0	×	0	0	×
1	1	0	1	1	×	0	×	0
1	1	1	0	0	×	1	×	1

Step 4 : K-maps and simplifications :



(C-3763)

Step 5 : Draw the logic diagram :



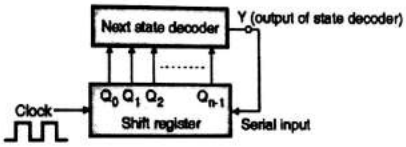
(C-3764) **Fig. P. 8.16.3(a)**

8.17 Sequence Generator :

- A sequence generator is a sequential circuit which generates the prescribed sequence at its output. The output sequence is produced in synchronization with the clock input.
- It is possible to design a sequence generator using either counters or using the shift registers.

8.17.1 Sequence Generator using Shift Register :

- The sequence generator is a circuit which generates a desired sequence of bits at its output in synchronization with the clock.
- Some of the applications of the sequence generator are as follows :
  1. Random bit generator
  2. Counters
  3. Code generators
  4. Period and sequence generator.
- Fig. 8.17.1 shows the basic structure of a sequence generator using a shift register.
- The outputs of an N bit shift register (Q_0 through Q_{N-1}) are applied as inputs to a combinational circuit called "Next state decoder".



(C-1511) **Fig. 8.17.1 : Basic structure of a sequence generator**

- And the output (Y) of the next state decoder is applied as the serial input to the shift register.
- The "Next state decoder" is designed by keeping in mind the required sequence.
- Design of a sequence generator using shift registers has been demonstrated in the following examples.

Ex. 8.17.1 : Design a sequence generator to generate the following sequence 1 0 1 1 0 ....

OR

Design **pulse train** generator using shift register to generate the following pulse : .....10110....

Soln. :

Table P. 8.17.1 : State stable for the sequence generator

Number of clock pulse	Flip flop outputs Q_2	Q_1	Q_0	Decimal equivalent
1	1	0	1	5 same
2	0	1	0	2
3	1	0	1	5
4	1	1	0	6
5	0	1	1	3

Prescribed sequence delayed by 2 bits

Prescribed sequence delayed by 1 bit

Prescribed sequence

The length of given sequence = L = 5 bits.

Scanned by CamScanner

1. The minimum number of flip flops (N) required to generate a sequence of length L is given by,
2. Length of the sequence $L = 2^N - 1$ ...(1)
3. Here L = 5. Therefore number of flip flops N = 3. Thus three flip-flops will have to be used.
4. Write the states of the circuit as shown in Table P. 8.17.1. Note that the column for Q_2 output consists of the prescribed sequence, whereas the column for Q_1 consists of 1 bit delayed version of the prescribed sequence and that for Q_0 consists of 2 bit delayed version.
5. The use of N = 3 i.e. three flip-flops will lead us to the correct generation of prescribed sequence, if and only if all the states in the state table are distinct. Otherwise we will have to increase the number of flip-flops.
6. In Table P. 8.17.1, all the states are not distinct. States 1 and 3 are same. Hence only three flip-flops are not sufficient. So assume N = 4 and prepare a new state table as Table P. 8.17.1(A) in a similar manner as the previous state table.

Table P. 8.17.1(A) : Modified state table using N = 4

Clock pulse number (state)	Flip flop outputs				Decimal Equivalent
	Q_3	Q_2	Q_1	Q_0	
1	1	0	1	1	11
2	0	1	0	1	5
3	1	0	1	0	10
4	1	1	0	1	13
5	0	1	1	0	6

- Note that with 4 flip-flop, all the states are distinct states.

7. **Design of next state decoder :**

- We have to find the output (Y) of the next state decoder which is connected to the serial input of the shift register so that the state changes from present to next as per our requirements.
- For example if $Q_3 Q_2 Q_1 Q_0$ = 1 0 1 1 i.e. (11 decimal). Then what should be the serial input so as to get $Q_3 Q_2 Q_1 Q_0$ = 0 1 0 1 (i.e. 5 decimal) ? The answer is, serial input should be "0" (right shifting is assumed).
- Applying same logic to Y which is output of the next state decoder and serial input to the shift register we find Y for each state as shown in Table P. 8.17.1(B).

Table P. 8.17.1(B) : Output Y of the next state decoder

Number of clock pulse (state)	Flip-flop outputs				Output Y	State (Decimal equivalent)
	Q_3	Q_2	Q_1	Q_0		
1	1	0	1	1	0	11
2	0	1	0	1	1	5
3	1	0	1	0	1	10
4	1	1	0	1	0	13
5	0	1	1	0	1	6

Valid states

Note that the Y output of the next state decoder is applied to D_3 (input of FF – 3) and the shift register is operated in the right shift mode.

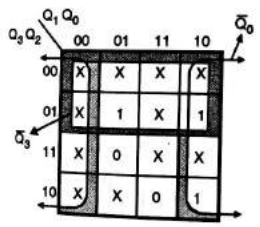
8. **K-map and simplifications :**

- The states 11, 5, 10, 13 and 6 shown in Table P. 8.17.1(B) are called as the stable states. So in the K-map corresponding to these states we will enter the corresponding value of Y (0 or 1 as per Table P. 8.17.1(B)).
- In the remaining boxes of K-maps we should enter the don't care (X) conditions.
- The K-map is shown in Fig. P. 8.17.1(a). The simplified expression for the output of the next state decoder is given by,

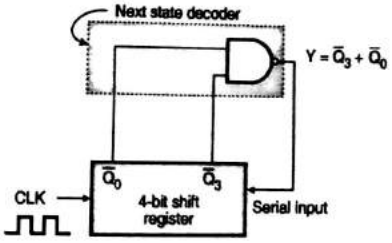
$$\therefore Y = \overline{Q_0} + \overline{Q_3}$$

9. **Draw the logic diagram :**

The logic diagram is shown in Fig. P. 8.17.1(b).



(C-1513) **Fig. P. 8.17.1(a) : K-map and simplifications**



(C-1514) **Fig. P. 8.17.1(b) : Logic diagram of the sequence generator**

Ex. 8.17.2 : Design a sequence generator to generate the following pulse train using shift register 1 1 0 0 1 1 1 0.

Soln. :

Step 1 : Decide the number of flip-flops :

The length of given sequence L = 8

$$L = 2^N - 1 \quad \text{where N = Number of flip-flops}$$

$$\therefore 8 = 2^N - 1$$

$$\therefore 2^N = 9 \qquad \therefore N = 4$$

Step 2 : Write the state table for the sequence generator :

- The state table for the sequence generator is shown in Table P. 8.17.2(a).

Scanned by CamScanner

Table P. 8.17.2(a) : State table for the generator

State	D	C	B	A	Decimal equivalent
1	1	1	0	0	12
2	1	0	0	1	9
3	0	0	1	1	3
4	0	1	1	1	7
5	1	1	1	0	14
6	1	1	0	1	13
7	1	0	1	1	11
8	0	1	1	0	6

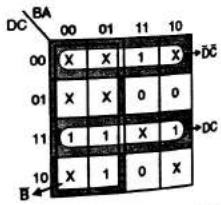
- Note that we have delayed the sequence in the opposite direction. Hence the directions of all the arrows have reversed.
- All the states in Table P. 8.17.2(a) are distinct states. Hence 4 flip-flops will be sufficient.

Step 3 : Find Y :

Refer Table P. 8.17.2(b) which shows the values of Y i.e. the output of the next state generator.

Table P. 8.17.2(b) : State table to obtain Y

State	D	C	B	A	Y
1	1	1	0	0	1
2	1	0	0	1	1
3	0	0	1	1	1
4	0	1	1	1	0
5	1	1	1	0	1
6	1	1	0	1	1
7	1	0	1	1	0
8	0	1	1	0	0



(C-1515) **Fig. P. 8.17.2(a) : K-map and simplification**

Step 4 : K-map and simplifications :

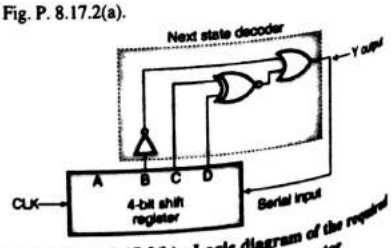
The K-map for output Y is as shown in Fig. P. 8.17.2(a).

$$\therefore \quad Y = \bar{B} + DC + \bar{D}\bar{C}$$

$$\therefore \quad Y = \bar{B} + (\overline{D \oplus C})$$

Step 5 : Draw the logic diagram :

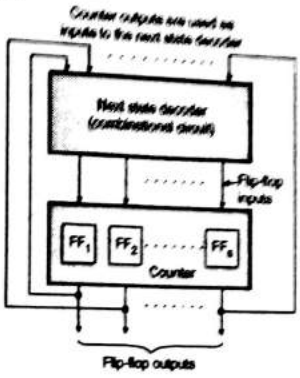
The logic diagram of the sequence generator is shown in Fig. P. 8.17.2(b).



(C-1516) **Fig. P. 8.17.2(b) : Logic diagram of the required sequence generator**

8.17.2 Sequence Generator using Counters :

- The general block diagram of a sequence generator constructed using counter is shown in Fig. 8.17.2.
- The next state decoder is a combinational circuit. The inputs to it are obtained from the flip-flop outputs and its outputs are applied to the inputs of the flip-flops as shown in Fig. 8.17.1.



(C-857) **Fig. 8.17.2 : A general block diagram of a sequence generator**

Deciding the number of flip-flops :

- We will be given the required sequence of 1's and 0's that is to be generated such as 1011011 ....
- Based on the given sequence we have to decide the number of flip-flops as follows.

1. Count the number of 1's and 0's in the given sequence.
2. Choose the higher number of the two. Let this number be N.
3. The number of FFs (n) is then calculated as,

$$N \le 2^{n-1}$$

- Let us apply this principle to the given sequence, i.e. 1011011 ...
- Number of 1's = 5, number of 0's = 2. So select higher one of them i.e. 5. So N = 5.
- But $N = 2^{n-1}$ Hence $5 \le 2^{n-1}$

$$\therefore \quad n = 4 \quad \text{... so four FFs will be required.}$$

Ex. 8.17.3 : For the following sequence, design a sequence generator using JK flip-flops.

1001001

Soln. :

Step 1 : Decide the number of flip-flops :

- Number of 1's = 3 and number of 0's = 4.
- So select higher one of them i.e. 4. So N = 4.
- Hence $N = 2^{n-1}$, so $4 \le 2^{n-1}$
- Therefore number of flip-flops n = 3.

Step 2 : Write the state table :

- Assume that the required sequence is obtained at the output of flip-flop C. The state assignment for the other outputs is shown in Table P. 8.17.3(a).

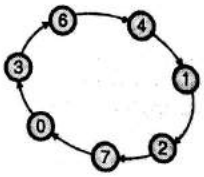
Scanned by CamScanner

Table P. 8.17.3(a)

Flip flop outputs			State
C	B	A	
1	0	0	4
0	0	1	1
0	1	0	2
1	1	1	7
0	0	0	0
0	1	1	3
1	1	0	6

These entries have been made in an arbitrary manner

This is the given sequence.



(C-858) **Fig. P. 8.17.3(a) : State diagram**

- There is only one unused state i.e. 5. We assume that the next state of 5 is don't care XXX.

Step 3 : Draw the state diagram :

From the state table, draw the state diagram as shown in Fig. P. 8.17.3(a).

Step 4 : Write the excitation table :

The excitation table of a JK FF is given in Table P. 8.17.3(b) and the circuit excitation table is shown in Table P. 8.17.3(c).

Table P. 8.17.3(b) : Excitation table of JK FF

Present state Q_n	Next state Q_{n+1}	FF inputs	
		J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

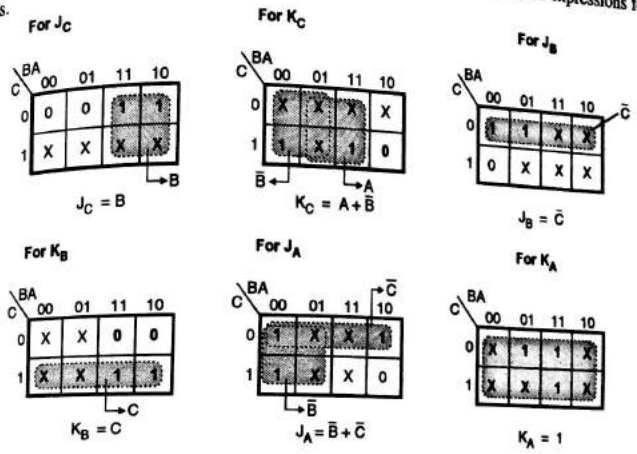
Table P. 8.17.3(c) : Circuit excitation table

Present state			Next state			Flip-flop inputs					
C	B	A	C_{n+1}	B_{n+1}	A_{n+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	1	1	0	X	1	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	1	1	1	X	X	0	1	X
0	1	1	1	1	0	1	X	X	0	X	1
1	0	0	0	0	1	X	1	0	X	1	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	1	0	0	X	0	X	1	0	X
1	1	1	0	0	0	X	1	X	1	X	1

← The next state and FF inputs are don't care for the unused state.

Step 5 : K-maps and simplified Boolean expressions :

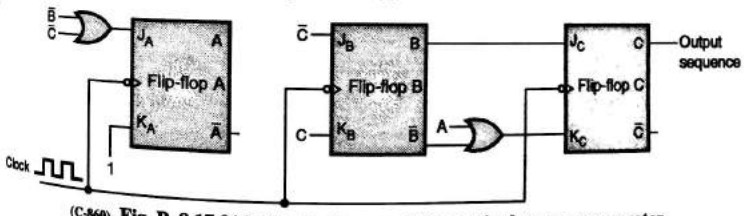
Fig. P. 8.17.3(b) shows the K-maps and corresponding simplified boolean expressions for all the FF inputs.



(C-859) **Fig. P. 8.17.3(b) : K-map and simplification**

Step 6 : Draw the logic diagram :

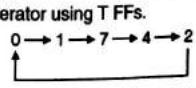
The logic diagram is shown in Fig. P. 8.17.3(c).



(C-860) **Fig. P. 8.17.3(c) : Logic diagram of the required sequence generator**

- Note that the required sequence is obtained at the C output.
- All the flip-flops are clocked simultaneously.

Ex. 8.17.4 : Design a sequence generator using T FFs.



(C-861) **Fig. P. 8.17.4(a) : Given state diagram**

Soln. : Draw the bush diagram to check for the lockout conditions.

We shall assume that the next state of each unused state is a don't care condition, while writing the circuit excitation table.

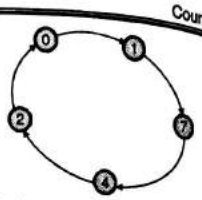
Scanned by CamScanner

Step 1 : Draw the state diagram :

- The state diagram is as shown in Fig. P. 8.17.4(b).

Step 2 : Number of FFs :

- Since the highest state is "7" i.e. 111, we have to use three T flip-flops.



(C-862) **Fig. P. 8.17.4(b) : State diagram**

Step 3 : Write the excitation table :

- Table P. 8.17.4(a) shows the excitation table for a T FF.
- Table P. 8.17.4(b) shows the circuit excitation table.

Table P. 8.17.4(a) : Excitation table for a T FF

Present state	Next state	T input
0	0	0
0	1	1
1	0	1
1	1	0

Table P. 8.17.4(b) : Circuit excitation table

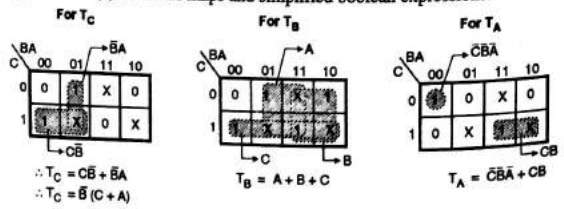
Present state			Next state			Flip-flop inputs		
C	B	A	C_{n+1}	B_{n+1}	A_{n+1}	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	1	1	1	1	1	0
0	1	0	0	0	0	0	1	0
0	1	1	X	X	X	X	X	X
1	0	0	0	1	0	1	1	0
1	0	1	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X
1	1	1	1	0	0	0	1	1

FF inputs are don't care for the unused inputs.

Next state of each unused state is don't care. We can assume don't care conditions to be logic 1's.

Step 4 : K-maps and simplifications :

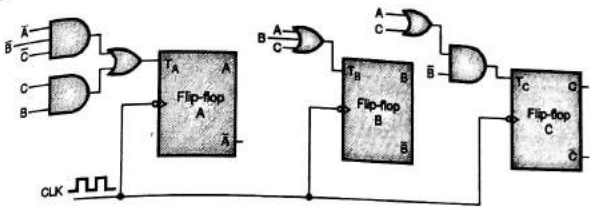
Refer Fig. P. 8.17.4(c) for the K-maps and simplified boolean expressions.



(C-863) **Fig. P. 8.17.4(c) : K-maps and simplifications**

Step 5 : Logic diagram :

Fig. P. 8.17.4(d) shows the required logic diagram.



(C-864) **Fig. P. 8.17.4(d) : Logic diagram of the required counter**

Step 6 : Examine the counter action for the unused states :

- The unused states are 3, 5 and 6.

For state 3 :

C = 0, B = 1, A = 1.

Table P. 8.17.4(c)

Present state			FF inputs			Next state			
C	B	A	$T_C = \bar{B}(C+A)$	$T_B = A+B+C$	$T_A = \bar{C}\bar{B}\bar{A} + CB$	C_{n+1}	B_{n+1}	A_{n+1}	
0	1	1	0	1	0	0	0	1	→ Valid state
1	0	1	1	1	0	0	1	1	
1	1	0	0	1	1	1	0	1	

Thus if counter goes to state 3 it returns to state 1 which is a valid state.

For state 5 :

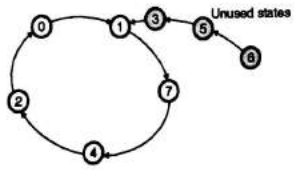
Refer Table P. 8.17.4(c). The next state of 5 is 3 which ultimately terminates into 1.

For state 6 :

Refer Table P. 8.17.4(c). The next state of 6 is 5 and that of 5 is 3 which ultimately gets terminated into 1 which is a valid state.

Step 7 : Draw the bush diagram :

- The bush diagram is shown in Fig. P. 8.17.4(e).
- From the bush diagram it is evident that lockout condition will be avoided.



(C-865) **Fig. P. 8.17.4(e) : Bush diagram**

Ex. 8.17.5 : Design a synchronous counter using JK FF for following sequence :

$1 \rightarrow 0 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$

May 11. 10 Marks

Scanned by CamScanner

Soln. :

Given sequence :

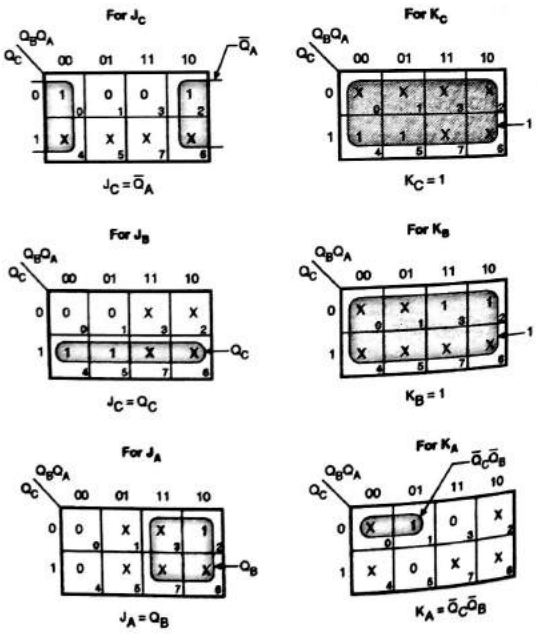
$$1 \rightarrow 0 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$$

Step 1 : Write the excitation table :

Table P. 8.17.5 : Circuit excitation table

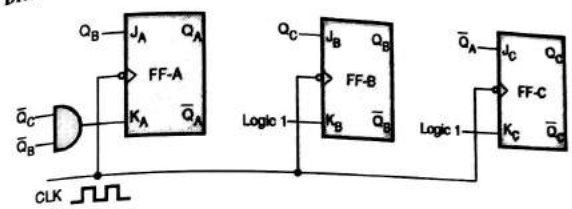
Present state			Next state			Flip flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	1	0	0	1	×	0	×	0	x
0	0	1	0	0	0	0	×	0	×	×	1
0	1	0	1	0	1	1	×	×	1	1	x
0	1	1	0	0	1	0	×	×	1	×	0
1	0	0	0	1	0	×	1	1	×	0	x
1	0	1	0	1	1	×	1	1	×	×	0
1	1	0	×	×	×	×	×	×	×	×	x
1	1	1	×	×	×	×	×	×	×	×	x

Step 2 : K-maps and simplification :



(C-3285) **Fig. P. 8.17.5(a)**

Step 3 : Draw the logic diagram :



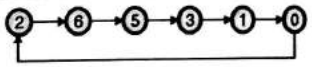
(C-3286) **Fig. P. 8.17.5(b)**

Ex. 8.17.6 : For synchronous sequence counter with sequence 2 – 6 – 5 – 3 – 1 – 0 – 2

1. Give present state / next state table
2. Write state transition table using D flip – flops
3. Simplify and realize the circuit. Draw state diagram.
4. If the counter enters any unused state will it go to lockout condition. Justify.

Dec. 07. 12 Marks

Soln. :

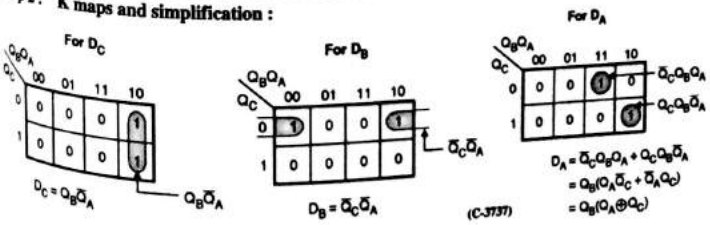


(C-3736)

Step 1 : Write the state transition table :

Present state			Next state			Flip flop inputs		
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	D_C	D_B	D_A
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	0
0	1	0	1	1	0	1	1	0
0	1	1	0	0	1	0	0	1
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	1	0	1	1	0	1
1	1	1	0	0	0	0	0	0

Step 2 : K maps and simplification :

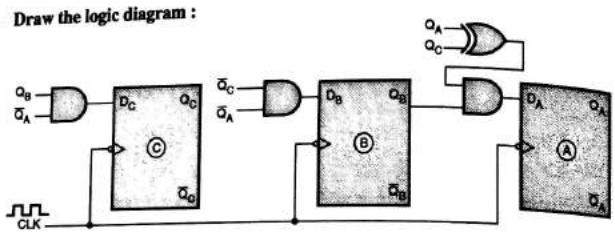


$$D_A = \bar{Q}_C Q_B Q_A + Q_C Q_B \bar{Q}_A$$
$$= Q_B(Q_A\bar{Q}_C + \bar{Q}_A Q_C)$$
$$= Q_B(Q_A \oplus Q_C)$$

(C-3737)

Scanned by CamScanner

Step 3 : Draw the logic diagram :



(C-3738) **Fig. P. 8.17.6**

Ex. 8.17.7 : Design a sequence generator using T flip-flop for the given sequence. Check for lock-out conditions.

$0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 0$

May 10, Dec. 12. 10 Marks

Soln. :

Step 1 : Draw the state diagram :

The state diagram is as shown in Fig. P. 8.17.7(a).

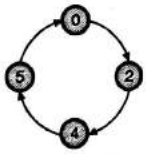
Step 2 : Number of FFs :

Since the highest state is 5, we have to use three T flip-flops.

Step 3 : Write the excitation table :

Excitation table for a T FF

Present state	Next state	T input
0	0	0
0	1	1
1	0	1
1	1	0



(C-3284)

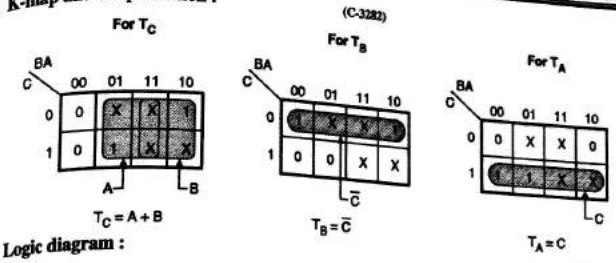
Fig. P. 8.17.7(a)

Circuit excitation table

Present state			Next state			Flip-flop inputs		
C	B	A	C_{n+1}	B_{n+1}	A_{n+1}	T_C	T_B	T_A
0	0	0	0	1	0	0	1	0
0	0	1	×	×	×	×	×	×
0	1	0	1	0	0	1	1	0
0	1	1	×	×	×	×	×	×
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	0	1
1	1	0	×	×	×	×	×	×
1	1	1	×	×	×	×	×	×

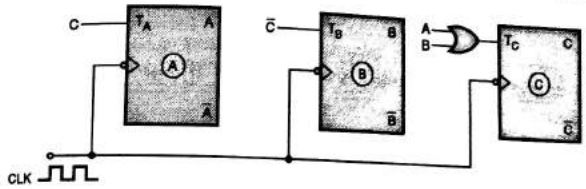
Step 4 : K-map and simplification :

(C-3282)



Step 5 : Logic diagram :

(C-3283)



Step 6 : Examine the counter action for the unused states :

The unused states are 1, 3, 6 and 7.

Present state			FF inputs			Next state		
C	B	A	$T_C = A + B$	$T_B = \bar{C}$	$T_A = C$	C_{n+1}	B_{n+1}	A_{n+1}
0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	1	0	1
1	1	0	1	0	1	0	1	1
1	1	1	1	0	1	0	1	0

Valid state

Thus if counter goes to state 7 it returns to state 2.

Similarly states 1, 3 and 6 gets terminated to 2.

Ex. 8.17.8 : Design a sequence generator to generate the sequence using 'D' FF 1101001 and repeat. Draw neat state diagram and circuit diagram.

Dec. 13. 10 Marks

Soln. :

Step 1 : Decide the number of flip flops :

Number of 1's = 4 and number of 0's = 3.

So select higher one of them i.e. 4. So N = 4.

Hence $N = 2^{n-1}$, so $4 \leq 2^{n-1}$

$\therefore$ Number of flip flop n = 3

Step 2 : Write the state table and state diagram :

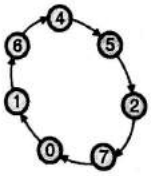
Assume that the required sequence is obtained at the output of flip flop C. The state assignment for other outputs is as shown in Table P. 8.17.8(a).

Scanned by CamScanner

Table P. 8.17.8(a)

Flip flop outputs			State
C	B	A	
1	0	0	4
1	0	1	5
0	1	0	2
1	1	1	7
0	0	0	0
0	0	1	1
1	1	0	6

Given sequence



(C-4206) **Fig. P. 8.17.8(a) : State diagram**

- There is only one unused state i.e. 3. We assume that next state of 3 is don't care.

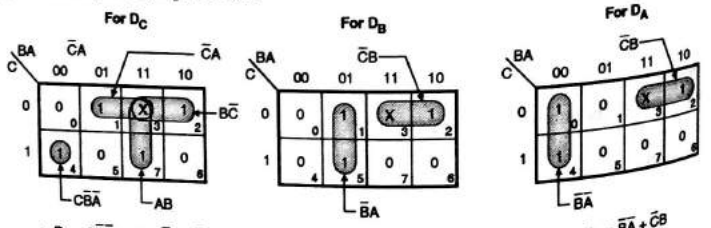
Step 3 : Write the excitation table :

Circuit excitation table is as shown in Table P. 8.17.8(b).

Table P. 8.17.8(b)

Present state			Next state			Flip flop inputs		
C	B	A	C_{n+1}	B_{n+1}	A_{n+1}	D_C	D_B	D_A
0	0	0	0	0	1	0	0	1
0	0	1	1	1	0	1	1	0
0	1	0	1	1	1	1	1	1
0	1	1	×	×	×	×	×	×
1	0	0	1	0	1	1	0	1
1	0	1	0	1	0	0	1	0
1	1	0	1	0	0	1	0	0
1	1	1	0	0	0	0	0	0

Step 4 : K-maps and simplification : (C-4207)

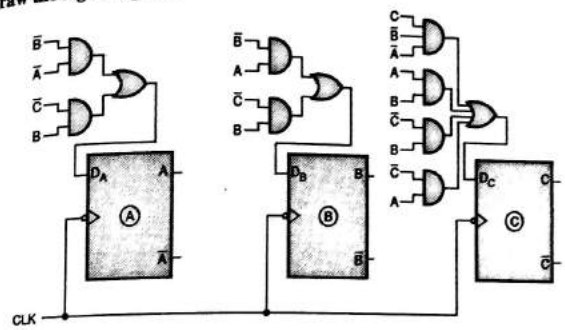


$\therefore D_C = C\bar{B}\bar{A} + BA + \bar{C}B + \bar{C}A$

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

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

Step 5 : Draw the logic diagram :



(C-4208) **Fig. P. 8.17.8(b) : Logic diagram**

Ex. 8.17.9 : Design a sequence generator for following sequence. Identify and check for lock out condition.

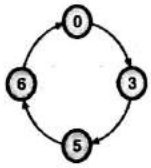
$0 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 0.$

May 14. 10 Marks

Soln. :

Step 1 : Draw the state diagram :

The state diagram is as shown in Fig. P. 8.17.9.



(C-4968) **Fig. P. 8.17.9**

Step 2 : Number of flip flops :

Since the highest state is 6, we have to use three T flip-flops.

Step 3 : Write the excitation table :

Excitation table of T flip-flop :

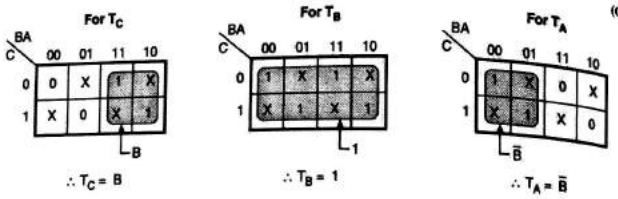
Present state	Next state	T input
0	0	0
0	1	1
1	0	1
1	1	0

Circuit excitation table

Present state			Next state			Flip flop inputs		
C	B	A	C_{n+1}	B_{n+1}	A_{n+1}	T_C	T_B	T_A
0	0	0	0	1	1	0	1	1
0	0	1	×	×	×	×	×	×
0	1	0	×	×	×	×	×	×
0	1	1	1	0	1	1	1	0
1	0	0	×	×	×	×	×	×
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	×	×	×	×	×	×

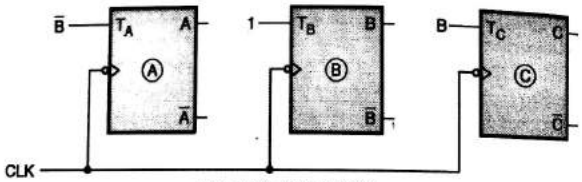
Scanned by CamScanner

Step 4 : K-map and simplification :



(C-4969)

Step 5 : Logic diagram :



(C-4970) **Fig. P. 8.17.9(a)**

Step 6 : Examine the counter action for the unused states :

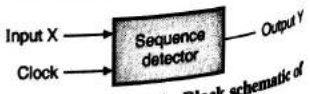
The unused states are 1, 2, 4, 7.

Present state			Flip flop inputs			Next state		
C	B	A	$T_C = B$	$T_B = 1$	$T_A = \bar{B}$	C_{n+1}	B_{n+1}	A_{n+1}
0	0	1	0	1	1	0	1	0
0	1	0	1	1	0	1	0	0
1	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	1

Thus if counter goes to state 7 it returns to state. Similarly states 1, 2, 4 are terminated to 1.

8.18 Sequence Detector :

- In the previous section we have discussed the design of a sequence generator which produces the desired output sequence.
- Now let us see how to design a sequence detector which is a logic circuit used to detect a desired sequence.
- The available or received sequence is applied at the input of the detector. It checks the sequence bit by bit. If required bit is at its input then the detector moves to the next state.
- Detector output will be equal to zero as long as the complete desired sequence is not detected. The output will be equal to 1 if the complete desired sequence has been detected.



(C-691) **Fig. 8.18.1 : Block schematic of a sequence detector**

- The sequence to be detected is given to us. Follow the steps given below to design the sequence detector.

Steps to design a sequence detector :

Step 1 : A sequence to be detected is given to us.
Step 2 : Develop the state diagram.
Step 3 : Write the state table and circuit excitation table.
Step 4 : From the circuit excitation table write K-maps and obtain simplified equations.
Step 5 : Draw the logic diagram.

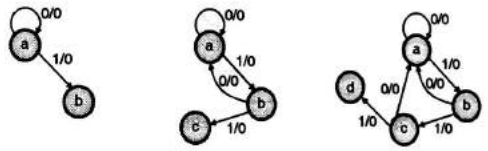
- The design of a sequence detector is demonstrated in the following example.

Ex. 8.18.1 : Design a sequence detector to detect three or more consecutive 1's in a string of bits coming through an input line.

Soln. :

Step 1 : Develop the states diagram : State "a" :

- Let the initial state be "a". If the input is 0 then the detector will stay in the same state. But if the input is equal to 1 then it will move to the next state "b" as shown in Fig. P. 8.18.1(a).
- For both the values of input the output remains to be equal to "0" since the complete sequence has not yet been detected.



(a) State "a" (b) State "b" (c) State "c"

(C-692) **Fig. P. 8.18.1 : Developed state diagram**

State "b" :

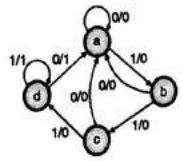
- Let the detector be in state "b".
- If the next input is 1, then the circuit will move on to next state "c". This indicates the arrival of two consecutive 1's as shown in Fig. P. 8.18.1(b).
- But if the input is "0" then we go back to state "a". The output will be equal to "0" for both the values of input.

State "c" :

- If the next input is "1" then the circuit will move to next state "d" indicating the arrival of three consecutive 1's as shown in Fig. P. 8.18.1(c). Output will be zero.
- But if the input is 0 then the circuit goes back to state "a" output will be equal to 0.

State "d" :

- If more 1's as detected, then the circuit stays in state "d", whereas if the next input is 0 then we should return to state "a".
- The complete state diagram is shown in Fig. P. 8.18.1(d).
- In this way the circuit stays at "d" as long as there are three or more consecutive 1's received.



(C-693) **Fig. P. 8.18.1(d) : Complete state diagram**

Scanned by CamScanner

Step 2 : Assign binary codes to the various states :

Let the binary code assignment for the states a to d be as shown in Table P. 8.18.1(a).

Table P. 8.18.1(a) : State assignment table

State name	Assigned binary state
a	00
b	01
c	10
d	11

Step 3 : Write down the state table :

Table P. 8.18.1(b) represents the state table for the sequence detector to be designed.

Table P. 8.18.1(b) : State table for sequence detector

Present state		Input	Next state		Output	
A	B	X	A_{n+1}	B_{n+1}	Y	
0	0	0	0	0	0	Output is 0 because the desired sequence is yet to be detected
0	0	1	0	1	0	
0	1	0	0	0	0	
0	1	1	1	0	0	
1	0	0	0	0	0	
1	0	1	1	1	0	
1	1	0	0	0	1	Output is 1 because the desired sequence has been detected
1	1	1	1	1	1	

Step 4 : Decide the type of flip-flop and number of flip-flops :

- Since the number of states is 4, we have to use 2 flip-flops A and B.
- Let us choose the D flip-flops for implementing the detector.

Step 5 : Write the excitation table :

The circuit excitation table is shown in Table P. 8.18.1(d) and the excitation table for a D flip-flop is shown in Table P. 8.18.1(c).

Table P. 8.18.1(c) : Excitation table of a D FF

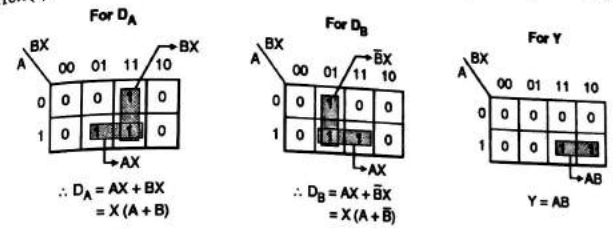
Present state	Next state	D
0	0	0
0	1	1
1	0	0
1	1	1

Table P. 8.18.1(d) : Circuit excitation table

Present state		Input	Next state		Output	FF inputs	
A	B	X	A_{n+1}	B_{n+1}	Y	D_A	D_B
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1

Step 6 : Write K-maps and simplify :

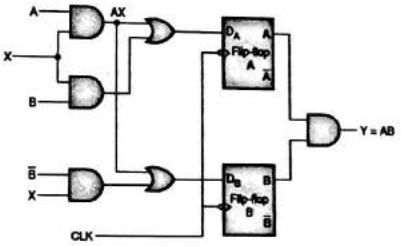
The K-maps and simplified equations for the FF inputs D_A and D_B and output Y are given in Fig. P. 8.18.1(e).



(C-694) **Fig. P. 8.18.1(e) : K-maps and simplifications**

Step 7 : Draw the logic diagram :

The logic diagram of the required sequence detector is shown in Fig. P. 8.18.1(f).



(C-695) **Fig. P. 8.18.1(f) : Logic diagram of the required sequence detector**

Review Questions

Q. 1 What is counter ? What are its types ?

Q. 2 Explain the terms "synchronous" and "asynchronous".

Q. 3 What is the difference between synchronous and asynchronous counter ?

Q. 4 What are the merits and demerits of the synchronous counter over the asynchronous counter ?

Q. 5 State the procedure for designing mod counters from N-bit ripple counter.

Q. 6 What is the similarity between a decade counter and a binary counter ? Explain in brief.

Q. 7 What are the applications of counter ?

Q. 8 How is a counter different from a register ?

Q. 9 Write the count sequence of three bit binary down counter.

Q. 10 How does a counter works as frequency divider ?

Q. 11 What is synchronous counter ?

Q. 12 Define modulus of counter. State the states of MOD 5 counter.

Q. 13 Explain the working of 4-bit ripple counter (asynchronous counter) using T flip-flop with suitable circuit diagram and timing diagram.

Scanned by CamScanner

Q. 14 Write a note on Mod-8 synchronous up-counter using T flip-flop.

Q. 15 Write a note on Mod-8 synchronous down counter using T flip-flop.

Q. 16 Draw the divide by 7 asynchronous up counter using T flip flop. Write truth table. Draw timing diagram.

Q. 17 Draw the circuit for mod-12 counter. Explain the same with neat waveforms.

Q. 18 Compare synchronous and ripple counters.

Q. 19 Compare counters and registers.

Q. 20 Design a 3 bit synchronous counter using JK fillip flops.

Q. 21 What is lock out ? How is it avoided ?

Q. 22 Explain decade counter.

Q. 23 Draw the general model of sequential circuits and explain.

Q. 24 What is FSM ?

Q. 25 Compare synchronous and asynchronous sequential circuits.

Q. 26 Define the following :
1. Input and output variables 2. State variable 3. State

Q. 27 Define present state and next state.

Q. 28 Explain the design procedure for a sequence generator.

Q. 29 What is the difference between asynchronous and synchronous sequential circuits ?

Q. 30 What are the advantages of FSM ?

Q. 31 State the disadvantages of FSM.

Q. 32 What is the advantage of state reduction technique ?

Q. 33 Write short note on sequence generators.

Q. 34 Write a short note on sequence detectors.

8.19 University Questions and Answers :

Q. 1 What is modulus of the counter ? For MOD-6 counter how many flip-flops are needed ? **(Dec. 2015, 2 Marks)**

Ans. :

For modulus of the counter refer section 8.5.

In MOD-6 number of states is 6 so the number of FFs required is 3.

□□□

CHAPTER 9

Module 4

Shift Registers

Syllabus :

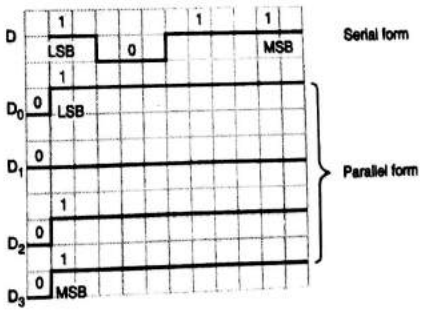
Shift registers : SISO, SIPO, PIPO, PISO, Bidirectional shift registers, Universal shift registers, Ring and Twisted ring / Johnson counter, sequence generator.

9.1 Introduction :

- In chapter 7, we have stated various applications of flip-flops. One of them was "registers".
- Flip-flop is a 1 bit memory cell which can be used for storing the digital data.
- To increase the storage capacity in terms of number of bits, we have to use a group of flip-flops. Such a group of flip-flops is known as **a register.**
- Thus register is a group of flip-flops. The "n-bit" register will consist of "n" number of flip-flops and it is capable of storing an "n bit" word.

9.2 Data Formats :

- The data can be entered in **serial** (one bit at a time) manner or in the **parallel** form (all the bits at the same time) into a register. And the stored data can be retrieved in the serial or parallel form.
- A four bit digital data 1 0 1 1 is represented in the serial and parallel forms in Fig. 9.2.1.

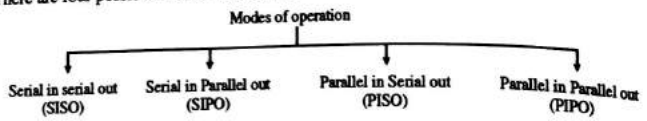


(C-729(b)) **Fig. 9.2.1 : Data representation in serial and parallel forms**

Scanned by CamScanner

9.3 Classification of Registers :

- Registers are classified on the basis of how the data bits are entered and taken out from a register. There are four possible modes as follows :



(C-729(I)) **Fig. 9.3.1 : Modes of operation of registers**

- We can design all these registers using discrete flip-flops such as SR, JK or D flip-flops.
- But the registers are also available in the IC form.
- Some of the registers available in 54/74 TTL series as listed in Table 9.3.1.

Table 9.3.1 : Shift registers available in 74/54 series

IC number	Description
7491, 7491 A	8 bit serial in, serial out
7494	4 bit parallel in, serial out
7495	4 bit serial/parallel in, parallel out (shift right shift left)
7496	5 bit parallel in / parallel out, serial in / serial out.

9.4 Buffer Registers :

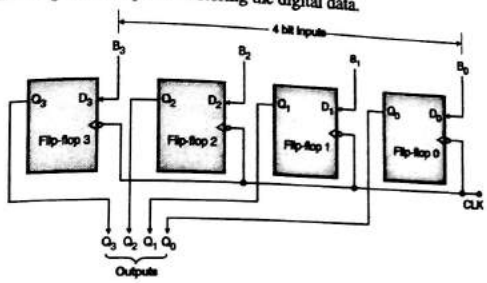
- The simplest type of register constructed using four D flip-flops is shown in Fig. 9.4.1. This is a 4 bit register, but we can construct an n-bit register by following the same principle.
- This register is also called as the **buffer register.**
- Each D-flip-flop is negative edge triggered and all the flip-flops are connected to a common clock signal. Hence all of them are triggered at the same instant of time, i.e. all the flip flops will change their state at the same instant of time.
- Buffer registers are used for temporary storage of digital words.

Operation :

- Let us assume that the word to be stored is $B_3 B_2 B_1 B_0 = 1\,0\,1\,0$.
- These bits are connected to the D inputs of the four D flip-flops as shown in Fig. 9.4.1.
- Then the clock pulse is applied.
- Corresponding to the first negative edge of the clock pulse, the outputs of all the D flip-flops will follow their respective inputs.

$$\therefore \quad Q_3 Q_2 Q_1 Q_0 = B_3 B_2 B_1 B_0 = 1\,0\,1\,0$$

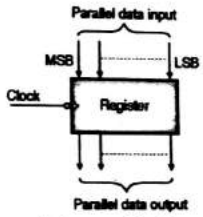
- Even if the inputs are now changed, the output remains latched to 1 0 1 0 till the next negative edge of the clock arrives at the input.
- Thus the buffer register is capable of storing the digital data.



(C-729(I)) **Fig. 9.4.1 : A four bit buffer register using D flip-flops**

Schematic diagram :

The schematic diagram of a buffer register is as shown in Fig. 9.4.2.



(C-730) **Fig. 9.4.2 : Schematic diagram of buffer register**

Conclusions :

Some of the important conclusions from the discussion till now are as follows :

1. There must be one-FF for each bit to be stored. Therefore to store a 4 bit number we need four flip-flops.
2. Note that all the four input bits $B_3 B_2 B_1 B_0$ are loaded into the buffer register simultaneously, i.e. at the same instant of time.
3. Therefore this way of applying the input and taking the output is called as Parallel Input Parallel Output (PIPO) and the mode of operation is called as **parallel shifting.**

9.5 Shift Registers :

MU : May 04, Dec. 05, Dec. 07, Dec. 10, Dec. 11, Dec. 14, Dec. 15, Dec. 16

University Questions

Q. 1 What is a shift register ?

(May 04, Dec. 05, Dec. 07, Dec. 10, Dec. 11, Dec. 14, Dec. 15, Dec. 16, 5 Marks)

- The binary data in a register can be moved within the register from one flip-flop to the other or outside it with application of clock pulses.
- The registers that allow such data transfers are called as shift registers.
- Shift registers are used for data storage, data transfer and certain arithmetic and logic operations.

Modes of operation of a shift register :

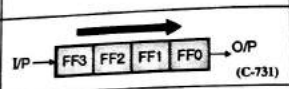
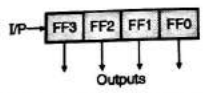
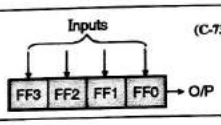
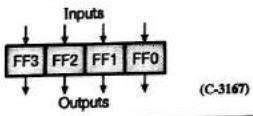
The various modes in which a shift register can operate are as follows :

1. Serial Input Serial Output. (SISO)
2. Serial Input Parallel Output. (SIPO)
3. Parallel In Serial Out. (PISO)
4. Parallel In Parallel Out. (PIPO)

Scanned by CamScanner

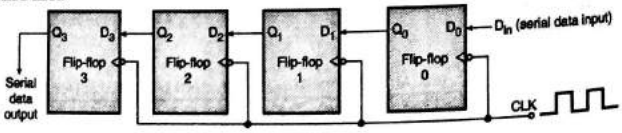
These modes are explained in brief in Table 9.5.1.

Table 9.5.1 : Brief explanation of various modes of shift register

Sr. No.	Mode	Illustrative diagram	Comments
1.	Serial input serial output (serial shift right)		Data bits shift from left to right by 1 position per clock cycle.
2.	Serial input serial output (serial shift left)		Data bits shift from right to left by 1 position per clock.
3.	Serial input parallel output		All output bits are made available simultaneously after 4-clock pulses.
4.	Parallel input serial output		All inputs are loaded simultaneously but output bit by bit.
5.	Parallel input parallel output		All inputs are loaded simultaneously and are available at the output simultaneously.

9.5.1 Serial Input Serial Output (Shift Left Mode) :

- The serial input serial output type shift register with shift left mode is shown in Fig. 9.5.1.
- Let all the flip-flops be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$.
- We are going to illustrate the entry of a four bit binary number 1 1 1 1 into the register.
- When this is to be done, this number should be applied to "D_{in}" bit by bit with the MSB bit applied first.



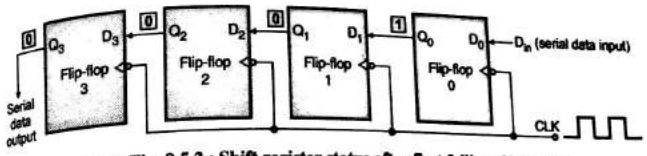
(C-732) **Fig. 9.5.1 : Serial shift left register**

- The D input of FF-0 i.e. D_0 is connected to serial data input (D_{in}). Output of FF-0 i.e. Q_0 is connected to the input of the next flip-flop i.e. D_1 and so on.

Operation :

- Before application of clock signal let $Q_3\,Q_2\,Q_1\,Q_0 = 0\,0\,0\,0$ and apply MSB bit of the number to be entered to D_{in}. So $D_{in} = D_0 = 1$.
- Apply the clock. On the first falling edge of clock, the FF-0 is set and the stored word in the register is

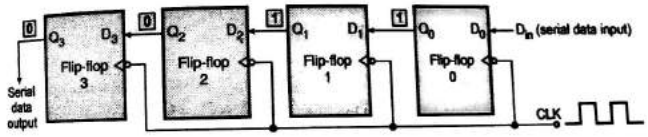
$$Q_3\,Q_2\,Q_1\,Q_0 = 0\,0\,0\,1$$



(C-733) **Fig. 9.5.2 : Shift register status after first falling clock edge**

- Apply the next bit to D_{in}. So $D_{in} = 1$.
- As soon as the next negative edge of the clock hits, FF-1 will set and the stored word changes to,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0\,0\,1\,1$$



(C-734) **Fig. 9.5.3 : Shift register status after the second falling edge of clock**

- Apply the next bit to be stored i.e. 1 to D_{in}.
- Apply the clock pulse. As soon as the third negative clock edge hits, FF-2 will be set and the output get modified to,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0\,1\,1\,1$$

- Similarly with $D_{in} = 1$, and with the fourth negative clock edge arriving, the stored word in the register is

$$Q_3\,Q_2\,Q_1\,Q_0 = 1\,1\,1\,1$$

(C-737) **Table 9.5.2 : Summary of shift left operation**

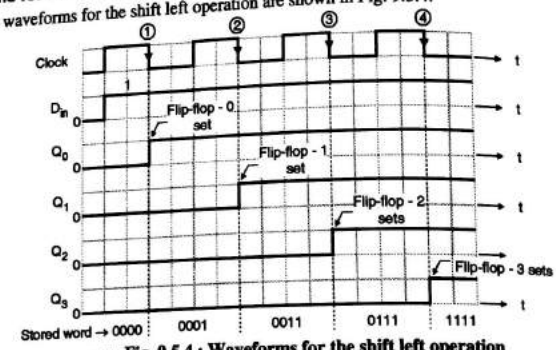
	CLK	Q_3	$Q_2 = D_3$	$Q_1 = D_2$	$Q_0 = D_1$	Serial input $D_{in} = D_0$
Initially		0	0	0	0	
1st	↓	0	0	0	1	1
2nd	↓	0	0	1	1	1
3rd	↓	0	1	1	1	1
4th	↓	1	1	1	1	1

Direction of data travel ←

Scanned by CamScanner

Waveforms for shift left operation :

The waveforms for the shift left operation are shown in Fig. 9.5.4.

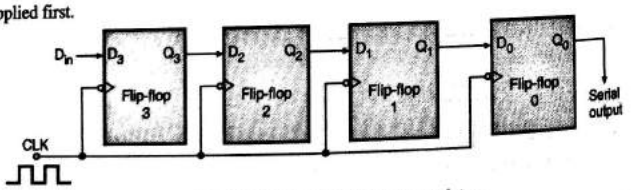


(C-738) **Fig. 9.5.4 : Waveforms for the shift left operation**

Important note : Using the SISO mode, we needed 4-clock pulses to store a 4-bit word. So in general we can conclude that it requires n number of clock pulses to store an n bit word using SISO mode.

9.5.2 Serial In Serial Out (Shift Right Mode) :

- The serial input serial output type shift register with shift right mode is shown in Fig. 9.5.5.
- Let all the flip-flops be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$.
- We are going to illustrate the entry of a four bit binary number 1 1 1 1 into the register.
- When this is to be done, this number should be applied to "D_{in}" bit by bit with the LSB bit applied first.



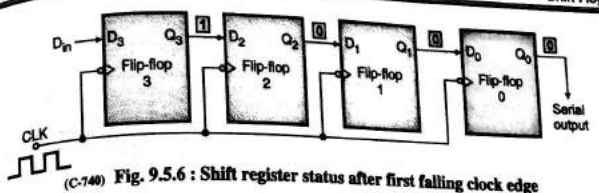
(C-739) **Fig. 9.5.5 : Serial shift right register**

- The D input of FF-3 i.e. D_3 is connected to serial data input (D_{in}). Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

Operation :

- Before application of clock signal let $Q_3\,Q_2\,Q_1\,Q_0 = 0\,0\,0\,0$ and apply LSB bit of the number to be entered to D_{in}. So $D_{in} = D_3 = 1$.
- Apply the clock. On the first falling edge of clock, the FF-3 is set, and the stored word in the register is

$$Q_3\,Q_2\,Q_1\,Q_0 = 1000$$

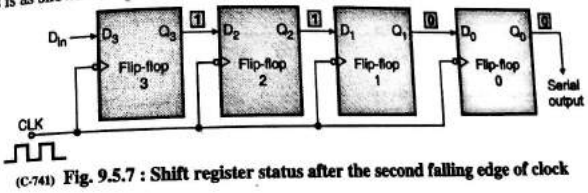


(C-740) **Fig. 9.5.6 : Shift register status after first falling clock edge**

- The shift register after the application of the first clock pulse is as shown in Fig. 9.5.6.
- Apply the next bit to D_{in}. So $D_{in} = 1$.
- As soon as the next negative edge of the clock is applied, FF-2 will set and the stored word changes to,

$$Q_3\,Q_2\,Q_1\,Q_0 = 1100$$

- This is as shown in Fig. 9.5.7.



(C-741) **Fig. 9.5.7 : Shift register status after the second falling edge of clock**

- Apply the next bit to be stored i.e. 1 to D_{in}.
- Apply the clock pulse. As soon as the third negative clock edge gets applied, FF-1 will be set and the output get modified to,

$$Q_3\,Q_2\,Q_1\,Q_0 = 1110$$

- Similarly with $D_{in} = 1$ and with the fourth negative clock edge arriving, the stored word in the register is

$$Q_3\,Q_2\,Q_1\,Q_0 = 1111$$

Table 9.5.3 summarizes the shift right operation.

(C-744) **Table 9.5.3 : Summary of shift right operation**

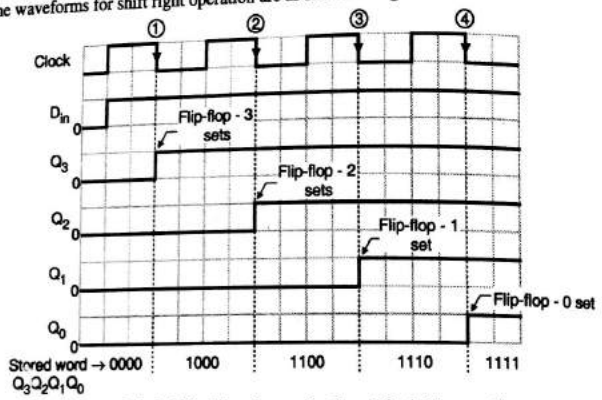
	CLK	$D_{in} = D_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
1st	↓	1	1	0	0	0
2nd	↓	1	1	1	0	0
3rd	↓	1	1	1	1	0
4th	↓	1	1	1	1	1

Direction of data travel

Scanned by CamScanner

Waveforms for shift right operation :

The waveforms for shift right operation are as shown in Fig. 9.5.8.

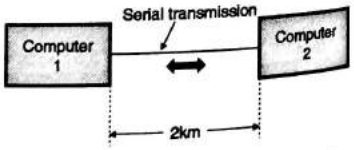


(C-745) **Fig. 9.5.8 : Waveforms for the shift right operation**

Important note : Using the SISO mode, we needed 4-clock pulses to store a 4-bit word. So in general we can conclude that it requires n number of clock pulses to store an n bit word using SISO mode.

9.5.3 Applications of Serial Operation :

- The transmission of data from one place to the other takes place in serial manner as shown in Fig. 9.5.9. The serial shifting of data transmits one bit at a time per clock cycle.
- It takes a longer time for serial transmission, because the time required to transmit one bit is equal to the time corresponding to one clock cycle. The parallel transmission is much faster as it can transmit 8 bits per clock cycle. But it needs more wires for connecting a transmitter to receiver.
- However for long distance communication where the distances are in kilometres, serial communication has an advantage that only one conductor is required to be used for the data transfer.



(C-746) **Fig. 9.5.9 : Application of serial operation**

9.6 Serial In Parallel Out (SIPO) :

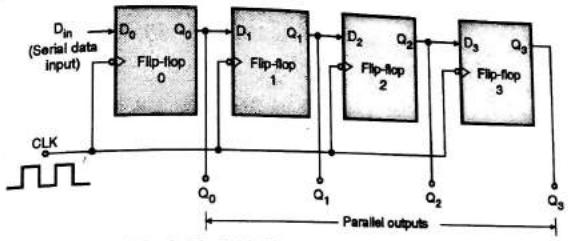
MU : Dec. 16

University Questions

Q. 1 Explain PISO and SIPO operations. **(Dec. 16, 5 Marks)**

- In this operation the data is entered serially and taken out in parallel.

- That means first the data is loaded bit by bit. The outputs are disabled as long as the loading is taking place.
- As soon as the loading is complete, and all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines simultaneously.
- Number of clock cycles required to load a four bit word is 4. Hence the speed of operation of SIPO mode is same as that of SISO mode.



(C-747) **Fig. 9.6.1 : Serial input parallel output mode**

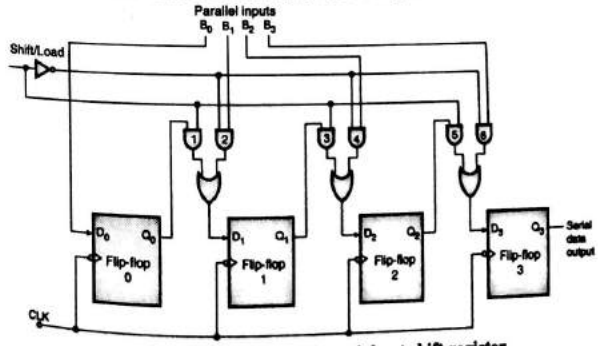
9.7 Parallel In Serial Out Mode (PISO) :

MU : Dec. 16

University Questions

Q. 1 Explain PISO and SIPO operations. **(Dec. 16, 5 Marks)**

- In this mode, the bits are entered in parallel i.e. simultaneously into a shift register as shown in Fig. 9.7.1.
- The circuit shown in Fig. 9.7.1 is a four bit parallel input serial output register.
- Output of previous FF is connected to the input of the next one via a combinational circuit.
- The binary input word B_0, B_1, B_2, B_3 is applied through the same combinational circuit.
- There are two modes in which this circuit can work namely shift mode or load mode.



(C-748) **Fig. 9.7.1 : Parallel in serial out shift register**

Load mode :

- In order to load the word B_3 B_2 B_1 B_0 into the shift register we have to select the load mode by setting shift/$\overline{\text{load}}$ input to 0.

Scanned by CamScanner

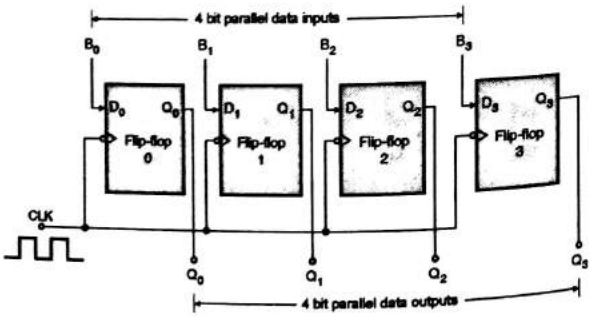
- When the shift / $\overline{load}$ line is low (0), the AND gates 2, 4 and 6 become active. They will pass B_1, B_2 and B_3 bits to the inputs D_1, D_2 and D_3 of the corresponding flip-flops. D_0 is directly connected to B_0.
- On the low going edge of clock, the binary inputs B_0 B_1 B_2 B_3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode :

- In order to operate the shift register in the shift mode we have to select the shift mode by applying a logic 1 to the shift/ $\overline{load}$ input.
- When the shift / $\overline{load}$ line is high (1), the AND gates 2, 4, 6 become inactive. Hence the parallel loading of the data becomes impossible.
- But the AND gates 1, 3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses becomes possible. D_0 acts as the data input terminal and at Q_3 we get the serial data output.
- Thus the parallel in serial out operation takes place.

9.8 Parallel In Parallel Out (PIPO) :

- Fig. 9.8.1 demonstrates the parallel in parallel out mode of operation.
- The 4 bit binary input B_0 B_1 B_2 B_3 is applied to the data inputs D_0, D_1, D_2 and D_3 respectively of the four flip-flops.
- As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously.
- The loaded bits will appear simultaneously to the output side. Only one clock pulse is essential to load all the bits. Therefore PIPO mode is the fastest mode of operation.



(C-749) **Fig. 9.8.1 : Parallel in parallel out shift register**

9.9 Bidirectional Shift Register :

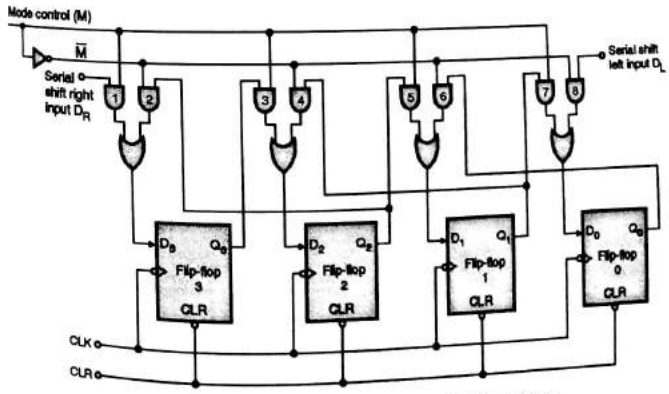
MU : May 04, Dec. 04, Dec. 05, Dec. 07, Dec. 10, Dec. 11, Dec. 14, Dec. 15

University Questions

Q.1 With the help of a neat diagram, explain the functioning of a 4 bit bidirectional shift register. **(May 04, Dec. 04, Dec. 05, Dec. 07, Dec. 10, Dec. 11, 5 Marks)**

Q.2 Explain 4 bit bi-directional shift register. **(Dec. 14, Dec. 15, 5 Marks)**

- If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2. On the other hand if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2. This is illustrated below.
- Let a four bit number Q_3 Q_2 Q_1 Q_0 = 0010 = $(2)_{10}$ is existing in a shift register. Now with a 0 applied at the input, if we shift these contents by one position to left then we get Q_3 Q_2 Q_1 Q_0 = 0100 = $(4)_{10}$. Thus the shift left is equivalent to multiplying by 2. Now shift it right by one position to get Q_3 Q_2 Q_1 Q_0 = 0001 = $(1)_{10}$. Thus shifting right is equivalent to dividing by 2.
- Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction as and when we want.
- Such a register is called as a bi-directional register. A four bit bi-directional shift register is shown in Fig. 9.9.1.
- There are two serial inputs namely the serial right shift data input D_R and the serial left shift data input D_L alongwith a Mode control input (M).



(C-750) **Fig. 9.9.1 : A 4-bit bi-directional shift register**

Scanned by CamScanner

Operation :

With M = 1 : Shift right operation

- If M = 1, then the AND gates 1, 3, 5 and 7 are enabled whereas the remaining AND gates 2, 4, 6 and 8 will be disabled.
- Hence the data at D_R (shift right input) is shifted right bit by bit from FF-3 to FF-0 on the application of clock pulses.
- Thus with M = 1 we get the serial right shift operation.

With M = 0 : Shift left operation

- When the mode control M is connected to "0" then the AND gates 2, 4, 6 and 8 are enabled while 1, 3, 5 and 7 are disabled.
- Therefore the data at D_L (shift left input) is shifted left bit by bit from FF-0 to FF-3 on application of the clock pulses.
- Thus with M = 0 we get the serial left shift operation.
- Note that M should be changed only when CLK = 0, otherwise the data stored in the register may get changed in an undesirable manner.

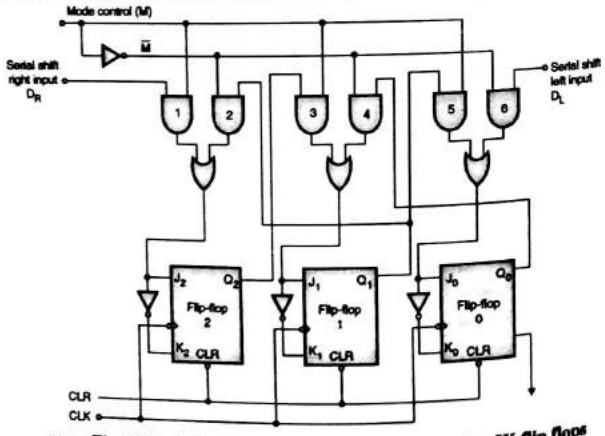
9.9.1 A 3-bit Bidirectional Register using the JK Flip Flops : MU : May 05, Dec. 13

University Questions

Q. 1 Write a short note on 3 bit bi-directional shift register. **(May 05, 10 Marks)**

Q. 2 Explain 3 bit bidirectional shift register using JK flip flop. Draw the neat waveforms. **(Dec. 13, 10 Marks)**

- A 3-bit bi-directional shift register using JK flip flops is shown in Fig. 9.9.2.



(C-751) **Fig. 9.9.2 : A 3 bit bi-directional shift register using JK flip flops**

- This circuit is same as the 4 bit bi-directional shift register discussed in the previous section.

- The only modification is the inclusion of an inverter between the J and K inputs of each flip flops.
- With M = 1 the shift right operation will take place as discussed in the previous section and with M = 0 the shift left operation is going to take place.

9.10 Universal Shift Register :

MU : Dec. 02, Dec. 08, May 11, May 12, Dec. 12, May 14, May 16, Dec. 16

University Questions

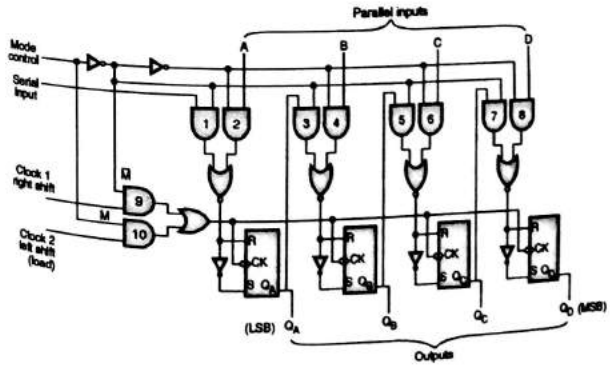
Q. 1 Draw the diagram of a 4 bit universal shift register and explain its operation. **(Dec. 02, Dec. 08, May 11, May 12, Dec. 12, 10 Marks)**

Q. 2 Explain the operation of 4-bit universal shift register. **(May 14, 10 Marks)**

Q. 3 Explain 4 bit universal shift register. **(May 16, 10 Marks)**

Q. 4 Draw a 4-bit universal shift register. **(Dec. 16, 5 Marks)**

- A shift register which can shift the data in only one direction is called as a unidirectional shift register.
- A shift register which can shift the data in both the directions is called as a bi-directional shift register.
- Applying the same logic, a shift register which can shift the data in both the directions (shift right or left) as well as load it parallely, then it is called as a **universal shift register.**
- Fig. 9.10.1 shows the logic diagram of a universal shift register.
- This shift register is capable of performing the following operations :
  1. Parallel loading (parallel input parallel output)
  2. Left shifting 3. Right shifting
- The mode control input is connected to Logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting.
- With mode control pin connected to ground, the universal shift register acts as a bi-directional register.



(C-752) **Fig. 9.10.1 : Logic diagram of a universal shift register**

Scanned by CamScanner

- For serial left operation, the input is applied to the serial input which goes to AND gate-1 in Fig. 9.10.1.
- Whereas for the shift right operation, the serial input is applied to D input (input of AND gate 8).
- The well known example of universal shift register in the IC form is IC7495.

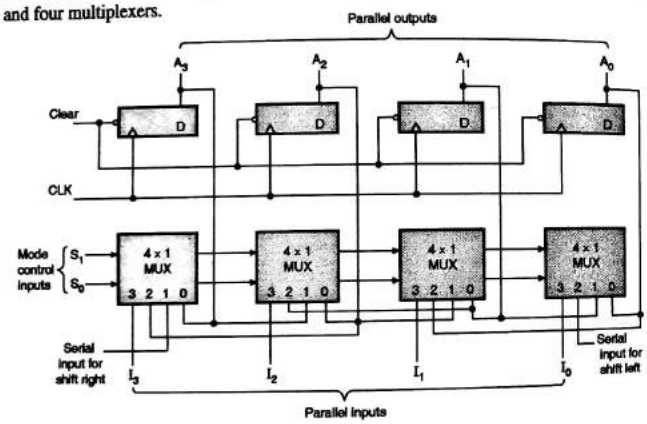
Ex. 9.10.1 : Draw and explain 4-bit shift register having shift left, shift right and parallel in facilities. Use suitable multiplexers in your circuit. Explain any one application of such register.

Soln. :

- Such a shift register is called as the universal shift register.

Universal shift register using multiplexers and D-flip flops :

- Fig. P. 9.10.1 shows the diagram of a 4 bit universal shift register. It consists of four D-flip flops and four multiplexers.



(C-753) **Fig. P. 9.10.1 : 4-Bit universal shift register**

Operation :

- The S_1 and S_0 inputs are the select lines of all the four multiplexers and they are connected together. These lines act as the **mode control inputs.**
- The mode of operation is dependent on the status of the S_1 and S_0 lines as shown in Table P. 9.10.1.

Table P. 9.10.1 : Function table of universal shift register

Mode Control S_1	S_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift Left
1	1	Parallel load

1. **Operation for $S_1 S_0 = 00$ (No change) :**
   - When $S_1 S_0 = 00$ input 0 of each multiplexer is selected. As the output of each FF is connected to input 0 of the corresponding MUX, the present value of the register is applied to D inputs of the flip flops.

   - On the next positive clock edge, this present value gets transferred to the output. That means, the next state will be same as the present state. Thus there is **no change** in state for $S_1 S_0 = 00$.

2. **Operation for $S_1 S_0 = 01$ (Right shift) :**
   - When $S_1 S_0 = 01$, terminal 1 of multiplexer input has a path to flip flop input D. This will cause the shift right operation.

3. **Operation for $S_1 S_0 = 10$ (Left shift) :**
   - When $S_1 S_0 = 10$, the input 2 of each multiplexer get connected to the D input of corresponding flip flop.
   - This will result in shift left operation.

4. **Operation for $S_1 S_0 = 11$ (Parallel load) :**
   - When $S_1 S_0 = 11$, the input 3 of each multiplexer gets connected to the D input of corresponding flip flop.
   - This will result in the parallel load operation.

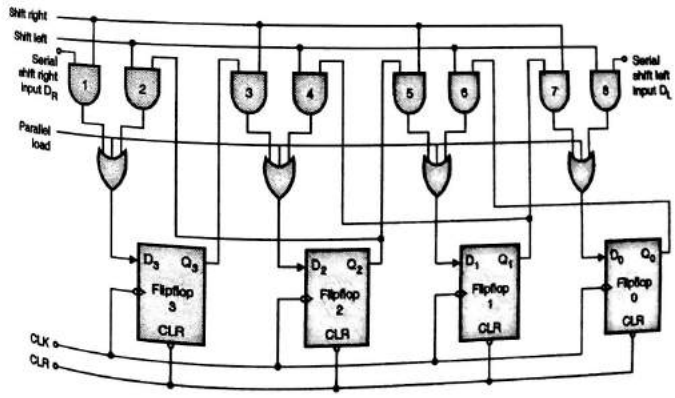
Ex. 9.10.2 : Design 4-bit shift register with the following facilities :

1. Shift-left (if L1 = 0 and L0 = 0)
2. Shift-right (if L1 = 0 and L0 = 1)
3. Parallel-in (if L1 = 1 and L0 = 1)

Control signals 'L1 L0' will select one of the possible operation.

Soln. :

The required universal shift register is shown in Fig. P. 9.10.2(a), whereas the combinational circuit required for the generation of operation selecting signals is shown in Fig. P. 9.10.2(b).



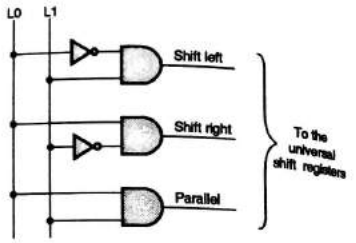
(C-3560) **Fig. P. 9.10.2(a) : Universal 4-bit shift register**

Scanned by CamScanner

Table P. 9.10.2

Inputs		Outputs		
L0	L1	Shift left	Shift right	Parallel
0	0	–	–	–
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

Control signals :



(C-3561) **Fig. P. 9.10.2(b) : Control logic**

9.11 Applications of Shift Registers :

- Some of the common applications of a shift register are :
  1. For temporary data storage.
  2. For multiplication and division.
  3. As a delay line.
  4. Serial to parallel converter.
  5. Parallel to serial converter.
  6. Ring counter.
  7. Twisted ring counter or Johnson counter.
  8. Serial data transmission.
- We have already seen the use of shift register for temporary storage of data and for multiplication or division. Let us now discuss the remaining applications one by one.

9.11.1 Serial to Parallel Converter :

- In many applications, the data in serial form needs to be converted in the parallel form.
- The Serial Input Parallel Output mode (SIPO) of the shift register is used for such applications.

9.11.2 Parallel to Serial Converter :

- The data communication between two computers takes place in the serial transmission form.
- But internally the data processing takes place in the parallel form.
- Hence we need to use a parallel to serial converter to convert the internal parallel data into a serial data suitable for transmission.
- We can use the parallel to serial mode of the shift register for this operation.

9.11.3 Ring Counter :

MU : May 03. May 08. May 12. Dec. 12. Dec. 13. Dec. 15

University Questions

Q. 1 Draw diagram of a 5-bit ring counter and explain its working with help of the timing diagram. **(May 03, 8 Marks)**

Q. 2 Design 4 bit ring counter using JK-FF, draw the timing diagram for the same. **(May 08, 10 Marks)**

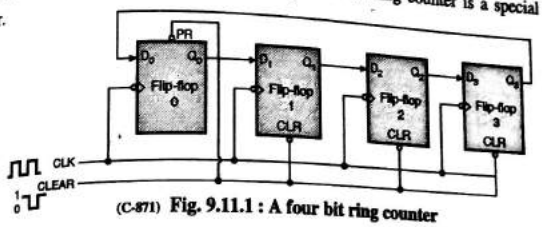
Q. 3 Draw a 4 bit ring counter. Draw the timing diagram and explain the working of counter. **(May 12, Dec. 12, 10 Marks)**

Q. 4 Write short note on ring counter using 'D' FF. **(Dec. 13, 5 Marks)**

Q. 5 Draw and explain the working of 4-bit ring counter with timing diagram. **(Dec. 15, 10 Marks)**

- Fig. 9.11.1 shows a typical application of shift registers called Ring Counter.

- The connections reveal that they are similar to the connections for shift right operation, except for one change.
- Output of FF-3 is connected to data input D_0 of FF-0. Ring counter is a special type of shift register.



(C-871) **Fig. 9.11.1 : A four bit ring counter**

Operation :

- Initially a low clear (CLR) pulse is applied to all the flip-flops.
- Hence FF-3, FF-2 and FF-1 will reset but FF-0 will be preset. So the outputs of the shift register are :

$$Q_3\,Q_2\,Q_1\,Q_0 = 0001.$$

- Now the clear terminal is made inactive by applying a high level to it.
- The clock signal is then applied to all the flip-flops simultaneously. Note that all the flip-flops are negative edge triggered.

On the first negative going CLK edge :

- As soon as the first falling edge of the clock hits, only FF-1 will be set because $Q_0 = D_1 = 1$.
- The FF-0 will reset because $D_0 = Q_3 = 0$ and there is no change in the status of FF-2 and FF-3.
- Hence after the first clock pulse the outputs are

$$Q_3\,Q_2\,Q_1\,Q_0 = 0010.$$

On the second falling edge of clock :

- At the second falling edge of the clock, only FF-2 will be set because $D_2 = Q_1 = 1$.
- FF-1 will reset since $D_1 = Q_0 = 0$. There is no change in status of FF-3 and FF-0.
- So after the second clock pulse the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0100.$$

- Similarly after the third clock pulse the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 1000.$$

- And after the fourth one the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0001.$$

- These are the outputs from where we started. Hence the operation repeats from this point onwards.

Number of output states :

The number of output states for a ring counter will always be equal to the number of flip-flops. So for a 4-bit ring counter the number of states is equal to 4.

Scanned by CamScanner

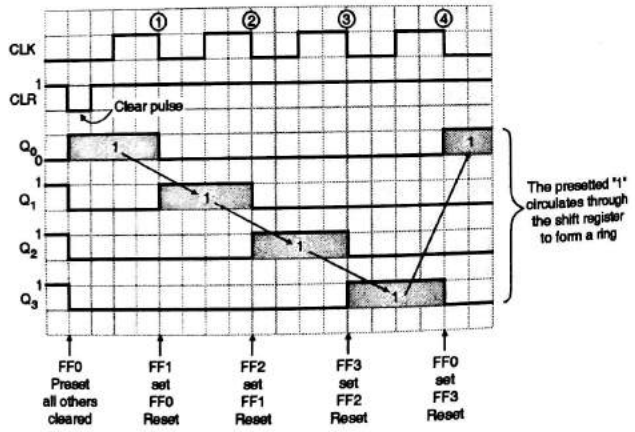
The operation of a four bit ring counter is summarized in Table 9.11.1.

(C-871(a)) **Table 9.11.1 : Summary of operation of a ring counter**

CLR	CLK	Q_0	Q_1	Q_2	Q_3	
⊓	X	1	0	0	0	← FF-0 Preset, others cleared
1	↓	0	1	0	0	The presetted 1 follows a circular path to form a ring
1	↓	0	0	1	0	
1	↓	0	0	0	1	
1	↓	1	0	0	0	

Waveforms for the ring counter :

- The waveforms for the 4-bit ring counter are as shown in Fig. 9.11.2.
- These waveforms clearly show that the presetted "1" shifts one bit per clock cycle and forms a ring. Hence the name ring counter.



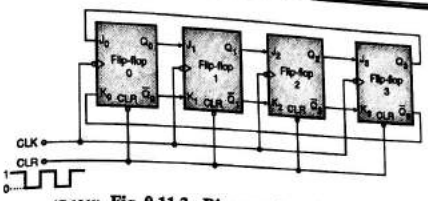
(C-1360) **Fig. 9.11.2 : Waveforms of a four bit ring counter**

Applications of ring counter :

Ring counters are used in those applications in which several operations are to be controlled in a sequential manner. For example in resistance welding the operations called squeeze, hold, weld and off are to be performed sequentially. We can use a ring counter to initiate these operations.

Ring counter using JK Flip Flop :

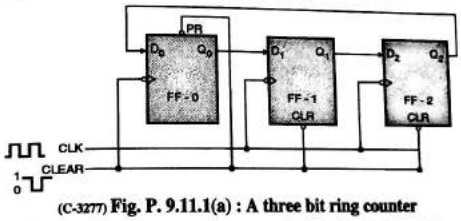
A ring counter can also be constructed using the JK flip flops, as shown in Fig. 9.11.3.



(C-1361) **Fig. 9.11.3 : Ring counter using JK flip flops**

Ex. 9.11.1 : Draw a 3 bit ring counter and draw timing wave form. Prove that it is a divide by 3 network. **May 11, 10 Marks**

Soln. : The 3 bit ring counter is as shown in Fig. P. 9.11.1(a).



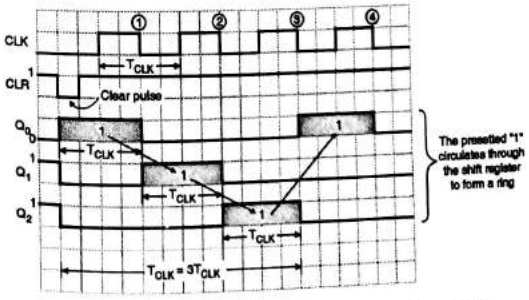
(C-3277) **Fig. P. 9.11.1(a) : A three bit ring counter**

The operation of a three bit ring counter is summarized in Table P. 9.11.1

(C-3278) **Table P. 9.11.1 : Summary of operation of a ring counter**

CLR	CLK	Q_0	Q_1	Q_2
⊓	×	1	0	0
1	↓	0	1	0
1	↓	0	0	1
1	↓	1	0	0

Waveforms : The waveforms of a 3-bit ring counter are as shown in Fig. P. 9.11.1(b).



(C-3279) **Fig. P. 9.11.1(b) : Waveforms of a 3-bit ring counter**

Scanned by CamScanner

Compare the time periods of waveforms at Q_2 and CLK.

$$T_{out} = 3\,T_{CLK} \qquad \therefore \quad f_{out} = \frac{1}{3} f_{CLK} \qquad \text{...Proved.}$$

9.11.4 Johnson's Counter (Twisted / Switch Tail Ring Counter) :

MU : Dec. 02, Dec. 03, May 09, Dec. 09, May 14, May 15, Dec. 16

University Questions

Q. 1 Draw the diagram of a 4 bit twisted ring counter, explaining its operation with the help of a timing diagram. **(Dec. 02, 6 Marks)**

Q. 2 Draw and explain 4 bit Johnson's counter. **(Dec. 03, 8 Marks)**

Q. 3 Draw a 4-bit Johnson's counter using shift register and prove that it is "Divide by 4" logic. **(May 09, 10 Marks)**

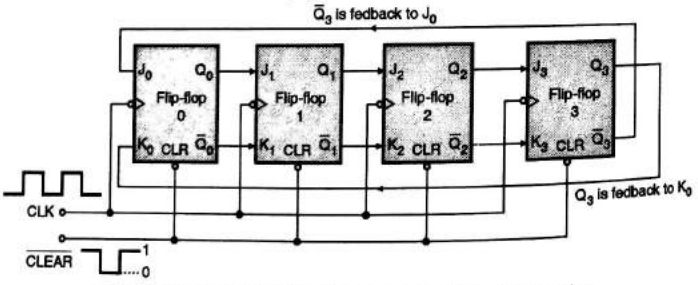
Q. 4 Draw a twisted ring counter and prove that it is "Divide by 2 N" circuit, where 'N' is number of Flip flops. Show necessary timing diagrams. **(Dec. 09, 10 Marks)**

Q. 5 Explain 4-bit Johnson counter. Draw its timing diagram. **(May 14, 10 Marks)**

Q. 6 Write short note on Johnson ring counter. **(May 15, 6 Marks)**

Q. 7 Draw and explain the working of 4-bit twisted ring counter with timing diagram. **(Dec. 16, 10 Marks)**

- In the ring counter the outputs of FF-3 were connected directly to the inputs of FF-0 i.e. Q_3 to J_0 $\overline{Q}_3$ to K_0.
- Instead if the outputs are cross coupled to the inputs i.e. if Q_3 is connected to K_0 and $\overline{Q}_3$ is connected to J_0 then the circuit is called as twisted ring counter or Johnson's counter.
- The Johnson's counter is shown in Fig. 9.11.4.



(C-1362) **Fig. 9.11.4 : Twisted ring counter or Johnson counter**

- All the flip-flops are negative edge triggered, and clock pulses are applied to all of them simultaneously.
- The clear inputs of all the flip-flops are connected together and connected to an external clear signal. Note that all these clear inputs are active low inputs.

Operation :

- Initially a short negative going pulse is applied to the $\overline{\text{clear}}$ input of all the flip-flops. This will reset all the flip-flops. Hence initially the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0000$$

- But $\overline{Q}_3 = 1$ and since it is coupled to J_0 it is also equal to 1.

$$\therefore \quad J_0 = 1 \text{ and } K_0 = 0 \quad \text{.... Initially}$$

On the first falling edge of clock pulse :

- As soon as the first negative edge of clock arrives, FF-0 will be set. Hence Q_0 will become 1.
- But there is no change in the status of any other flip-flop.
- Hence after the first negative going edge of the clock the flip-flop outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0001$$

On the second negative going clock edge :

- Before the second negative going clock edge, $Q_3 = 0$ and $\overline{Q}_3 = 1$. Hence $J_0 = 1$ and $K_0 = 1$. Also $Q_0 = 1$. Hence $J_1 = 1$.
- Hence as soon as the second falling clock edge arrives, FF-0 continues to be in the set mode and FF-1 will now set. Hence Q_1 will become 1 and $\overline{Q}_1 = 0$.
- There is no change in the status of any other flip-flop.
- Hence after the second clock edge the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0011$$

- Similarly after the third clock pulse, the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 0111$$

- And after the fourth clock pulse, the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 1111$$

- Note that now $\overline{Q}_3 = 0$ i.e. $J_0 = 0$ and $K_0 = 1$.
- Hence as soon as the fifth negative going clock pulse strikes, FF-0 will reset. But the outputs of the other flip-flops will remain unchanged. So after the fifth clock pulse, the outputs are,

$$Q_3\,Q_2\,Q_1\,Q_0 = 1110 \quad \text{..... after the } 5^{th} \text{ clock pulse}$$

- This operation will continue till we reach the all zero output state. (i.e. $Q_3\,Q_2\,Q_1\,Q_0 = 0000$).
- The operation of Johnson's counter is summarised in Table 9.11.2.

Table 9.11.2 : Summary of operation of Johnson's counter

CLEAR	CLK	Q_3	Q_2	Q_1	Q_0	State number	Decimal equivalent
↓	Initially	0	0	0	0	1	0
1	↓	0	0	0	1	2	1
1	↓	0	0	1	1	3	3
1	↓	0	1	1	1	4	7

Scanned by CamScanner

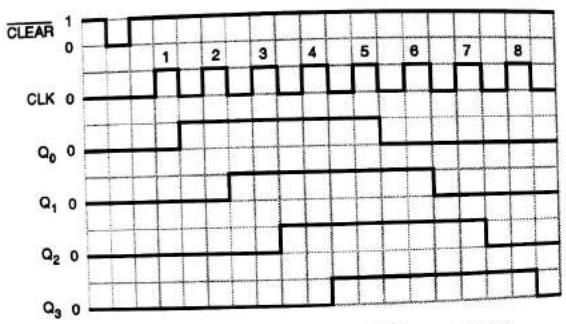
$\overline{CLEAR}$	CLK	Q_3	Q_2	Q_1	Q_0	State number	Decimal equivalent
1	↓	1	1	1	1	5	15
1	↓	1	1	1	0	6	14
1	↓	1	1	0	0	7	12
1	↓	1	0	0	0	8	8
1	↓	0	0	0	0	1	0

- Note that there are 8 distinct states of output.

In general we can say that the number of states of a Johnson's counter is twice the number of flip-flops used. Therefore for a 4-flip-flop Johnson's counter, there are 8-distinct output states.

Waveforms for Johnson's counter :

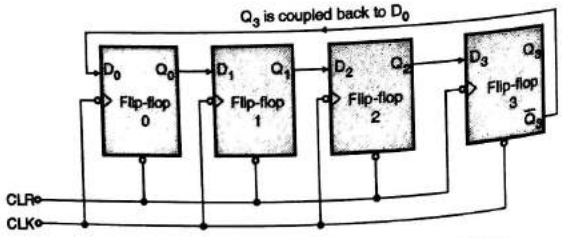
The waveforms for a 4-bit Johnson's counter are shown in Fig. 9.11.5.



(C-875) **Fig. 9.11.5 : Waveforms of Johnson counter**

9.11.5 Johnson's Counter using D Flip-flops :

It is possible to construct the Johnson counter using D flip-flops as shown in Fig. 9.11.6.



(C-1364) **Fig. 9.11.6 : Johnson counter using D flip-flops**

The operation of this circuit is summarized in Table 9.11.3.

Table 9.11.3 : Summary of operation of Johnson counter

CLR	CLK	Q_0	Q_1	Q_2	Q_3
⊓		1	0	0	0
1	↓	1	1	0	0
1	↓	1	1	1	0
1	↓	1	1	1	1
1	↓	0	1	1	1
1	↓	0	0	1	1
1	↓	0	0	0	1
1	↓	0	0	0	0
1	↓	1	0	0	0

9.11.6 Sequence Generator :

- A sequence generator is a sequential circuit which generates a desired sequence at its output. The output sequence is in synchronization with the clock input.
- It is possible to design a sequence generator using counters or using the shift registers.
- The available sequence is applied to the input of the detector. It checks the sequence bit by bit. If required bit is at its input then the detector moves to the next state.
- Detector output will be equal to zero as long as the complete sequence is not detected. The output will be equal to 1 if the complete sequence is detected.

(C-759) **Fig. 9.11.7 : Block schematic of a sequence detector**

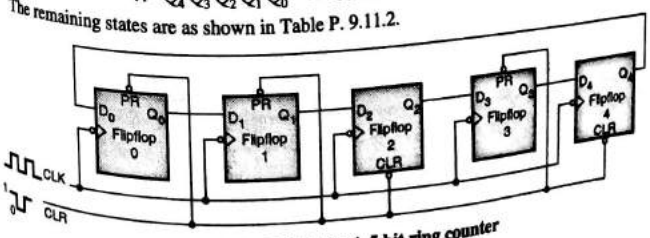
Ex. 9.11.2 : Explain ring counter design having initial state 01011 from initial state explain all possible states in the ring.

Soln. :

- Since there are 5 bits in the given initial state, we have to use 5 flip flops as shown in Fig. P. 9.11.2.
- When we apply a low going clear (CLR) pulse, then flip flops 0, 1, B and 3 will be preset to 1 output while flip flops 2 and 4 are reset to 0 output.

$$\therefore \quad Q_4 Q_3 Q_2 Q_1 Q_0 = 01011 \quad \text{...Initially}$$

The remaining states are as shown in Table P. 9.11.2.



(C-3562) **Fig. P. 9.11.2 : A 5 bit ring counter**

Scanned by CamScanner

(C-3563) **Table P. 9.11.2 : Ring counter states**

CLR	CLK	Q_0	Q_1	Q_2	Q_3	Q_4
⊓	X	1	1	0	1	0
1	↓	0	1	1	0	1
1	↓	1	0	1	1	0
1	↓	0	1	0	1	1
1	↓	1	0	1	0	1
1	↓	1	1	0	1	0

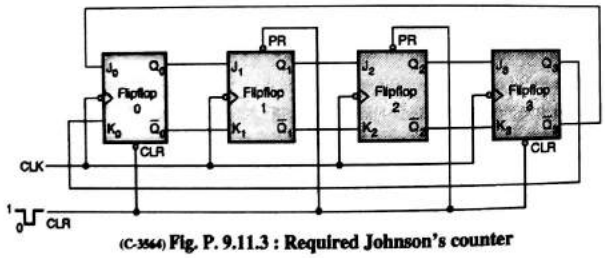
FFs 0, 1 and 3 are preset

Ex. 9.11.3 : Explain the Johnson's counter design for initial state 0110. From initial state explain and draw all possible states.

Soln. : The required Johnson's counter is shown in Fig. P. 9.11.3.

Counters 0 and 3 are reset to 0 while counters 1 and 2 and preset to 1 initially.

$$\therefore Q_3 Q_2 Q_1 Q_0 = 0\ 1\ 1\ 0 \quad \text{...initially}$$



(C-3564) **Fig. P. 9.11.3 : Required Johnson's counter**

The other possible states of this Johnson's counter are listed in Table P. 9.11.3.

(C-3565) **Table P. 9.11.3 : All possible states of a Johson's counter**

Clear	CLK	Q_3	Q_2	Q_1	Q_0
⊓	X	0	1	1	0
1	↓	1	1	0	1
1	↓	1	0	1	0
1	↓	0	1	0	0
1	↓	1	0	0	1

Review Questions

Q. 1 What is the function of a shift register ? Give its applications.

Q. 2 State the types of shift registers.

Q. 3 With a neat diagram explain the operation of 4-bit left shift register. Give its truth table and timing diagram.

Q. 4 With a neat diagram explain the operation of 4-bit SISO (Serial-In-Serial-Out) register. Draw the timing diagram and give its truth table.

Q. 5 With a neat diagram explain the operation of 4-bit Serial-In-Parallel Out (SIPO) register. Give the truth table and timing diagram.

Q. 6 With a neat diagram explain the operation of 4-bit Parallel-In-Serial-Out (PISO) register. Give the truth table and timing waveforms.

Q. 7 What is meant by "Universal shift register" ?

Q. 8 What do you understand by a bi-directional shift register ? Explain its operation.

Q. 9 List any one shift register IC. Sketch its schematic diagram and pin configuration. Give its specifications.

Q. 10 Draw circuit diagram of 3 bit SIPO shift register. Use D flip flops. Explain its working.

Q. 11 Define bi-directional shift register. Draw 3 bit bi-directional shift register using D flip flop.

□□□

Scanned by CamScanner

CHAPTER 10

Module 5

Introduction to VHDL

Syllabus :

Introduction : Fundamental building blocks Library, Entity, Architecture, Modeling Styles, Concurrent and sequential statements, simple design examples for combinational circuits and sequential circuits.

10.1 Introduction to VHDL :

MU : Dec. 14, May 15, May 16

University Questions

Q. 1 Write short note on : VHDL. **(Dec. 14, May 15, May 16, 7 Marks)**

- The long form of VHDL is Very High Speed Integrated Circuit (VHSIC) hardware description language.
- **VHDL** is hardware description language used to form a digital system at many levels of ideas ranging from the algorithmic to the gate level.
- VHDL language defines the syntax as well as simulation semantics for each language. It is strong typed language which frequently contains too many words to write.
- VHDL is difficult to understand because it provides wide range of modeling capabilities but without learning the more complex features it is possible to incorporate a core subset of language which is simple and easy to understand.
- VHDL is an official IEEE standard which has become the industry standard language for explaining digital circuits.
- The original standard for VHDL called IEEE 1076 was accepted in 1987.

10.1.1 Features of VHDL :

1. **Concurrency :** VHDL is concurrent language which executes statements simultaneous in parallel.
2. **Supports sequential statement :** VHDL can execute one statement at a time in sequence only.
3. VHDL supports for test and simulation of programs.
4. **Strongly typed language :** Only LHS and RHS operators of same type are allowed in VHDL.
5. **Supports hierarchies :** Using VHDL, hierarchy can be represented. e.g. full adder, in this case it is composed of half adder and OR gate.
6. VHDL supports different types of modeling :
   1. Structural 2. Data flow 3. Behavioral 4. Mixed
7. VHDL supports synchronous and asynchronous models.
8. VHDL can be used as communication medium between different CAD and CAE tools.

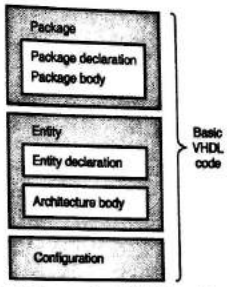
10.2 Structure of VHDL Module / VHDL Design Units :

MU : Dec. 14, May 15, May 16

University Questions

Q. 1 Write short note on : VHDL. **(Dec. 14, May 15, May 16, 7 Marks)**

- Design units of VHDL code are independent components which are separately compiled and stored in library.
- VHDL program is composed of following design units :
  1. Package (Optional) 2. Entity
  3. Architecture 4. Configuration (Optional)
- From the above design units, entity and architecture are necessary and others are optional.
- Design units of VHDL code is as shown in Fig. 10.2.1.



(C-5801) **Fig. 10.2.1 : Design units of VHDL code**

- In the following subsections we will discuss each design unit in detail.

10.2.1 Package :

- A VHDL package is a method to store and share some declarations which are common across many design units.
- Package is optional design unit. A set of declarations included in a package declaration can be shared by many design units.
- A set of useful type declarations, functions and procedures are declared by using the package and package body units.
- A package is represented by using package declaration and package body.

```
Package      package_name is
             (declarations)
endpackage _name ;
```

Fig. 10.2.2 : Package declaration

```
Package body     package_name is
                 [Function and procedure descriptions]
endpackage_name ;
```

Fig. 10.2.3 : Package body

- In a package declaration all declarations (type, subtype, subprogram) are encapsulated and package body contains behavior and definitions of subprograms declared in package declaration.
- A package declaration is necessary part while package body is mandatory when one or more subprograms are declared in package declaration.
- The name of package and package body should be same.

Scanned by CamScanner

- As the entity configuration and package declarations can be compiled independently they are called as **'primary design units'**, while architecture and package bodies are compiled with reference to primary design unit. They are **"Secondary design units"**.

10.2.2 Entity :

- The basic unit of VHDL hardware design is **entity.** The entity describes the interface between design and external environment.
- A VHDL entity states the parameters such as name and port of entity. All VHDL designs are generated using one or more entities.
- An entity is represented by using an **entity declaration** and an associated architecture body.
- An entity described input and output ports of a design.

10.2.2.1 Entity Declaration :

- The entity declaration defines the entity name and lists the input and output ports.
- Fig. 10.2.4 shows syntax of entity.

```
entity entity_name is
    port ( port_name : mode port_type;
           port_name : mode port_type);
end [ entity_name ] ;
```

Fig. 10.2.4 : The general form of an entity declaration

- In entity declaration, an entity starts with keyword **entity** which is followed by **name of entity** and keyword **is.**
- In entity declaration, names and identifiers can include letters, numbers and underscore (_) character, but it must starts with alphabet characters.
- In the next line port declarations are reserved with keyword **port.**
- The keyword **end** is always used to end an entity declaration which is followed by entity name. Here we can use statement end entity.
- VHDL language is not sensitive to white spaces, due to this alignment shown in entity syntax is not necessary. It is used for documentation.
- The communication between the entity and other models which occurs in its external environment takes place through signal is known as **ports.**

10.2.2.2 Port Modes in VHDL :

- In VHDL program, there are four modes available for ports.
- Table 10.2.1 shows port modes and its purpose in VHDL program.

Table 10.2.1 : Port modes

Sr. No.	Mode	Purpose
1.	In	Used for variable / signal i.e. an input to entity. This mode can read value, but they can't allow assignment of value to it. **In port** is unidirectional.

Sr. No.	Mode	Purpose
2.	Out	Used for signal i.e. an output from an entity. This mode can only write the value, they can't read a value from it. **Out port** is unidirectional.
3.	Inout	Bidirectional port. Both assignments to ports and read from port is allowed in **inout.**
4.	Buffer	Used to indicate that signal is an output of entity whose value can be read inside the entity architecture.

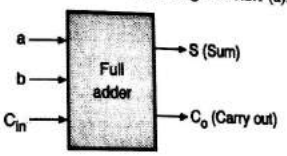
Type :

Type is user defined type of signal. e.g. bit, bit_vector, Boolean, Character, Std_logic, Std_ulogic.

Ex. 10.2.1 : Write down entity declaration for full adder circuit.

Soln. :

Block diagram of full adder circuit is as shown in Fig. P. 10.2.1 (a).



(C-5802) **Fig. P. 10.2.1(a) : Block diagram of full adder**

Where $S = C_{in} \oplus a \oplus b$, $C_o = (ab) + (a\,C_{in}) + (b\,C_{in})$

Entity declaration for full adder is as shown in Fig. P. 10.2.1 (b).

```
entity    FULLADDER is
port      (a, b, C_in    :   in bit;
           S, C_o        :   out bit);
end  FULLADDER ;
```

Fig. P. 10.2.1(b) : Entity declaration

- Refer Fig. P. 10.2.1(a). There are five interface ports in FULLADDER. a, b and C_{in} are input ports indicated by keyword **in.** S and C_o are output ports represented by keyword **out.** The **type** bit is selected for the signals going through these ports.
- The type bit represents the binary logic values which consists of two characters 0 and 1.

10.2.3 Architecture Body :

- An architecture describes the internal organization or entity operation and it includes the statements which are used to model the entity behaviour.
- Architecture body is used to explain internal details of a design entity using any one of the following modeling styles :
  1. Structural style : As a set interconnected components
  2. Dataflow style : As a set of concurrent assignment statements.

Scanned by CamScanner

3. Behavioral style : As a set of sequential assignment statements.
4. Mixed style : Any combination of the above three.

- In VHDL, it is allowed to develop many architecture bodies for same entity declaration and after that for synthesis select only one body to bind with the entity.
- Fig. 10.2.5 shows the syntax of architecture.

```
architecture architecture_name of entity_name is
    { block_declarative_item }
begin
    { concurrent_statement }
end [ architecture_name ] ;
```

Fig. 10.2.5 : The general form of an architecture

- For a VHDL design some prefer dataflow while others can prefer structural model.
- Architecture bodies includes two parts :
  1. **Declarative part** which contains mandatory declarations. e.g. Type declarations, signal declarations, component declarations, subprogram declarations.
  2. **Statement part** which contains statements which explains organizations or operation of component.
- Architecture cannot be used without an entity.
- In the same architecture different types of statements such as, processes, blocks, concurrent signal assignments, component instantiations can be used.

Ex. 10.2.2 : Write down architecture for full adder circuit.

Soln. :

Fig. P. 10.2.2 shows architecture body for full adder circuit.

```
architecture    full adder_arch of full_adder is
  begin
      sum (S) <= a xor b xor C_in;
      carry (C_o) <= (a and b) or (a and C_in) or (b and C_in) ;
  end full adder _ arch
```

Fig. P. 10.2.2 : Architecture body

- The first part between keywords **architecture and begin** is known as declaration part.
- The second part between **begin** and **end** contains the statements, assignments and structure.
- For the end of architecture, keyword **end** is used.

10.2.3.1 Configuration Declaration :

- For project organization and configuration management configuration declarations are used in VHDL program.
- The configuration statement denotes the binding between the entity and architecture. It is used to select any one of architecture from multiple architecture bodies and bind it to corresponding entity.

- In VHDL design, configuration allows you to change between different architectures for an entity.
- In a VHDL design configuration statement is optional.
- Multiple configurations may available in an entity.
- Component declaration for full adder is as shown in Fig. 10.2.6.

```
Component full_adder
    port    (a, b, C_in  :  in bit;
             S, C_o      :  out bit);
end full_adder ;
```

Fig. 10.2.6

10.2.4 Important Points to Remember while Designing any Module using VHDL :

1. In VHDL, each statement is terminated with semicolon (;).
2. The entity name must start with an alphabet and can include underscore (_).
3. The uppercase and lowercase letters are treated as same because VHDL language is case insensitive.
4. The architecture body should starts with word **begin.**
5. The port name must be followed by colon (:).
6. VHDL allows the blank space between two words or at the beginning of the line.
7. In VHDL module, leaving the blank lines are allowed.
8. Comment in VHDL must begin with two hyphens (--).

10.3 Library :

- A storage of compiled VHDL codes is known as VHDL library.
- The case where the user can create library is called as **"user library".**
- Two line code is required for library declaration, one contains library clause and another contains use clause.
- Fig. 10.3.1 shows syntax of library.

```
LIBRARY    Library name;
  USE      Library_name_package_name.package_parts;
```

Fig. 10.3.1 : Syntax of library

- If ports are of **bit** type, then it is not necessary to include IEEE library in program. If ports are of type **std_logic** then it must include IEEE library.
- The library clause is used for visibility of the logical names of design libraries that can be located within a design unit.
- The design units present in the library can't be made visible by the library clause. For every design unit library STD, WORK is declared by default.
- Following are three packages from different libraries which are usually required in a design.
  1. ieee.std_logic_1164 (from the ieee library)
  2. Standard (from the std library)
  3. Work (work library).

Scanned by CamScanner

Their declarations are as follows :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY std;
USE std.standard.all;
LIBRARY work;
USE work.all;
```

10.4 Identifiers :

- To identify different VHDL objects i.e. design entity, signals, procedures, processes, functions etc. a **VHDL identifier** is used.

Rules to form an identifier :

1. Identifier must contain only alphabets, decimal digits and underscores. (i.e. a to z, 0-9 and _).
2. It must start with alphabetic character (a to z or A to Z).
3. The last character may not be an underscore.
4. Two consecutive underscores are not allowed.
5. As VHDL language is case insensitive, the following identifiers are considered as same : count, COUNT, count, COUNT.
6. To mention comments in VHDL description, it must be followed by two consecutive hyphens (– –). In a description comments can appear anywhere.
7. A separate comment line should be given to continue a comment on a second line.

10.5 VHDL Operators :

- For construction of expressions, operators are needed. Expressions in VHDL and programming language are same.
- In VHDL language predefined operators are classified into five types :
  1. Logical operators.
  2. Relational operators.
  3. Adding operators.
  4. Multiplying operators.
  5. Miscellaneous operators.

10.5.1 Logical Operators :

- The six important logical operators are **AND, OR, NAND, NOR, XOR, XNOR.** Logical operators are defined for the types **Bit** and **Boolean** and also for one dimensional array of **bit** and **Boolean.**
- The result of a logical operation and its operands has the same type.

Table 10.5.1 : Logical operators

Operator	Type of operand	Type of Result
AND	Bit	Bit
OR	Bit	Bit
NAND	Bit	Bit
NOR	Bit	Bit
XOR	Bit	Bit
XNOR	Bit	Bit

10.5.2 Relational Operators :

- To create equality or magnitude comparison function, relational operators are used.

Table 10.5.2 : Relational operators

Operator	Description	Type of operand	Type of Result
=	Equality	any type	Boolean
/ =	inequality	any type	Boolean
<	less than	Scalar or discrete array type	Boolean
<=	less than or equal	Scalar or discrete array type	Boolean
>	greater than	Scalar or discrete array type	Boolean
>=	greater than or equal	Scalar or discrete array type	Boolean

10.5.3 Adding Operators :

- To create arithmetic functions, arithmetic operators are used.

Table 10.5.3 : Adding operators

Operator	Description	Operands (A or B) type	Type of result
+	Addition (A + B)	A : numeric B : numeric	numeric
–	Subtraction (A – B)	A : numeric B : numeric	numeric
&	Concatenation (A & B)	A : numeric or array B : numeric or array	same as A

10.5.4 Multiplying Operators :

Table 10.5.4 : Multiplying operators

Operator	Operation	Type of operands (A or B)	Type of results
*	Multiplication	A : integer / real B : integer / real	Same as A
*	Multiplication	A : physical B : integer / real	Same as A

Scanned by CamScanner

Operator	Operation	Type of operands (A or B)	Type of results
*	Multiplication	A : integer / real B : physical	Same as B
/	division	A : integer / real B : integer / real	Same as A
/	division	A : integer / real B : physical	Same as B
/	division	A : physical B : integer / real	Same as A
mod	modulus	A : only integer B : only integer	integer
rem	remainder	A : only integer B : only integer	integer

- rem is defined as, A rem B =A – (A / B) * B
- Mod is defined as, A mod B=A – B * N

e.g.

7 rem 3 = 7 – ((7/3) * 3) = 1

7 mod 3 = 7 – (3 * 1) = 4

10.5.5 Miscellaneous Operators :

Table 10.5.5 : Miscellaneous operators

Operator	Operation	Type of operand	Type of result
**	Exponentiation (A ** B)	A : real / integer B : only integer	Same as A
abs	absolute abs (A)	A : numeric	Positive numeric

e.g.

$$2 ** 8 = 256$$

$$4 ** (-2) = \frac{1}{(4 ** 2)} = \frac{1}{16} = 0.0625$$

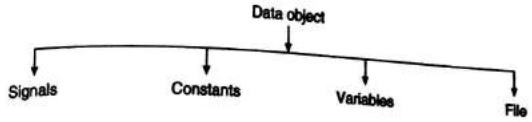
10.5.6 Precedence of Operators in VHDL :

1. Miscellaneous operators.
2. Multiplying operators.
3. Adding operators.
4. Relational operators.
5. Logical operators.

10.6 VHDL Data Objects :

- In VHDL code information is represented as data objects.
- The function of a data object is to hold a value of particular type.
  e.g. variable count : integer, constant count : integer.

10.6.1 Classes of Data Objects :



(C-5803) **Fig. 10.6.1 : Data object classes**

10.6.1.1 Signals :

- Signal holds a list of values which includes the current value and set of possible future values.
- Data object signal corresponds to wires, logic signals in a circuit. It has to be declared with a related type as follows :

  SIGNAL signal_name : type_name
- In signal declaration, the legal values that the signal can have and its legal uses in VHDL code can be determined by type_name.
- The different types of signals are : BIT, BIT_VECTOR, STD_LOGIC, SIGNED, UNSIGNED, STD_LOGIC_VECTOR, STD_ULOGIC, INTEGER, BOOLEAN and ENUMERATION etc.

  e.g. **SIGNAL** X : BIT
- In this example X is signal object which is of type BIT. 'O' is initial value of signal because leftmost value of this type BIT is 'O'.

10.6.1.2 Variable :

- For a given type, to hold a signal value a variable is used.
- A data object **variable** is used to hold calculated results and for the index variables in loops.
- The **variables** are also used for computations within functions, processes and procedures.
- A **variable** object holds different values at different times. For this variable assignment statement is used.
- The syntax for declaring variable is as follows :

  VARIABLE variable_name : variable_type : = initial value
- As **variable** can't store events, they don't have concept of memory.

  e.g. **VARIABLE** index : Integer range 0 to 30 : = 0;
- In this example, the variable index is an integer having range between 0 and 30 and it is initialized to '0'.
- To represent the behavior of the circuit variable and constants are used.

Scanned by CamScanner

10.6.1.3 Constant :

- In this type of data object, during the simulation time the value cannot be changed. In VHDL, the syntax for declaring a constant is given below :
  CONSTANT constant_name : type name : = constant value
- The main use of **constant** is to improve the code readability, which is achieved by using constant name in place of number or value.

10.6.1.4 File :

- This type of data object includes value of specific type sequentially. It provides an interface between VHDL programs and host environment.
- With the help of write and read process, values can be read from the file and written to the file.
- Following are the basic operations performed while reading and writing on files :
  1. File type declaration.
  2. File declaration.
  3. Open and close a file of a specified type.
  4. Read from a file and write to a file.

10.6.2 Comparison between Signal and Variable :

Table 10.6.1 : Comparison between signal and variable

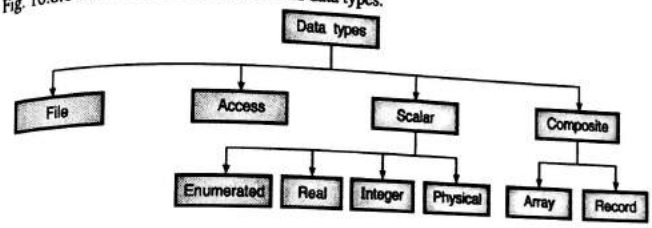
Parameter	Signal	Variable
Assignment	Must be scheduled.	Assignments are faster.
Memory	More memory required.	Requires less memory.
Scope	Can be global.	Local.
Use	In a package, entity, architecture.	Only in sequential code i.e. process, procedure, function.
Assignment	<=	:=
Example	**SIGNAL** X : BIT ;	**VARIABLE** index : Integer range 0 to 30 : = 0;

10.7 VHDL Data Types :

- VHDL language is called as **strongly typed language** i.e. an object can only be assigned a value of its type and on the object only the operations defined with the data type can be performed.
- In VHDL language, each object has a data type. By using type conversion functions or casting, objects of different base types can be assigned or compared to one another directly.

10.8 Classification of Data Types :

- Fig. 10.8.1 shows classification of VHDL data types.



(C-5804) **Fig. 10.8.1 : Classification of data types**

10.8.1 Scalar :

- Values belonging to scalar types are ordered. i.e. they appear in sequential order. Scalar type includes all numeric, enumerated and physical data object types. The numeric type consists of integer, floating point (real) and physical types. Scalar types holds single value. On these values relational operators can be used.
- As the values associated with integer, floating point (real) and physical types are numbers. These are classified as **numeric types** whereas integer and enumeration are **discrete types** because these types have discrete associated values.
- Every value in integer, physical and enumeration type has their associated **position number.**
- In the ordered list of values, this number is the location of the value.

10.8.1.1 Enumeration Types :

- An enumeration type declaration contains set of user defined values which includes **character literal** and **identifiers.**
- Within a single enumeration type there must be different characters and identifiers. They can be reused in many different enumeration types.
- To represent exact value required for specific operation a designer can use an enumerated type.
- An identifier is like a name e.g. ABC, black, X etc. Character literals are single character enclosed in quotes e.g. 'x', '1', '0'.
- Fig. 10.8.2 shows syntax for declaration of enumeration.

```
enumeration_type_declaration :: =
type identifier is (enumeration_literal { enumeration_literal })
enumeration_literal :: = identifier / character_literal
```

Fig. 10.8.2

- Example of enumerated type is, Type colour is (red, blue, yellow).

Scanned by CamScanner

10.8.1.2 Integer Types :

- An integer type is a range of integer values within a specified integer range.
  e.g. type index **is range** 0 **to** 15.
- The integer type must at least cover the range
  $-(2^{31}-1)$ to $+(2^{31}-1)$

10.8.1.3 Physical Types :

- A physical type is used to represent measurement of physical quantity. e.g. voltage, power, current, length, time etc.
- Values of physical types are declared as integer multiples of a base unit. To specify finite quantities a physical type is used.
- Syntax of physical type declaration is as shown in Fig. 10.8.3.

```
Units
    base_unit_declaration
    {secondary_unit_declaration}
end units
```

Fig. 10.8.3

```
e.g. type resistance is range 0 to 1 E 8
units
    ohms;
    kohms = 1000 ohms,
    mohms = 1E6 ohms
end units;
```

10.8.1.4 Real (Floating Point) Type :

- Floating point types are set of positive and negative numbers (i.e. Red) which contains a decimal point.
- Syntax for real type is as follows :
  Type type_name **is** range value ;
  e.g. **Type** REAL_DATA **is range** 0.0 **to** 31.9

10.8.2 Composite Types :

- A collection of values are represented by composite data type.
- Two classes of composite types are :
  1. Array 2. Record
- An **Array** is a collection of values which belongs to a same type. **Record** is a collection of values which belongs to a same or different type. An element of composite type can be a scalar, access or composite type.

10.8.2.1 Array Types :

- An array is collection of objects that have the same subtype. Array can be one-dimensional (with one index) or multidimensional (with a number of indices). By specifying the index values into the array, elements of an array can be accessed.

Syntax :

Type Type_name **is array** values ;

e.g.

Type address_word **is array** (0 to 63) of bit ;

10.8.2.2 Record Types :

- Record is heterogeneous composite type i.e. same or different types of elements are included in the record type. Elements in the record type are accessed through field name.
- The record type is equivalent to **struct declaration** in C and record data type in **Pascal.**

```
e.g.
    type DATE is
        record
            DAY : integer range 1 to 31
            MONTH : Month_Name
    end record ;
```

10.8.3 Access Types :

- The values which belongs to an access type are pointers to dynamically allocated object of some other types. They are analogous to pointers in Pascal and C languages.

e.g.

1. **type** FIFO **is array** (0 **to** 63) of BIT ;
2. **type** ptr **is access** date ;

In this example ptr is access type whose values are addresses that point to object of type date.

10.8.4 File Types :

- File types are used for object representation of file in the host environment. The value of file object is sequential values included in the host file.
- **Syntax :**
  type file-type-name **is file of** type-name.

10.9 Modeling Styles in VHDL :

- In VHDL, there are different ways of describing combinational circuits. These are based on the modeling styles used for describing the circuit.

Modeling styles :

1. Structural modeling / Gate level modeling.
2. Dataflow modeling.

Scanned by CamScanner

3. Behavioral modeling.
4. Mixed modeling.

A circuit can be described by using any one of the modeling style or combination of modeling styles. In the following subsections we will discuss modeling styles one by one.

10.9.1 Structural Modeling / Gate Level Modeling :

- In this style of modeling, an entity is explained as a set of interconnected components.
- Structural modeling style shows the graphical representation of modules, components with their interconnection.
- Structural modeling can be used to generate very high level or very low level description of a circuit.
- In structural modeling, architecture part has two components :
  1. Declaration 2. Instantiation.
- In **declaration** part, various components which are used in the system are declared.

  e.g. OR gate component is declared as follows :

```
Component  OR2
    Port (a1, b1 : in bit ; Z1 : out bit);
end  component ;
```

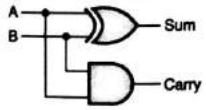
- The OR2 component has two inputs a1 and b1 and one output Z1. We can use the same component one or more times, once the component is declared.
- In the **instantiation** part, mapping of code takes place. i.e. The code maps generic inputs / outputs to the actual inputs or outputs of the system.

  e.g. OR2 port map (A, B, out); i.e. A maps to input a1 of OR2, input B maps to input b1 of OR2 and output out maps to Z1 of OR2.
- Ex. 10.9.1 shows structural description of half adder.

Ex. 10.9.1 : Write down VHDL code for half adder using structural modeling.

Soln. :

Fig. P. 10.9.1 shows the circuit diagram of half adder.



(C-5805) **Fig. P. 10.9.1 : Half adder circuit**

```
entity half_adder is
        port (A, B : in bit ; Sum, carry : out bit);
End half_adder ;
Architecture Structure of half_adder is
Component XOR1
                Port (X, Y : in bit ; S : out bit) ;
end component ;
```



```
Component AND 1
                Port (M, N : in bit ; C : out bit);
end component ;
begin
        V1 : XOR1 port map (A, B, Sum) ;
        V2 : AND1 port map (A, B, Carry) ;
end structure ;
```

10.9.2 Dataflow Modeling :

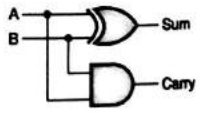
- In dataflow modeling, the dataflow through the entity is expressed using concurrent signal assignment statements.

Concurrent signal assignment statements :

- **Concurrent signal assignment statements** are statement which appear outside of a process.
- These statements are **event triggered.** i.e. in the expression whenever an event on signal appears, these statements are executed.
- Dataflow style of modeling describes how data is transferred from signal to signal and input to output. In a signal assignment, for assignment of a value to a signal the symbol < = is used.
- On the right hand side of statement, the value of the expression is calculated and is assigned to left hand side signal is known as **Target signal.** The ordering of concurrent statements in an architecture body is not important.
- Let us see the dataflow VHDL description of half adder.

Ex. 10.9.2 : Write down VHDL code for half adder using dataflow style of modeling.

Soln. : Fig. P. 10.9.2 shows the circuit diagram of half adder.



(C-5805) **Fig. P. 10.9.2 : Half adder circuit**

```
entity Adder is
        port (A, B : in bit ; sum, carry : out bit);
end Adder ;
Architecture half_adder of adder is
begin
        Sum < = A XOR B;   ... Signal assignment statement
        Carry < = A and B;  ... Signal assignment statement
end half_adder ;
```

- In this example, the statements between keyword begin and end are the concurrent signal assignment statements.

Scanned by CamScanner

- The signal on the right hand side of symbol <= is **Target signal.**
- In the signal assignment statements, delay information is added by using **after** clause in the architecture body in the following way :

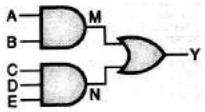
```
Architecture half_adder of adder is
begin
    Sum <= A XOR B        after 2 nS;
    Carry <= A and B      after 4 nS;
end half_adder;
```

- In this program, suppose at time 'T', either signal A or B which are input port signals of half_adder entity, has an event, both signal assignment statements on the right hand side are evaluated.
- Value of sum will be evaluated at time (T + 2) nS and value of carry will be evaluated at time (T + 4) nS. This will be repeated after every 2 nS and 4 nS respectively. Thus both signal assignment statements execute parallely / concurrently.

Ex. 10.9.3 : Write down dataflow description of AND_OR circuit.

Soln. :

Fig. P. 10.9.3 shows AND-OR circuit in which A to E are input signals, Y is an output signal, M and N are intermediate signals.



(C-5806) **Fig. P. 10.9.3 : AND_OR circuit**

```
entity AND-OR is
        port (A, B, C, D, E : in bit; Y : out bit);
end AND-OR;
architecture circuit of AND-OR is ;
signal M, N ;
begin
M <= A and B after 10 nS;    ... time 10 nS indicates the propagation delay of gate
N <= C and D and E after 10 nS; ... and  word after is used to specify propagation delay

Y <= M or N after 20 nS ;
end circuit;
```

10.9.3 Behavioral Style of Modeling :

- The behavioral level of abstraction is the highest level of abstraction supported in VHDL.
- **Process** is the keyword in the behavioral description.
- **Process** body must be included in every behavioral description. Process is concurrent statement which consists in an architecture body.

- The behavioral style of modeling denotes the entity behavior as a set of statements which executes sequentially.
- e.g. process (A, B) in this example the ports A and B are called as **sensitivity list.** The sensitivity list is treated as event if there is any change in the state of any element. The process is active only if an event occurs otherwise it is inactive. The execution of process is continuous if the process has no sensitivity list.
- Ex. 10.9.4 shows behavioral description of half adder.

Ex. 10.9.4 : Write down VHDL code for half adder using behavioral style of modeling.

Soln. :

```
entity adder is
    port (A, B : in bit ; sum, carry : out bit)
end half_adder ;
architecture half_adder of adder is
begin
    P1 : process (A, B)
begin
    Sum <= A XOR B ;
    Carry <= A and B ;
end process P1 ;
end adder ;
```

10.9.4 Mixed Style of Modeling :

In a single architecture body, we can mix the three modeling styles, i.e. within an architecture body to represent structure we can use component instantiation statements, to represent dataflow we can use concurrent signal assignment statements and to represent behavior, we can use process statements.

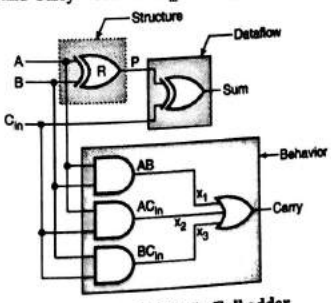
- Ex. 10.9.5 shows mixed style of modeling.

Ex. 10.9.5 : Write down VHDL code for full adder using mixed style of modeling.

Soln. :

We know that for full adder,

$$\text{Sum} = (A \oplus B) \oplus C_{in} \text{ and Carry} = AB + A\,C_{in} + B\,C_{in}$$



(C-5807) **Fig. P. 10.9.5 : Full adder**

Scanned by CamScanner

```
Entity fulladder is
    port (A, B, C_in : in bit ; sum, carry : out bit) ;
end fulladder ;
architecture FA_mix of fulladder is

component XOR1
    port (A, B : in bit ; Z : out bit) ;
end component ;
    signal P : Bit
begin
    R : XOR1 port map (A, B, P);
    Process (A, B, C_in)
        Variable (x1, x2, x3 : Bit);
begin
    x1 : = A and B ;
    x2 : = A and C_in ;
    x3 : = B and C_in ;
Carry < = x1 or x2 or x3 ;
end process ;
    sum < = P XOR C_in ;
end FA_mix ;
```

Structure

Behavior

... Dataflow

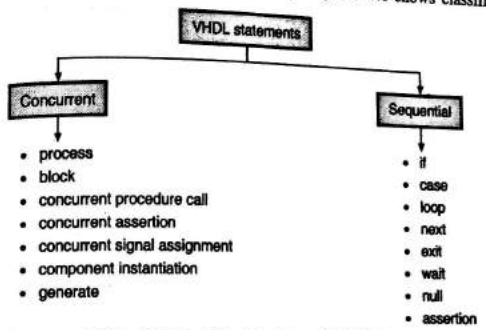
10.9.5 Comparison of Modeling Styles :

Sr. No.	Dataflow	Structural	Behavioral
1.	Dataflow modeling style shows that how the data / signal flows from input to output through the registers / components.	Structural modeling style shows the graphical representation of components with their interconnection.	Behavior modeling style shows that how our system performs according to current input values.
2.	Works on concurrent execution.	Works on concurrent execution.	Works on sequential execution.
3.	For design specifications, Boolean equations are needed.	For design specifications logical diagram is needed.	For design specifications truth table is needed.
4.	Level of abstraction is gate.	Level of abstraction is RTL (Resistor Transistor Logic).	Level of abstraction is gate.

10.10 VHDL Statements :

- In VHDL description two types of statements are used. These statements are as follows :
  1. Sequential statements 2. Concurrent statements
- Sequential statements execute in sequential manner because these are not event triggered.
- The behavior of sequential and concurrent statements is identical.

- In an architecture body, if a statement appears outside a process, it is a **concurrent statement** whereas if statement appears inside a process it is **sequential statement.**
- Sequential statements are useful for defining algorithms to express the behavioral of a design entity.
- Sequential statements appear only in process statement and subprograms.
- Concurrent statements in a design execute parallely. Fig. 10.10.1 shows classification of VHDL statements.



(C-5808) **Fig. 10.10.1 : Classification of VHDL statements**

10.10.1 Variable Assignment Statements :

To assign a value to a variable, variable assignment statements are used. We can use variables inside the process statement.

Syntax :

```
Variable_Object : = expression ;
```

- After the execution of statement, the expression is calculated and computed value is assigned to the variable object immediately with no delay time between evaluation and assignment.
- Instead of assignment operator (< =), : = is used to assign values to the variables.
- At the time of elaboration, variables are created and they holds its values for the entire simulation time because processes never exits or is either active or waiting for an event to occur and become active.
- Fig. 10.10.2 illustrates use of variables to store intermediary results for full adder.

```
entity fulladder is
    port (a, b, C_in : in bit ; sum, carry : out bit);
end fulladder ;
architecture variable of fulladder is
begin
    process (a, b, C_in)
        Variable p, q : bit ;
begin
    p : = a XOR b XOR C_in ;
```

Scanned by CamScanner

```
        q := (a and b) or (a and C_in) or (B and C_in) ;
    sum <= p;
    carry <= q ;
end process;
end variable;
```

Fig. 10.10.2

10.10.2 Signal Assignment Statements :

To assign a value to a signal, signal assignment statements are used.

Syntax :

Signal_object <= expression [after delay_value]

- Unlike a variable object, signal object is assigned the value after a delay. This delay is defined using a after clause or it will take the default value i.e. delta delay. Outside the process signal statement is a concurrent whereas when used inside a process it is sequential statement.
- **Delta delay** is infinitesimally small delay. During a simulation delta delay allows for ordering of events which can occur at the same simulation time. Each unit of simulation time is collection of an infinite number of delta delays. Signals use <= operator. In process, signals need to be global.

10.11 Concurrent Statements :

- The order of statements is not necessary in concurrent statements.
- All statements in VHDL are considered as concurrent. i.e. whenever there is change in the state of right hand side then statement is executed.

10.11.1 Process Statement :

- In behavioral style of modeling, a process statement which is **concurrent statement** is the main method used to explain the functionality of an entity.
- Sequential statements appears in a process statement which explains the functionality of a part of an entity in sequential terms.
- An independent sequential process represents the behavior of some part of a design. A process contains a list of sequential statements but process itself is a concurrent statement.
- The syntax of process statement is :

```
[process_label :] process [(sensitivity list)]
        [Local declarations]
begin
    sequential statements
end process [process_label] ;
```

10.11.2 Block Statement :

To represent designs in hierarchical manner, a block is a grouping of related concurrent statements.

Syntax :

```
Label : block (guard expression)
        block declarations ;
begin
        concurrent statements ;
end block lebel;
```

- The value of guard expression must be Boolean. To enable or disable the drivers, signal assignment statements present in block statement can use this **guard signal.**

10.11.3 Generate Statement :

- During the elaboration phase, the concurrent statements can be selected conditionally or simulated using the generate statement.
- Two forms of generate statements are :
  1. For generate 2. If generate.
- For replication of concurrent statement in a predetermined number of times, **for generate** scheme is used.
- For execution of concurrent statements which are conditionally selected, **if generate** scheme is used.
- For compact description of regular structures such as counters, registers, memories the generate statement is provided.

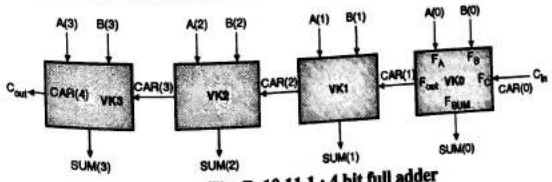
Syntax : for generate :

```
Label : for identifier in range generate
        (concurrent assignments)
end generate;
```

Syntax : if generate :

```
Label : if condition generate
        (concurrent assignments)
end generate;
```

Ex. 10.11.1 : Write VHDL code for 4-bit full adder using for generate statement.



(C-3628) **Fig. P. 10.11.1 : 4 bit full adder**

Soln.:

```
Library ieee;
Use ieee.std_logic_1164.all;
```

Scanned by CamScanner

```
entity full_add4 is
    port  (X,Y : in std_logic_vector (3 downto 0);
        Ci: in std_logic;
        S : out std_logic_vector (3 downto 0);
        COUT  out std_logic);
end full_add4;
architecture arch of full_add4 is
    component full_adder                -- name of entity of full adder component
        port (FC,FA,FB : in std_logic;
            FSUM,FCOUT : out std_logic);
    end component;
signal CAR: std_logic_vector (4 downto 0);
begin
    CAR (0)<=Ci;
    VK: for k in 3 downto 0 generate
    FA: full_adder port map (CAR (k),A (k),B (k),sum (k), CAR (k+1));
    end generate VK;
        COUT<=CAR (4);
    end arch;
```

10.12 Sequential Statements :

With behavioral description the various types of sequential statements are associated. These statements appers inside **process** in VHDL. These execute in sequential manner. Let us study these statements.

10.12.1 With Select when Statement :

- With select statement calculates each choice expressions and compares the value with each choice.
- Based on the value of select expression, the selected signal assignment statement selects various values for target signal.
- The statement is executed whenever an event occurs on a signal on the expressions.
- Assignment of corresponding expression to the target signal is based on the select expression which matches the specified value of choice.
- It is important to note that choices are not calculated in sequence.
- The choice **'others'** may cover values which are not covered explicitly.

Syntax :

```
With expression select
        Target signal <= Expression 1 When choice 1
                         Expression 2 When choice 2
                         Expression n When choice n
```


10.12.2 Wait Statement :

- Wait statement is an alternative for sensitivity list in a process.
- In a process, wait statement is used to suspend a process after last sequential statement.

Types of wait statements :

1. Wait on 2. Wait until 3. Wait for

1. **Wait on sensitivity list :**

   Until an extent occurs on one or more signals, this statement causes a process to suspend execution.

   e.g. Process ( )
   :
   Wait on A, B ;
   end process ;

2. **Wait until Boolean expression :**

   Until a given condition is not fulfilled, in this type of wait statement the process remains suspended.

   e.g. **Wait until** (A = B);

3. **Wait for time expression :**

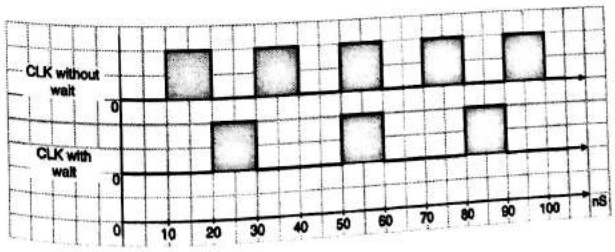
   In this type of statement the process remains suspended for the specified given time.

   e.g. **Wait for** 5 nS ;

Ex. 10.12.1 : Draw the waveform for the following VHDL code.

```
Process
CLK < = '0' ;
Wait for 20 nS
CLK < = '1' ;
Wait for 10 nS
end process
```

Soln. :



(C-5809) **Fig. P. 10.12.1 : Waveforms**

Scanned by CamScanner

10.12.3 Conditional Statements :

For some operations standard if ... else and case statements are used.

10.12.4 If Statement :

Based on the value of condition, an if statement selects sequence of statements for execution of process. The condition can be any expression related to Boolean value.

Syntax :

```
if (Boolean expression) then
      sequential statements ;
            ⋮
   else
      sequential statements ;
            ⋮
end if
```

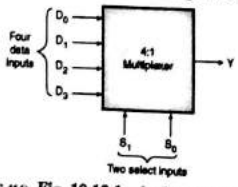
e.g.

```
if (E = '1') then
      Q : = S1;
else
      Q : = S2 ;
end if ;
```

- In this example if E = 1 (high) then value of S1 is assigned to Q otherwise value of S2 is assigned to Q.
- The general form of an if statement is :

```
if Boolean expression then
        sequential statements ;
   [elsif boolean expression then
        sequential statements ; ]
        [else
              sequential statements ; ]
end if ;
```

- A complete example of 4 : 1 MUX entity using if statement is as shown in Fig. 10.12.1.
- Fig. 10.12.1 shows the block diagram of a 4 : 1 multiplexer and Table 10.12.1 gives its truth table.



(C-416) **Fig. 10.12.1 : 4 : 1 multiplexer**

Table 10.12.1 : Truth table

Select inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3


```
entity MUX is
   port (D0, D1, D2, D3 : in bit ;
      S : in bit_vector (1 down to 0);
      Y : out bit);
end MUX ;
Architecture MUX of MUX is
   begin
      P1 : Process (D0, D1, D2, D3, S)
   begin
      if S = "00" then
        Y <= D0 ;
      elsif S = "01" then
            Y <= D1 ;
      elsif S = "10" then
            Y <= D2 ;
      else
            Y <= D3 ;
   End if ;
end process P1;
end MUX ;
```

10.12.5 Case Statements :

- Case statements allows to make a decision from a number of choices.
- The use of case statement is to specify a set of statements to execute which is based on the value of given selection signal.
- More than one value can be mentioned by separating by a symbol "|" e.g. "01" | "11".

Syntax :

```
Case expression is
   When choices ⇒ statement 1 ;
   When choices ⇒ statement 2 ;
   ⋮
   When others ⇒ statement n ;
end case ;
```

- A model for 4 : 1 MUX using case statement is as shown below :

```
Entity MUX is
   port (D0, D1, D2, D3 : in bit ;
         S : in bit ; Y : out bit);
end MUX ;
```

Scanned by CamScanner

```
Architecture MUX of MUX is
begin
    PMUX : process (D0, D1, D2, D3, S)
    begin
      case S is
      when "00" ⇒ Y : D0 ;
      when "01" ⇒ Y : D1 ;
      when "10" ⇒ Y : D2 ;
      when "11" ⇒ Y : D3 ;
      when "others" ⇒ Null ;
    end case ;
  end process PMUX;
  end MUX ;
```

10.12.6 Comparison between If and Case Statements :

Sr. No.	If statement	Case statement
1.	If statement produces priority encoded logic	Case statement produces parallel logic.
2.	Used for speed critical paths.	Used for complex decoding.
3.	Execution process is slow and requires easy design.	Execution process is fast but complex designs are required.
4.	In this case more hardware is required so more delay is generated by logic gates and hence speed is reduced.	In this case less hardware is required so delay by logic gates is less hence speed is high.

10.12.7 Loop Statements :

For repetitive operation of instructions in a program, loop structure is required. To iterate through a set of sequential statements, a loop statement is used.

Syntax :

```
[label :] iteration_scheme loop
      sequential statements ;
end loop [label] ;
```

Types of iteration schemes :

1. For scheme. 2. While scheme.
3. One where no iteration scheme is specified.

1. For scheme :

Syntax :

```
[label :] For variable in range loop
```



```
      sequential statements ;
end loop [label] ;
```

e.g.

```
A : = 1 ;
      For t in 1 to 15 loop
      A : = A + t ;
end loop ;
```

In above example loop executes for 15 times and each time value of t is incremented by 1.

2. While scheme :

Syntax :

```
[label :] while condition loop
      sequential statements ;
end loop [label] ;
```

e.g.

```
A : = 1, B : = 1
while A < 10 loop
      B : = B + 1
end loop
```

In this example, loop executes for 10 times and each time the value of A is incremented by 1.

3. Where no iteration scheme is specified :

All statements in loop body are executed repeatedly until some action causes it to exit the loop. These action can takes place by using statements exit or next or return.

Syntax :

```
[label :] loop
    sequential statements ;
end loop [label] ;
```

e.g.

```
A : = 1, B : = 1
loop
      B : = B + 1 ;
      A : = A + 1 ;
exit when A > 10 ;
end loop ;
```

- All types of loops may contain the next and exit statements.

10.12.8 Null Statements :

- When nothing is needed to be done, null statements are used.
- Execution continues with the statement which appears next to the null statement.

Syntax :

```
null
```

Scanned by CamScanner

- To indicate no action needs to be performed null statement is used. This statement can be used in if or case statement where some condition is to be satisfied.

e.g.

```
case ABC is
When "001" ⇒ Y1 : = A and B ;
When "010" ⇒ Y1 : A or B ;
When "100" ⇒ Y1 : = not A ;
When others ⇒ null ;
```

10.12.9 Exit Statements :

- The exit statement occurs only inside a loop. To exit the loop immediately exit statement is used.

Syntax :

[label :] exit [label 2] [when condition] ;

- Label 2 identifies the loop from which the execution jumps out. The execution jumps out of the innermost loop if no label is defined.

e.g.

```
Sum : = 1 ; j : = 0 ;
S : loop
    j : = j + 20 ;
    Sum : = Sum * 10 ;
    if (sum > 100) then
    exit S ;
end loop S ;
```

- When the condition is true (or if there is no condition) then exit statement is executed and control is passed to the first statement after the end loop.

10.12.10 Next Statement :

- In a loop to cause the next iteration, next statement is used.
- In the current iteration of specified loop, the next statement skips the remaining statements and execution continues with the first statement in the next iteration of loop.
- Innermost loop is assumed if no label is mentioned.
- The main difference between exit and next is, exit statement exits the loop entirely whereas the next statement skips to the "next" loop iteration i.e. it exits the current iteration of the loop.

Syntax :

next [Loop_Label] [when condition];

e.g.

```
for J in 10 downto 5 loop
    if (sum < total) then
    sum : = sum + 2 ;
    elsif (sum = total) then
        next ;
```



```
    else
        null ;
    end if ;
        k : = k + 1 ;
end loop ;
```

10.12.11 Assertion Statement :

- For error message generation and internal consistency check, assertion statement is used.

Syntax :

```
[label :] assert
    Boolean expression ;
[report string]
[severity name] ;
```

- Assertion statements can be used as sequential as well as concurrent statement. Report message is printed alongwith severity level if the value of Boolean expression is wrong.

10.13 Comparison between Concurrent and Sequential Statements :

Sr. No.	Concurrent statements	Sequential statements
1.	In concurrent statements, order of execution is not important.	In sequential statements order of execution is important as they appear within the process.
2.	A process block is a single concurrent statement.	Sequential statements appears only inside a process block.
3.	It can appear anywhere in the architecture body.	It must be in the process.
4.	Example of concurrent statement is process, concurrent signal assignment, block etc.	Example of sequential statement is for, case, if etc.

10.14 VHDL Attributes :

An attribute is function, type, value, signal, range or constant which can be associated within a VHDL description with certain name. Name of attribute can be an entity name, architecture name or any other name.

10.14.1 Types of Attributes :

1. User-defined 2. Predefined.

10.14.2 User Defined Attributes :

User defined attributes are used to declare constants of any type. To declare user defined attributes attribute declaration is used. An attribute declaration declares the name of attribute and its type.

Scanned by CamScanner

Syntax :

Attribute attribute-name : value-type ;

Attribute specification :

attribute attribute-name **of** item-name : name-class **is** value;

where,

item-name : is a list of one or more names of an entity, architecture, signal, variable, constant, type, procedure or function etc.

name-class : class type i.e. whether it is entity, architecture or others

value : '0', 27, "00 01 11 00" etc.

e.g.

attribute number - of - inputs : integer ... declaration

attribute number - of - inputs **of** nand3 : signal **is** 3; ... specification

- A user defined attribute can be declared anywhere in program excepting package body.
- It is ignored or warning is issued if user defined statements are not recognized by synthesis tool.

10.14.3 Pre-defined Attributes :

There are five classes of pre-defined attributes :

1. Value attributes : Return a constant value
2. Function attributes : Calls a function that returns a value
3. Signal attributes : Generates a new signal
4. Type attributes : Returns a type name
5. Range attributes : Returns a range

10.14.3.1 Value Attributes :

If T is any scalar type or subtype :

Sr. No.	Attribute	Result
1.	T'LEFT	Returns the left bound i.e. leftmost value of P.
2.	T'RIGHT	Returns the right bound i.e. rightmost value of P.
3.	T'HIGH	Returns the upper bound i.e. value of highest position of P.
4.	T'LOW	Returns the lower bound i.e. value of lowest position of P.

e.g.

type allowed-value **is range** 31 **downto** 0;

type week-day **is** (sun, mon, tue, web, thu, fri, sat) ;

subtype work-day **is** week-day range fri **downto** mon ;

If A is an array object then :

A' length (n) : returns the number of elements in the nth dimension (n = 1 if not specified)

10.14.3.2 Function Attributes :

- Function attributes are used to represent the functions that are called to obtain a value. To convert values from physical type to integer type, function attributes are used.

Table 10.14.1 : Attributes of physical types

Attribute	Result
T'Pos(s)	position number of s in T
T'Val(x)	value at position x in T (x is integer)
T'Succ(s)	value at position one greater than s in T
T'Pred(s)	value at position one less than s in T
T'Leftof(s)	value at position one to the left of s in T
T'Rightof(s)	value at position one to the right of s in T

- If A is an array object then refer Table 10.14.2.

Table 10.14.2 : Attributes of the array type

Attribute	Result
A'Left(n)	leftmost value in index range of dimension n
A'Right(n)	rightmost value in index range of dimension n
A'Low(n)	lower bound of index range of dimension n
A'High(n)	upper bound of index range of dimension n
A'Range(n)	index range of dimension n
A'Reverse_range(n)	reversed index range of dimension n
A'Length(n)	number of values in the n-th index range
A'Ascending(n)	true if index range of dimension n is ascending false otherwise

10.14.3.3 Signal Attributes :

Signal attributes create implicit signals which are created using signal declarations because these attributes create new signals from the associated signals.

Table 10.14.3 : Attributes of signals

Attribute	Result
S'Delayed(t)	Implicit signal, equivalent to signal S, but delayed units of time.
S'Stable(t)	Implicit signal that has the value true when no event has occurred on S for t time units, false otherwise.
S'Quiet(t)	Implicit signal that has the value true when no transaction has occurred on S for t time units, false otherwise.
S'Transaction	Implicit signal of type Bit whose value is changed in each simulation cycle in which a transaction occurs on S (signal S becomes active).
S'Event	True if an event has occurred on S in the current simulation cycle, false otherwise.
S'Active	True if a transaction has occurred on S in the current simulation cycle, false otherwise.
S'Last_event	The amount of time since last event occurred on S, if no event has yet occurred it returns Time' High.

Scanned by CamScanner

Attribute	Result
S'Last_active	The amount of time since last transaction occurred on S, if no event has yet occurred it returns Time' High.
S'Last_value	The previous value of S before last event occurred on it.
S'Driving	True if the process is driving S or every element of a composite S, or false if the current value of the driver for S or any element of S in the process is determined by the null transaction.
S'Driving value	The current value of the driver of S in the process containing the assignment statement to S.

10.14.3.4 Type Attributes :

T'Base is the only type attribute which indicates the base type for type T. It must be noted that this attribute is used only as a prefix for other attributes.

10.14.3.5 Range Attributes :

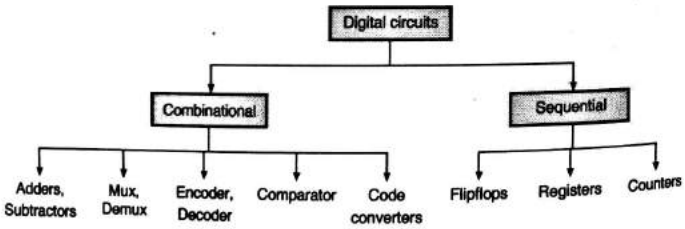
If A is an array object then,

- A'RANGE(N) : returns the Nth index range of A
- A'REVERSE.RANGE(N) : returns the Nth index range reversed.

This attribute is used in a for loop.

10.15 VHDL For Combinational and Sequential Circuits :

Fig. 10.15.1 shows classification of digital circuits.



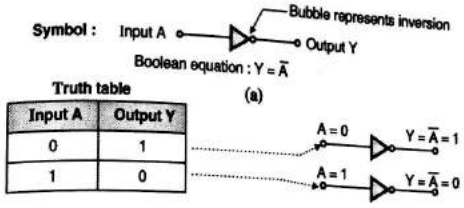
(C-5810)**Fig. 10.15.1 : Classification of Digital circuits**

In the following subsections we will discuss VHDL codes for digital circuits.

10.16 VHDL For Basic Logic Gates :

10.16.1 Implementation of NOT Gate :

The NOT gate or Inverter is a logic gate having one input (A) and one output (Y). Its symbol and truth table are shown in Fig. 10.16.1.



Input A	Output Y
0	1
1	0

(b) (c) Equivalent circuits

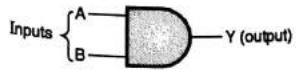
(B-450) **Fig. 10.16.1 : Symbol, truth table and equivalent circuit of a NOT gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity ANDgate is
  Port ( A : in std_logic;
         Y : out std_logic);
end ANDgate;
architecture Behavioral of ANDgate is
begin
process (A)
begin
    Y<=not A;
end Behavioral;
```

10.16.2 Implementation of AND Gate :

The Boolean expression for a two input AND gate is as follows :

$$Y = A \cdot B$$



Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(a) Logical symbol (b) Truth table

(B-453) **Fig. 10.16.2**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Scanned by CamScanner

```
use IEEE.STD_LOGIC_ARITH.ALL;
entity ANDgate is
   Port ( A,B : in std_logic;
          Z : out std_logic);
end ANDgate;
architecture Behavioral of ANDgate is
begin
process (A,B)
begin
      Z<=A and B;
end Behavioral;
```

10.16.3 Implementation of OR Gate :

The Boolean expression for a two input OR gate is as follows :

$$Y = A + B$$

(a) Logical symbol

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

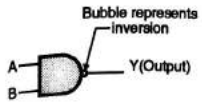
(B-460) **Fig. 10.16.3 : Two input OR gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity ORgate is
   Port ( A,B : in std_logic;
          Z : out std_logic);
end ORgate;
architecture Behavioral of ORgate is
begin
process(A,B)
begin
      Z<=A OR B;
end Behavioral;
```

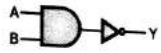
10.16.4 Implementation of NAND Gate :

- The Boolean expression for the NAND gate is,

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



(a) Logical symbol



(b) Equivalent circuit

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

(c) Truth table

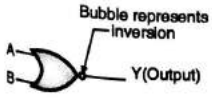
(B-467) **Fig. 10.16.4 : Two input NAND gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity NANDgate is
   Port ( A,B : in std_logic;
          Y : out std_logic);
end NANDgate;
architecture Behavioral of NANDgate is
begin
process(A,B)
begin
      Y<=A nand B;
end Behavioral;
```

10.16.5 Implementation of NOR Gate :

- The Boolean expression for the NOR gate is given by,

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



(a) Logical symbol

(b) Equivalent circuit

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

(c) Truth table

(B-471) **Fig. 10.16.5 : A two input NOR gate**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

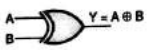
Scanned by CamScanner

```
entity NORgate is
   Port ( A,B : in std_logic;
          Y : out std_logic);
end NORgate;
architecture Behavioral of NORgate is
begin
process
begin
      Y<=A nor B;
end Behavioral;
```

10.16.6 Implementation of XOR Gate :

- The Boolean expression for the two input EX-OR gate is,

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



(a) Symbol of a two input EX-OR gate

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth table of a two input EX-OR gate

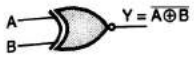
(B-4897) Fig. 10.16.6

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity  xorgate is
   Port ( A,B : in std_logic;  Z : out std_logic);
end  xorgate;
 architecture Behavioral of xorgate is
begin
process(A,B)
begin
Z<=A xor B;
end Behavioral;
```


10.16.7 Implementation of XNOR Gate :

- The Boolean expression for an EX-NOR gate is given by,

$$Y = \overline{A \oplus B} \text{ or } A \odot B = \overline{A}\,\overline{B} + AB$$



(a) Symbol of a two input EX-NOR gate

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

(b) Truth table of a two input EX-NOR gate

(B-478) Fig. 10.16.7

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity xnorgate is
   Port ( A,B : in std_logic;
          Y : out std_logic);
end xnorgate;
 architecture Behavioral of NANDgate is
begin
process(A,B)
begin
y<=A xnor B;
end Behavioral;
```

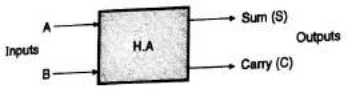
10.17 VHDL For Combinational Circuits :

10.17.1 Implementation of Adders :

10.17.1.1 Implementation of Half Adder :

- Fig. 10.17.1 shows block diagram and truth table of half adder and half adder circuit is as shown in Fig. 10.17.2.

Scanned by CamScanner

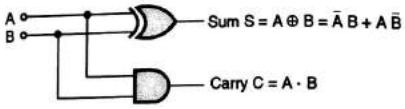


(a) Block diagram

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Truth table

(C-344) **Fig. 10.17.1 : Half adder**



(C-346) **Fig. 10.17.2 : Half adder circuit**

Behavioral model for half adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity halfadder1 is
   Port ( a,b : in std_logic; sum,carry : out std_logic);
end halfadder1;
architecture Behavioral of halfadder1 is
begin
process(a,b)
 begin
 sum<=a xor b;
  carry<= a and b;
end process;
end Behavioral;
```

Dataflow model for half adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity halfadder is
   Port ( a,b : in std_logic;
        sum,carry : out std_logic);
end halfadder;
architecture dataflow of halfadder is
```



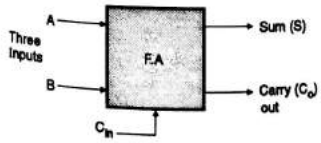
```
begin
      sum <= (a xor b);
      carry <= (a and b);

end dataflow;
```

Structural model for half adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity halfadder2 is
   Port ( a,b : in std_logic;
        sum,carry : out std_logic);
end halfadder2;
architecture structural of halfadder2 is
component xor1
port(c,d : in std_logic;y: out std_logic);
end component;
component and1
port(e,f : in std_logic;z:out std_logic);
end component;
begin
v1: xor1 port map (a,b,sum);
v2:and1 port map (a,b,carry);
end structural;
```

10.17.1.2 Implementation of Full Adder :

- Figs 10.17.3(a) and (b) shows block diagram and truth table of full adder and full adder circuit is as shown in Fig. 10.17.3(c).



(a) Block diagram

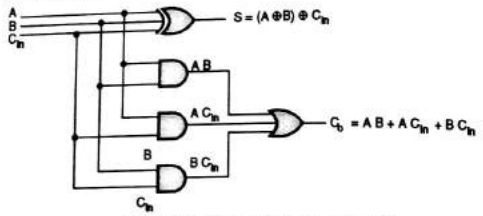
Inputs			Outputs	
A	B	C_{in}	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(b) Truth table

(C-350) **Fig. 10.17.3 : Full adder**

Scanned by CamScanner

Logic diagram for full adder :



(C-353) **Fig. 10.17.3(c) : Full adder circuit**

Behavioral model for full adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity behavior is
   Port ( a,b,c : in std_logic;
        sum,carry : out std_logic);
end behavior;
architecture Behavioral of behavior is
begin
process( a,b,c)
begin
        sum<=a xor b xor c;
        carry<=(a and b) or (a and C) or (B and C);
end process;
end Behavioral;
```

Dataflow model for full adder :

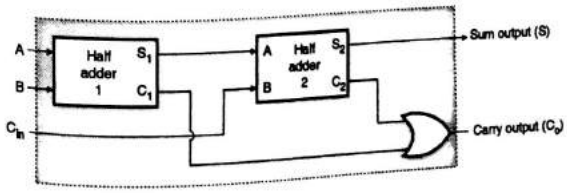
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fulladder1 is
   Port ( a,b,c : in std_logic;
        sum,carry : out std_logic);
end fulladder1;
architecture dataflow of fulladder1 is
begin
        sum<=a xor b xor c;
        carry<=(a and b) or (a and C) or (B and C);
end dataflow;
```


Structural model for full adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity structural is
   Port ( a,b,c : in std_logic;
        sum,carry : out std_logic);
end structural;
architecture structural of structural is
component xor1
port(p,q,r : in std_logic;y: out std_logic);
end component;
component andgate
port(x,y : in std_logic;z: out std_logic);
end component;
component or1
port(d,e,f : in std_logic; g: out std_logic);
end component;
signal c1,c2,c3,c4: std_logic;
begin
y1: andgate port map(a,b,c1);
y2: andgate port map(a,c,c2);
y3: andgate port map(b,c,c3);
y4: xor1 port map(a,b,c,c4);
y5: or1 port map (c1,c2,c3,carry);
end structural;
```

10.17.1.3 Implementation of Full Adder Using Half Adders :

- The full adder circuit can be constructed using two half adders as shown in Fig. 10.17.4.



(C-354) **Fig. 10.17.4 : Full adder using half adders**

Scanned by CamScanner

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity fa is
   Port ( x,y,z : in std_logic;
          sum,carry : out std_logic);
end fa;
architecture structural of fa is
component HA is
port(A,B: in std_logic;S,C:out std_logic);
end component;

signal P,Q,R: std_logic;
begin
  u1: HA port map (x,y,P,R);
  u2: HA port map (P,z,sum,Q);
 carry<=Q or R;

end structural;
```

10.17.1.4 Implementation of 4-Bit Full Adder :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity fbitadder is
   Port ( c : in std_logic;
          A3,A2,A1,A0 : in std_logic;
          B3,B2,B1,B0 : in std_logic;
          S3,S2,S1,S0 : out std_logic;
          Cout : out std_logic);
end fbitadder;

architecture structural of fbitadder is

component fulladd
port(c,A,B: in std_logic;S,COUT:out std_logic);
```



```
end component;
signal c1,c2,c3:std_logic;
begin
  stage0:fulladd port map (c,A0,B0,S0,c1);
  stage1:fulladd port map (c1,A1,B1,S1,c2);
  stage2:fulladd port map (c2,A2,B2,S2,c3);
  stage3:fulladd port map (c3,A3,B3,S3,COUT);


end structural;
```

10.17.2 Implementation of Subtractor :

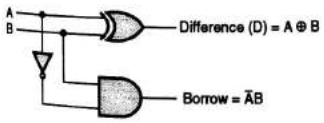
10.17.2.1 Implementation of Half Subtractor :

(C-3514)

Table 10.17.1 : Truth table for half subtractor

Inputs		Outputs	
A	B	Difference D (A – B)	Borrow B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Logic diagram :

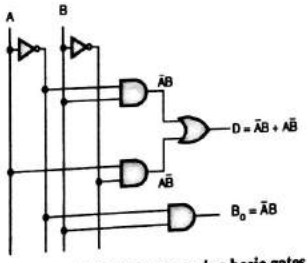


(C-357) **Fig. 10.17.5 : Half subtractor circuit**

Half subtractor using basic gates :

$$D = \bar{A}B + A\bar{B} \quad \text{and} \quad B_o = \bar{A}B$$

The half subtractor using basic gates is as shown in Fig. 10.17.6.



(C-358) **Fig. 10.17.6 : Half subtractor using basic gates**

Behavioral model for half subtractor :

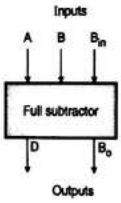
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

Scanned by CamScanner

```
entity halfsub is
   Port ( a,b : in std_logic;
        difference,borrow : out std_logic);
end halfsub;

architecture Behavioral of halfsub is
signal abar: std_logic;
begin
process(a,b)
begin
 difference<=a xor b;
 borrow<= ( not a) and b;
end process;
end Behavioral;
```

10.17.2.2 Implementation of Full Subtractor :

The truth table for full subtractor is shown in Table 10.17.2.

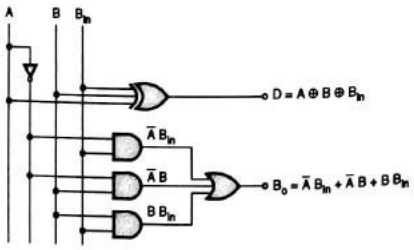


(C-3515)

Table 10.17.2 : Truth table for a full subtractor

Inputs			Outputs	
A (Minuend)	B (Subtrahend)	B_{in} Previous borrow	$(A - B - B_{in})$	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Logic diagram :



(C-361) **Fig. 10.17.7 : Logic diagram for a full subtractor**

Behavioral model for full subtractor :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity fullsub is
 Port ( a,b,bi : in std_logic;    difference, borrow : out std_logic);
end fullsub;

architecture Behavioral of fullsub is

begin
process (a,b,bi)

        difference<=a xor b xor bi;
        borrow<=(not a) and bi or (not a) and b or b and bi;
end process;
end Behavioral;
```

Structural model for full subtractor :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity fullsubtractor is
   Port ( a,b,bi : in std_logic;
        difference, borrow : out std_logic);
end fullsubtractor;
architecture structural of fullsubtractor is
component xor1
     port(c,d,e : in std_logic;y: out std_logic);
end component;
component and1
     port(w,x : in std_logic;z:out std_logic);
end component;
component or1
     port(p,q,r : in std_logic;s: out std_logic);
end component;
```

Scanned by CamScanner

```
 signal c1,c2,c3,c4:std_logic;
 begin
 v1:and1 port map (not a,bi,c1);
 v2: and1 port map (not a,b,c2);
 v3: and1 port map (b,bi,c3);
 v4:xor1 port map (a,b,bi,difference);
 v5:or1 port map(c1,c2,c3,borrow);
 end structural;
```

Dataflow model for full subtractor :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity fs is
   Port ( a,b,bi : in std_logic;
          difference,borrow : out std_logic);
end fs;

architecture dataflow of fs is

begin
          difference<=a xor b xor bi;
          borrow<=(not a and bi) or (not a and b) or (b and bi);

end dataflow;
```

10.17.2.3 Implementation of 4 bit Adder-Subtractor :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity addsub is
   Port ( op : in std_logic;
          A,B : in std_logic_vector(3 downto 0);
          R : out std_logic_vector(3 downto 0);
          cout,overflow : out std_logic);
end addsub;
architecture structural of addsub is
   component fulladder is
   port(x,y,cin: in std_logic;sum,cout:out std_logic) ;
   end component;
 signal c1,c2,c3,c4:std_logic;
```



```
signal tmp:std_logic_vector(3 downto 0);
begin
tmp<=A xor B;
FA0:fulladder port map(A(0),tmp(0),op,R(0),c1);
FA1:fulladder port map(A(1),tmp(1),op,R(1),c2);
FA2:fulladder port map(A(2),tmp(2),op,R(2),c3);
FA3:fulladder port map(A(3),tmp(3),op,R(3),c4);
overflow <=c3 xor c4;
cout<=c4;
end structural;
```

10.17.3 Implementation of Comparator :

VHDL code for 2 bit comparator :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity comparator is
   Port ( A,B : in std_logic_vector(1 downto 0); AltB,AgtB,AetB : out std_logic);
end comparaor;
architecture dataflow of comparator is
begin
   AetB<='1' when A=B else '0';
   AgtB<='1' when A>B else '0';
   AltB<='1' when A<B else '0';
end dataflow;
```

VHDL code for 4 bit comparator :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity bitcomparator is
   Port ( A,B : in std_logic_vector(0 to 3); AgtB,AetB,AltB : out std_logic);
end bitcomparator;
architecture Behavioral of bitcomparator is

begin
process(A,B)
begin
if(A>B) then
AgtB<='1';
AetB<='0';
```

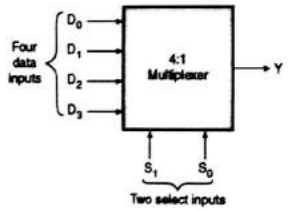
Scanned by CamScanner

```
AltB<='0';
elsif (A<B) then
AltB<='1';
AetB<='0';
AgtB<='0';
else
AetB<='1';
AgtB<='0';
AltB<='1';
end if;
end process;
end Behavioral;
```

10.17.4 Implementation of Multiplexer :

10.17.4.1 Implementation of 4 : 1 Multiplexer :

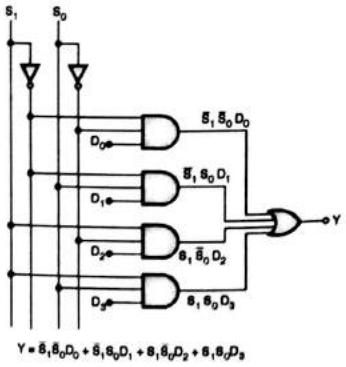
- Fig. 10.17.8 shows the block diagram of a 4 : 1 multiplexer and Table 10.17.3 gives its truth table.



(C-416) **Fig. 10.17.8 : 4 : 1 multiplexer**

Table 10.17.3 : Truth table

Select inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



(C-417) **Fig. 10.17.9 : Realization of 4 : 1 multiplexer**

Dataflow model for 4 : 1 MUX :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity data is
  Port ( d0,d1,d2,d3,s0,s1 : in std_logic;
         y : out std_logic);

end data;
architecture dataflow of data is
begin
     y<= d0 when s1='0'     and s0='0'else
         d1 when s1='0'     and s0='1'else
         d2 when s1='1'     and s0='0'else
         d3 when s1='1'     and s0='1' ;
end dataflow;
```

Behavioural model for 4 : 1 MUX (using IF statement) :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity muxif is
  Port ( d0,d1,d2,d3,s0,s1  : in std_logic;
       y : out std_logic);
end muxif;

architecture Behavioral of muxif is

begin

       process (d0,d1,d2,d3,s0,s1)
      begin
           if  (s1='0' and s0 = '0') then  Y<= d0;
        elsif  (s1='0' and s0 = '1') then  Y<= d1;
        elsif  (s1='1' and s0 = '0') then  Y<= d2;
        else            Y<= d3;
        end if;
      end process;

end Behavioral;
```

Scanned by CamScanner

Behavioural model for 4 : 1 MUX (using case statement) :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity muxcase is
   Port ( d0,d1,d2,d3 : in std_logic;S:in std_logic_vector(1 downto 0) ; y : out std_logic);
end muxcase;
architecture Behavioral of muxcase is
begin
process(d0,d1,d2,d3,s )
begin
            case S is
                  when "00" => y<=d0;
                  when "01" => Y<=d1;
                  when "10" => Y<=d2;
                  when "11" => Y<=d3;
                  when others => Y<=d0;
                  end case;
end process;
end Behavioral;
```

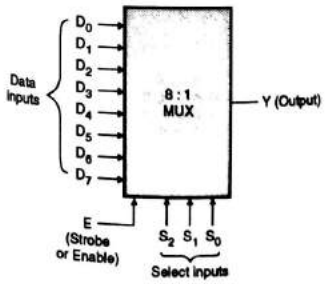
Structural model for 4 : 1 MUX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity muxstructure is
   Port ( s0,s1 : in std_logic_vector(0 downto 0);   d : in std_logic_vector(3 downto 0);
 y : out std_logic);
end muxstructure;
architecture structural of muxstructure is
component INV
     port(pin: in std_logic;
```



```
     pout : out std_logic);
end component;
component        gate
   port(A0, A1, A3 : in std_logic; Aout : out std_logic);
end component;
component OR_4
   port(B0,B1,B2,B3 : in std_logic;
      Bout : out std_logic);
end component;
signal s0bar, s1bar, T1,T2,T3,T4: std_logic;
begin
INV0 :  INV port map (s0,s0bar);
INV1 :  INV port map (s1,s1bar);
A4 :  gate port map      (d(0),s0bar,s1bar,T1);
A5 :  gate port map      (d(1),s0,s1bar,T2);
A6 :  gate port map      (d(2), s0bar,s1,T3);
A7 :  gate port map  (d(3), s0, s1,T4);
O1 :  OR_4 port map (T1,T2,T3,T4,y);
end structural;
```

10.17.4.2 Implementation of 8 : 1 MUX :



(a) Block diagram

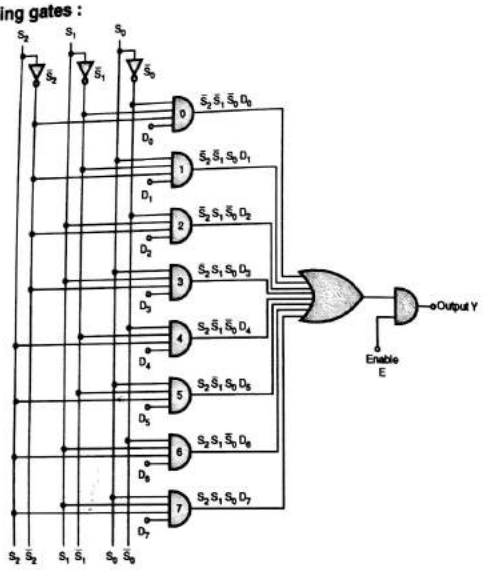
Enable	Select inputs			Output
E	S_2	S_1	S_0	Y
0	X	X	X	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

(b) Truth table

(C-419) **Fig. 10.17.10 : 8 : 1 multiplexer**

Scanned by CamScanner

Realization using gates :



(C-420) **Fig. 10.17.11 : 8 : 1 MUX using gates**

Dataflow model for 8 : 1 MUX :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity dataflow is
   Port ( d0,d1,d2,d3,d4,d5,d6,d7,s0,s1,s2 : in std_logic;
        y : out std_logic);
end dataflow;
architecture dataflow of dataflow is
begin
     y<=  d0 when s2='0'and s1='0'and s0='0'else
          d1 when s2='0'and s1='0'and s0='1'else
          d2 when s2='0'and s1='1'and s0='0'else
          d3 when s2='0'and s1='1'and s0='1'else
          d4 when s2='1'and s1='0'and s0='0'else
          d5 when s2='1'and s1='0'and s0='1'else
          d6 when s2='1'and s1='1'and s0='0'else
          d7 when s2='1'and s1='1'and s0='1';
end dataflow;
```


Behavioural model for 4 : 1 MUX :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity behavior is
   Port ( d0,d1,d2,d3,d4,d5,d6,d7 : in std_logic;  s : in std_logic_vector(2 downto 0);
          y : out std_logic);
end behavior;
architecture Behavioral of behavior is
begin
process(d0,d1,d2,d3,d4,d5,d6,d7,s )
begin
                case S is
                      when "000" => y<=d0;
                      when "001" => y<=d1;
                      when "010" => Y<=d2;
                      when "011" => Y<=d3;
          when "100" => y<=d4;
          when "101" => y<=d5;
          when "110" => y<=d6;
           when "111" => y<=d7;
            when others => Y<=d0;
 end case;
end process;
end Behavioral;
```

Structural model for 8 : 1 MUX :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity new1 is
   Port ( d : in std_logic_vector(7 downto 0);
          s : in std_logic_vector(2 downto 0);
          y : out std_logic);
end new1;
architecture structural of new1 is
component INV
```

Scanned by CamScanner

```
    port(pin: in std_logic; pout : out std_logic);
end component;
component gate
    port(p,q,r,s : in std_logic; Aout : out std_logic);
end component;
component OR8
    port(B0,B1,B2,B3,B4,B5,B6,B7 : in std_logic; Bout : out std_logic);
end component;
signal s0bar,s1bar,s2bar,T1,T2,T3,T4,T5,T6,T7,T11 : std_logic;
begin
INV0 :  INV port map (s0,s0bar);
INV1 :  INV port map (s1,s1bar);
INV2  :  INV port map (s2,s2bar);
A4  :   gate port map (d,s0bar,s1bar,s2bar,T1);
A5  :   gate port map (d,s0,s1bar,s2bar,T2);
A6 :    gate port map  (d(2),s0bar,s1,s2bar,T3);
A7 :    gate port map  (d(3),s0, s1,s2bar,T4);
A9 :   gate port map (d(4),s0bar,s1bar,s2,T5);
A10:    gate port map (d(5),s0,s1bar,s2,T6);
A11:    gate port map (d(6),s0bar,s1,s2,T7);
A12:    gate port map(d(7),s0,s1,s2,T11);
O1 :     OR8 port map (T1,T2,T3,T4,T5,T6,T7,T11,y);
end structural;
```

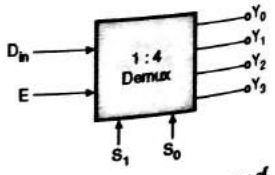
10.17.5 Implementation of De-Multiplexers :

10.17.5.1 Implementation of 1 : 4 De-Multiplexer :

- The 1 : 4 demultiplexer is shown in Fig. 10.17.12 and its truth table is given in Table 10.17.4.

Table 10.17.4 : Truth table for 1 : 4 demultiplexer

		Inputs		Outputs				
E	D_{in}	S_1	S_0	Y_0	Y_1	Y_2	Y_3	
1	0	0	0	0	0	0	0	Y_0 is connected to D_{in}
1	1	0	0	1	0	0	0	
1	0	0	1	0	0	0	0	Y_1 is connected to D_{in}
1	1	0	1	0	1	0	0	
1	0	1	0	0	0	0	0	Y_2 is connected to D_{in}
1	1	1	0	0	0	1	0	
1	0	1	1	0	0	0	0	Y_3 is connected to D_{in}
1	1	1	1	0	0	0	1	



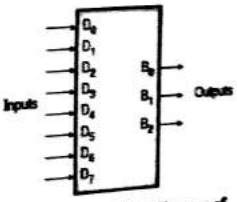
(C-451) **Fig. 10.17.12 : Block diagram of 1 : 4 demultiplexer**


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity demux is
  Port ( I,s0,s1 : in std_logic;
      y0,y1,y2,y3 : out std_logic);
end demux;
architecture Behavioral of demux is
begin
process(I,s0,s1)
begin
if(s0='0'and s1='0')  then y0<=I;
elsif(s0='0'and s1='1')  then y1<=I;
elsif(s0='1'and s1='0')  then y2<=I;
else
y3<=I;
end if;
end process;
end Behavioral;
```

10.17.6 Implementation of Encoders :

10.17.6.1 Implementation of 8 : 3 (Octal to Binary) Encoder :

Fig. 10.17.13 shows the block diagram of octal to binary encoder and Table 10.17.5 gives its truth table.



(C-474) **Fig. 10.17.13 : Block diagram of octal to binary encoder**

Table 10.17.5 : Truth table of octal to binary encoder

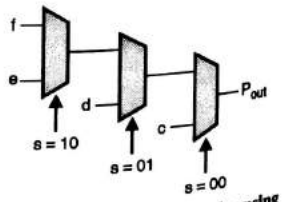
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	B_0	B_1	B_2
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Scanned by CamScanner

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity encoder is
   Port ( D : in std_logic_vector(7 downto 0);B : out std_logic_vector(2 downto 0));
end encoder;

architecture Behavioral of encoder is

begin

process (D)
      begin
         case D is
            when "00000001" => B <= "000";
            when "00000010" => B <= "001";
            when "00000100" => B <= "010";
            when "00001000" => B <= "011";
            when "00010000" => B <= "100";
            when "00100000" => B <= "101";
            when "01000000" => B <= "110";
            when "10000000" => B <= "111";
            when others => B <= "XXX";
end case;
end process;
end Behavioral;
```

10.17.6.2 Implementation of 4 : 2 Priority Encoders using If-Then-Else Statement :

If-then-else statements should be used to imply priority on a late arriving signal. In the following examples, shown in Fig. 10.17.14, signal *c* is a late arriving signal.



(C-1464) **Fig. 10.17.14 : Priority encoder using if_then_else statement**

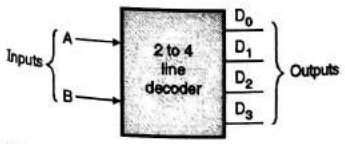

```
library IEEE;
use IEEE.std_logic_1164.all;

entity priorityencoder is
   port (c, d, e, f : in std_logic;
      s : in std_logic_vector(1 downto 0);
      pout : out std_logic);
end priorityencoder;

architecture arch of priorityencoder is
begin
   myif_pro: process (s, c, d, e, f)
begin
      if s = "00" then
            pout <= c;
      elsif s = "01" then
            pout <= d;
      elsif s = "10" then
            pout <= e;
      else pout <= f;
      end if;
   end process myif_pro;
end arch;
```

10.17.7 Implementation of Decoders :

10.17.7.1 VHDL Code for 2 : 4 Decoder :

Table 10.17.6 : Truth table of a 2 line to 4 line decoder

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



(C-400) **Fig. 10.17.15 : Block diagram of a 2 line to 4 line decoder**

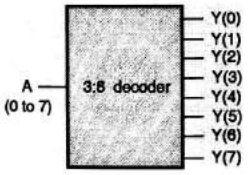
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

Scanned by CamScanner

```
entity decoder is
   Port ( A,B : in std_logic_vector(1 downto 0);
          D : out std_logic_vector(3 downto 0));
end decoder;
architecture Behavioral of decoder is
begin
process(A)
begin
case A is
when "00"  =>D<="0001" ;
when "01"  =>D<="0010" ;
when "10" =>D<="0100" ;
when others=>D<="1000"  ;
end case;
end process;
end Behavioral;
```

10.17.7.2 Implementation of 3 : 8 Binary Decoder :

- Fig. 10.17.16 shows 3 : 8 binary decoder and Table 10.17.6 shows truth table for the same.



(C-1465) **Fig. 10.17.16 : 3 : 8 binary decoder**

Table 10.17.7 : Truth table of 3:8 decoder

Inputs			Outputs							
A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
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


```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;


entity Decoder3_8 is
    Port ( A : in integer range 0 to 7;
           Y : out std_logic_vector(7 downto 0));
end Decoder3_8;

architecture behavioral of Decoder3_8 is
begin
  process (A)
    begin
        case A is
            when 0 => Y <= "00000001";
            when 1 => Y <= "00000010";
            when 2 => Y <= "00000100";
            when 3 => Y <= "00001000";
            when 4 => Y <= "00010000";
            when 5 => Y <= "00100000";
            when 6 => Y <= "01000000";
            when 7 => Y <= "10000000";
        end case;
    end process;
end behavioral;
```

10.17.7.3 VHDL Code for BCD to 7 Segment Decoder :

Table 10.17.8 : Truth table for BCD to seven segment decoder with a common cathode display

Decimal	Inputs				Outputs						
	B_3	B_2	B_1	B_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0

Scanned by CamScanner

Decimal	Inputs				Outputs						
	B_3	B_2	B_1	B_0	a	b	c	d	e	f	g
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity bcdtoseven is
   Port ( B : in std_logic_vector(3 downto 0);
        Y : out std_logic_vector(0 to 6));
end bcdtoseven;
architecture Behavioral of bcdtoseven is
begin
 process(B)
 begin
 case B is
 when "0000"=>Y<="1111110";
 when "0001"=>Y<="0110000";
 when "0010"=>Y<="1101101";
 when "0011"=>Y<="1111001";
 when "0100"=>Y<="0110011";
 when "0101"=>Y<="1011011";
 when "0110"=>Y<="1011111";
 when "0111"=>Y<="1110000";
 when "1000"=>Y<="1111111";
 when "1001"=>Y<="1111011";
 when others=>Y<="1111111";
 end case;
 end process;
 end Behavioral;
```

10.17.8 Implementation of Parity Circuits :

- The Parity generator is a combinational circuit which adds additional bit (parity) to the pattern so that total number of bits is even or odd.
- If the total number of 1's becomes odd after addition of a bit then it is called as **odd parity** and **even parity**, if number of 1's becomes even.

10.17.8.1 VHDL Code for Even Parity Generator :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity EPG is
   Port ( x,y,z : in std_logic; p : out std_logic);
end EPG;
architecture Behavioral of EPG is
begin
p<=((x xor y) xnor z);
end Behavioral;
```

10.17.8.2 VHDL Code for 8 bit Parity Generator using for Loop and Generic Statement :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity PG is
generic (n:integer:=7);
   Port ( a : in std_logic_vector(n-1 downto 0); b : out std_logic_vector(n downto 0));
end PG;
architecture Behavioral of PG is
begin
 process(a)
        variable temp1:std_logic;
        variable temp2:std_logic_vector(b'range);
        begin
                temp1:='0';
                for i in a'range loop
                        temp1:=temp1 xor a(i);
                        temp2(i):=a(i);
                end loop;
                        temp2(b'high):=temp1;
                        b<=temp2;
end process;
end Behavioral;
```

10.17.8.3 VHDL Code for 4-bit Parity Checker :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Scanned by CamScanner

```
use IEEE.STD_LOGIC_ARITH.ALL;
entity PC is
   Port ( a0,a1,a2,a3 : in std_logic; p : out std_logic);
end PC;
architecture Behavioral of PC is
signal r,s : std_logic;
begin
 r <= a0 xor a1;
 s <= a2 xor r;
 p <= s xor a3;
end Behavioral;
```

10.17.9 Implementation of Code Converters :

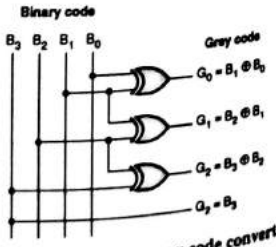
10.17.9.1 VHDL Code for Binary to Gray Code Converter :

Table 10.17.9 : Truth table for binary to gray code converter

Decimal	Binary inputs				Gray outputs			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Realization :

Binary to gray code converter is shown in Fig. 10.17.17.



(C-376) **Fig. 10.17.17 : Binary to gray code converter**

Behavioral model for binary to gray code converter :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity b2g is
   Port ( B : in std_logic_vector(3 downto 0); G : out std_logic_vector(3 downto 0));
end b2g;

architecture Behavioral of b2g is

begin

   process (B)
      begin
         case B is
            when "0000" => G <= "0000";
            when "0001" => G <= "0001";
            when "0010" => G <= "0011";
            when "0011" => G <= "0010";
            when "0100" => G <= "0110";
            when "0101" => G <= "0111";
            when "0110" => G <= "0101";
            when "0111" => G <= "0100";
            when "1000" => G <= "1100";
            when "1001" => G <= "1101";
            when "1010" => G <= "1111";
            when "1011" => G <= "1110";
            when "1100" => G <= "1010";
            when "1101" => G <= "1011";
            when "1110" => G <= "1001";
            when others => G <= "1000";
 end case;
 end process;
end Behavioral;
```

Dataflow model for binary to gray code converter:

```
library IEEE;
```

Scanned by CamScanner

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity b2gray is
   Port ( B : in std_logic_vector(3 downto 0);
          G : out std_logic_vector(3 downto 0));
end b2gray;
architecture dataflow of b2gray is

begin
  G(0)<=B(1) xor B(0);
  G(1)<=B(2) xor B(1);
  G(2)<=B(3) xor B(2);
  G(3)<=B(3);
end dataflow;
```

10.17.9.2 VHDL Code for BCD to Excess-3 Converter :

Truth table for BCD to Excess 3 Converter:

- We know that Excess-3 code can be derived from the BCD code by adding 3 to each BCD number.
- For example decimal 13 is represented as 0001 0011 in BCD.

  If we add 3 i.e. (0011 0011) then the corresponding Excess 3 code is 0100 0110.

Decimal	BCD inputs				Excess-3 outputs			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Behavioral model for BCD to excess-3 Converter :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity b2ex3 is
```



```
  Port ( B : in std_logic_vector(3 downto 0);Ex3 : out std_logic_vector(3 downto 0));
end b2ex3;
architecture Behavioral of b2ex3 is
begin
  process (B)
begin
          case B is
         when "0000" => EX3 <= "0011";
         when "0001" => EX3 <= "0100";
         when "0010" => EX3 <= "0101";
         when "0011" => EX3 <= "0110";
         when "0100" => EX3 <= "0011";
         when "0101" => EX3 <= "1000";
         when "0110" => EX3 <= "1001";
         when "0111" => EX3 <= "1010";
         when "1000" => EX3 <= "1011";
         when "1001" => EX3 <= "1100";
         when others => null;
       end case;
   end process;
end Behavioral;
```

10.17.9.3 VHDL Code for Gray to Binary Code Converter :

Table 10.17.10 : Truth table for gray to binary code converter

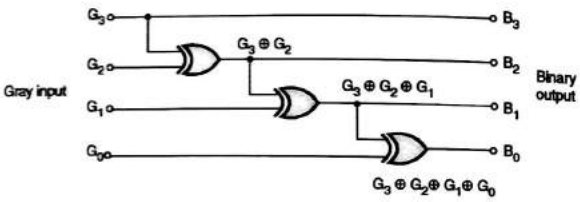
Decimal	Gray code input				Binary output			
	G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
3	0	0	1	1	0	0	1	0
2	0	0	1	0	0	0	1	1
6	0	1	1	0	0	1	0	0
7	0	1	1	1	0	1	0	1
5	0	1	0	1	0	1	1	0
4	0	1	0	0	0	1	1	1
12	1	1	0	0	1	0	0	0
13	1	1	0	1	1	0	0	1
15	1	1	1	1	1	0	1	0
14	1	1	1	0	1	0	1	1
10	1	0	1	0	1	1	0	0
11	1	0	1	1	1	1	0	1

Scanned by CamScanner

Decimal	Gray code input				Binary output			
	G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
9	1	0	0	1	1	1	1	0
8	1	0	0	0	1	1	1	1

Realization :

The gray to binary code converter is as shown in Fig. 10.17.18.



(C-380) **Fig. 10.17.18 : Gray to Binary code converter**

Behavioral Model for gray to binary code converter :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity GBC is
   Port ( G : in std_logic_vector(3 downto 0); B : out std_logic_vector(3 downto 0));
end GBC;
architecture Behavioral of GBC is
begin
  process(G)
  begin
  case G is
            when "0000" => B <= "0000";
            when "0001" => B <= "0001";
            when "0010" => B <= "0011";
            when "0011" => B <= "0010";
            when "0100" => B <= "0111";
            when "0101" => B      <= "0110";
            when "0110" => B      <= "0100";
            when "0111" => B <= "0101";
            when "1000" => B <= "1111";
            when "1001" => B <= "1110";
            when "1010" => B <= "1100";
            when "1011" => B <= "1101";
            when "1100" =>B <= "1000";
            when "1101" => B <= "1001";
```



```
            when "1110" => B <= "1011";
            when others => B <= "1010";
        end case;
  end process;
end Behavioral;
```

Dataflow model for gray to binary code converter :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity gray2b is
   Port ( G : in std_logic_vector(3 downto 0); B : out std_logic_vector(3 downto 0));
end gray2b;
architecture dataflow of gray2b is
begin
   B(0)<=(G(3)xor G(2))xor (G(1)xor G(0));
   B(1)<= G(3) xor G(2) xor G(1);
   B(2)<=G(3) xor G(2);
   B(1)<=G(3);
end dataflow;
```

10.18 VHDL for Sequential Circuits :

10.18.1 Implementation of Flip-Flops :

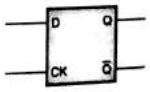
10.18.1.1 VHDL Code for D Flip-Flop :

SPPU : Dec. 15

University Questions

Q.1 Write entity and architecture of D flip-flop with clear input using behavioural modeling. **(Dec. 15, 6 Marks)**

Logic diagram :



(C-3419) **Fig. 10.18.1 : Logic symbol**

Truth table of D flip flop :

Table 10.18.1 : Truth table of a D flip flop

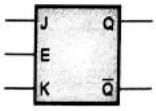
Q output		Input
Present state	Next state	D_n
0	0	0
0	1	1
1	0	0
1	1	1

Scanned by CamScanner

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity dff is
   Port ( D,clk : in std_logic;  Q: out std_logic);
end dff;
architecture Behavioral of dff is
begin
process(clk)
begin
if(clk'event and clk='1')then
Q<=D;

end if;
end process;


end Behavioral;
```

10.18.1.2 Implementation of JK Flip Flop :

Logic diagram :



(C-3420) **Fig. 10.18.2 : Logic Symbol**

Truth table :

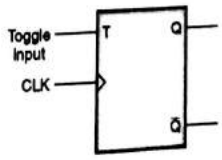
Table 10.18.2 : Truth table

J	K	Q_n	Q_{n+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


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity JKFF is
  Port ( j,k,clock : in std_logic;
       q,qbar : out std_logic);
end JKFF;
architecture Behavioral of JKFF is
begin
  process (clock)
    variable tmp:std_logic;
      begin
        if (clock = '1'and clock'event) then
         if(j='0' and k='0')     then
         tmp:=tmp;
         elsif(j='1'and k='1')then
         tmp:=not tmp;
         elsif(j='0'and k='1')then
         tmp:='0';
         else
         tmp:='1';
         end if;
         end if;
         q<=tmp;
         qbar<=not tmp;


  end process;

end Behavioral;
```

10.18.1.3 Implementation of T Flip Flop :

Logic diagram :



(C-5847) **Fig. 10.18.3 : Logic symbol**

Scanned by CamScanner

Truth table :

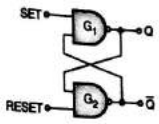
Table 10.18.3 : Truth table of T flip-flop

Q output		Input
Present state	Next state	T_n
0	0	0
0	1	1
1	0	1
1	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity tff is
   Port ( T,clk : in std_logic;
        Q,Qbar : out std_logic);
end tff;

architecture Behavioral of tff is
 signal tmp:std_logic;

begin
process(clk)
begin

if(clk'event and clk='1')then
if T='0'then
tmp<=tmp;
elsif T='1' then
tmp<=not tmp;
end if;
end if;
end process;
 Q<=tmp;
end Behavioral;
```


10.18.1.4 Implementation of SR Latch :

The circuit diagram of a S-R flip-flop using two NAND gates is shown in Fig. 10.18.4.

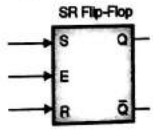


(C-4377) **Fig. 10.18.4 : SR FF using NAND gates**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity SRlatch is
  Port ( S,R : in std_logic; Q,Qbar : inout std_logic);
end SRlatch;
architecture structure Of SRlatch  is
component nand 2
        Port (A, B : in std_logic ; Y : out std_logic);
end component;
begin
    g1 : nand 2 port map (S, Qbar, Q);
    g2 : nand 2   port map (R, Q, Qbar);
end structure;
```

10.18.1.5 Implementation of SR Flip-Flop :

Logic diagram :



(C-5848) **Fig. 10.18.5 : Logic diagram**

Truth table :

Table 10.18.4 : Truth table

Q	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	unknown
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	unknown

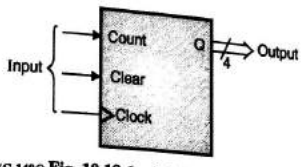
Scanned by CamScanner

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity SRFF is
   Port ( S,R ,clk: in std_logic;
        Q,Qbar : inout std_logic);
end SRFF;
architecture Behavioral of SRFF is
begin
process(clk)
variable tmp,z:std_logic;
begin
if (clk'event and clk='1') then
if(S='0' and R='0')then
tmp:=tmp;
elsif(S='1' and R='1')then
tmp:=z;
elsif(S='0' and R='1')then
tmp:='0';
else
tmp:='1';
end if;
end if;
Q<=tmp;
Qbar<=not tmp;
end process;
end Behavioral;
```

10.18.2 Implementation of Asynchronous Counters :

10.18.2.1 VHDL code for 4-bit Binary Up Counter with Asynchronous Clear :

- To perform additions on STD_Logic_Vectors, the statement USE IEEE.STD_LOGIC_UNSIGNED.ALL is required.
- To store the current count the signal **value** is used. Using the expression OTHERS ⇒ '0', value is assigned the value '0000' after declaration of **clear.** After declaration of **count,** value will be incremented by 1 on the next rising clock edge.
- Finally count in value is assigned to the counter output Q using a concurrent statement. It is declared outside the process block.

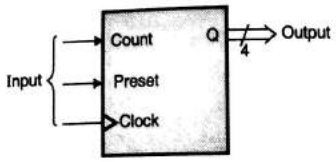


(C-1486) **Fig. 10.18.6 : 4 bit binary up counter**

```
library ieee;
use ieee.std_logic_1164.all;


entity counter is
    port (clock: in std_logic;
            clear: in std_logic;
            count: in std_logic;
          Q : out std_logic_vector(3 downto 0));
end counter;

architecture behav of counter is
signal value: std_logic_vector(3 downto 0);
begin
    process (clock, clear)
        begin
            if clear = '0' then
                value <= (others => '0');
            elsif (clock'event and clock='1') then


                if count = '1' then
                    value <= value + 1;
                end if;
            end if;
    end process;
Q <= value;
end behav;
```

Scanned by CamScanner

10.18.2.2 VHDL Code for 4-Bit Binary Down Counter with Asynchronous Preset :



(C-1487) **Fig. 10.18.7 : 4 bit binary down counter**

```
library ieee;
use ieee.std_logic_1164.all;


entity counter is
    port (clock: in std_logic;
          preset: in std_logic;
          count: in std_logic;
          Q : out std_logic_vector(3 downto 0));
end counter;
architecture behav of counter is
signal value: std_logic_vector(3 downto 0);
begin
    process (clock, clear)
      begin
          if preset = '1' then
              value <= (others => '1');
          elsif (clock'event and clock='1') then
             if count = '1' then
                 value <= value - 1;
             end if;
          end if;
    end process;
Q <= value;
end behav;
```


10.18.2.3 VHDL Code for 4-Bit Binary Up-Down Counter :

Note : In this code, the signal assignment statement Q < = Value is declared inside the process block that means it is now a sequential statement.

```
library ieee;
use ieee.std_logic_1164.all;

entity udcounter is
   port (clock: in std_logic;
         clear: in std_logic;
         count: in std_logic;
         down: in std_logic;
         Q: out integer range 0 to 15);
end udcounter;

architecture behav of udcounter is
begin
   process (clock, clear)
       variable value: integer range 0 to 15;
       begin
          if clear = '1' then
             value := 0;
          elsif (clock'event and clock='1') then
             if count = '1' then
                if down = '0' then
                    value := value + 1;
                else
                    value := value - 1;
                end if;
             end if;
          end if;
          Q <= value;
   end process;
end behav;
```

10.18.2.4 VHDL Code for 4-Bit BCD Up Asynchronous Counter :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Scanned by CamScanner

```
use IEEE.STD_LOGIC_ARITH.ALL;

entity bcdupcounter is
   Port ( clock_enable_B : in std_logic;
          clock : in std_logic;
          reset : in std_logic;
          output : out std_logic_vector(0 to 3));
end bcdupcounter;

architecture Behavioral of bcdupcounter is
 signal temp: std_logic_vector(0 to 3);
begin
 process(Clock,Reset)
  begin
    if Reset='1' then
      temp <= "0000";
    elsif(rising_edge(Clock)) then
      if Clock_enable_B='0' then
        if temp="1001" then
          temp<="0000";
        else
          temp <= temp + 1;
        end if;
      end if;
    end if;
  end process;
  Output <= temp;
end Behavioral;
```

10.18.2.5 VHDL Code for BCD Up-Down Asynchronous Counter :

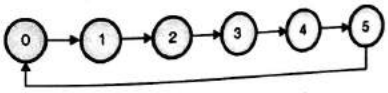
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity BCDupdown is
   Port ( clk,clr,up_down,sload : in std_logic;
```



```
          q : out std_logic_vector(3 downto 0));
end BCDupdown;

architecture Behavioral of BCDupdown is
  signal tmp: STD_LOGIC_VECTOR(3 downto 0);
begin
process(clk,clr,sload)
begin
    if(sload='1')then
    if(clr='1')then
    tmp<="0000";
    elsif(clk'event and clk='1')then
    if(up_down='1')then
    tmp<=tmp + 1 ;
    else
    tmp<=tmp - 1;
end if;
end if;
end if;
end process;
Q<=tmp;
end Behavioral;
```

10.18.3 Implementation of Synchronous Counters :

10.18.3.1 VHDL Code for Synchronous Mod-6 Counter :



(C-1489) **Fig. 10.18.8 : Mod-6 counter**

```
library ieee;
use ieee.std_logic_1164.all;

entity counter is
   port (clear: in bit;
         clock: in bit;
         count: buffer integer range 0 to 5);

end counter;
```

Scanned by CamScanner

```
architecture example of counter is
begin
    process
      begin
            wait until clock'event and clock = '1';
                  if (clear = '1' or count >= 5) then
                     count <= 0;
                  else
                        count <= count + 1;
                  end if;
      end process;
end example;
```

10.18.3.2 VHDL Code for 4-Bit Synchronous Counter :

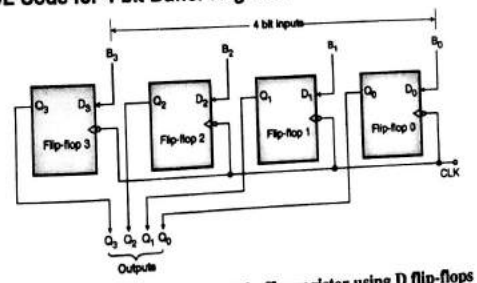
```
library ieee;
use ieee.std_logic_1164.all;


entity cntrnbit is
   generic(n : positive := 4);
      port(clock, reset, enable : in std_logic;
            count : out std_logic_vector((n-1) downto 0));
end cntrnbit;
architecture vk of cntrnbit is
signal count_int : std_logic_vector((n-1) downto 0);
 begin
   process
      begin
         wait until rising_edge(clock);
            if reset = '1' then
                  count_int <= (others => '0');
            elsif enable = '1' then
                  count_int <= count_int + 1;
            else
                  null;
            end if;
   end process;
count <= count_int;
end vk;
```


10.18.3.3 VHDL code for 3-Bit Synchronous Up/Down Counter With Asynchronous Clear Input :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity updown is
   Port ( clk,clr,ud,load : in std_logic;
         Q : out std_logic_vector(2 downto 0));
end updown;
architecture Behavioral of updown is
signal tmp:std_logic_vector(2 downto 0);
begin
 process(clk,clr,load)
 begin
 if(load='1') then
 if(clr='1')then
 tmp<="000" ;
 elsif(clk'event and clk='1')then
 if(ud='1')then
 tmp <=tmp+1;
 else
 tmp <=tmp-1;
 Q<=tmp;
 end if;
 end if;
end if;
 end process;
 Q<=tmp;
end Behavioral;
```

10.18.4 Implementation of Shift Registers :

10.18.4.1 VHDL Code for 4-bit Buffer Register :



(C-729(J)) **Fig. 10.18.9 : A four bit buffer register using D flip-flops**

Scanned by CamScanner

Behavioral model for 4-bit buffer register :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity fourbitreg is
   Port ( D : in std_logic_vector(0 to 3); clock,clear : in std_logic; Q : out std_logic_vector(0 to
3));
end fourbitreg;
architecture Behavioral of fourbitreg is
signal temp: std_logic_vector(0 to 3);
begin
  process(clock)
  begin
  if clear='0'then
  temp<="0000" ;
  elsif(clock='1'and clock'event) then
  temp<=D;
  end if;
  end process;
  Q<=temp;
end Behavioral;
```

Structural model for 4-bit buffer register :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fourbitregister is
   Port ( D : in std_logic_vector(0 to 3); clock,clear : in std_logic; Q : out std_logic_vector(0 to
3));
end fourbitregister;
architecture structural of fourbitregister is
 component DFF
 port(a,b,c:in std_logic;O:out std_logic);
 end component;
begin
   FF0: DFF port map(D(0),clock,clear,Q(0));
   FF1: DFF port map(D(1),clock,clear,Q(1));
   FF2: DFF port map(D(2),clock,clear,Q(2));
   FF3: DFF port map(D(3),clock,clear,Q(3));

end structural;
```


10.18.4.2 VHDL Code For Serial Input Serial Output Shift Register :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity SISO is
   Port ( si,clk,rst : in std_logic; q : inout std_logic_vector(2 downto 0); sout : out std_logic);
end SISO;
architecture structural of SISO is
component DFF is
Port( data,clk,rst: in std_logic;output : out std_logic);
end component;
begin
d0:DFF port map (si, clk, rst, q(0));
d1:DFF port map (q(0), clk, rst, q(1));
d2:DFF port map (q(1), clk, rst, q(2));
d3:DFF port map (q(2), clk, rst, sout);
end structural;
```

10.18.4.3 VHDL Code for Serial In Parallel Out Shift Register :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity PISO is
   Port ( B : in std_logic_vector(3 downto 0); clk : in std_logic; load, si : in std_logic;
Q : inout std_logic_vector(3 downto 0));
end PISO;
architecture arch of PISO is
begin
  process
  begin
  wait until clk'event and clk='1';
  if load='1'then
  Q<=B;
  else
  Q(0) <=Q(1);
  Q(1)<=Q(2);
  Q(2) <=Q(3);
  Q(3)<=si;
```

Scanned by CamScanner

```
  end if;
  end process;
end arch;
```

10.18.4.4 VHDL Code for Parallel In Serial Out Shift Register :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity PISO is
   Port ( B : in std_logic_vector(3 downto 0); clk : in std_logic;     load, si : in std_logic;
        Q : inout std_logic_vector(3 downto 0));
end PISO;
architecture arch of PISO is
begin
  process
  begin
  wait until clk'event and clk='1';
  if load='1'then
  Q<=B;
  else
  Q(0) <=Q(1)     ;
  Q(1)<=Q(2);
  Q(2) <=Q(3);
  Q(3)<=si;
  end if;
  end process;
end arch;
```

10.18.4.5 VHDL Code for Parallel In Parallel Out Shift Register :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity PIPO is
   Port ( clk : in std_logic; D : in std_logic_vector(3 downto 0); Q : out std_logic_vector(3 downto
0));
end PIPO;
architecture arch of PIPO is
```



```
begin
  process(clk)
  begin
  if(clk'event and clk='1')then
  Q<=D;
  end if;
  end process;
end arch;
```

10.19 Advantages of VHDL :

1. VHDL allows designers to quickly develop designs requiring tens of thousands of logic gates.
2. For describing complex logic, VHDL provides powerful high level constructs.
3. VHDL supports multiple level of hierarchy and modular design methods.
4. For design and simulation only one language i.e. VHDL can be used.
5. VHDL allows formation of device independent designs which are convenient for many vendors.
6. VHDL allows user to pick any synthesis tool.
7. VHDL is multipurpose i.e. once calculation block is created it can be used in many other projects.

10.19.1 Disadvantages of VHDL :

VHDL language is not a low level language i.e. gate level program. It is not suitable for verification of basic objects like gates. Because in VHDL these basic elements are readily available in VHDL.

10.19.2 Applications of VHDL :

1. It is used in electronic design automation to describe mixed signal system such as FPGA (Field Programmable Gate Arrays) and integrated circuits.
2. VHDL can be used as a general purpose parallel programming language.

Review Questions

Q. 1 Write a short note on VHDL.

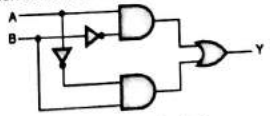
Q. 2 Distinguish between entity declaration and architecture declaration.

Q. 3 State the VHDL rules and characteristics.

Q. 4 Write a short note on design entity.

Q. 5 Write the entity declaration constructs in VHDL for AND and OR gates.

Q. 6 Write the entity declaration constructs in VHDL for the XOR circuit shown in Fig. Q. 10.6.



(C-1490) Fig. Q. 10.6

Scanned by CamScanner

Q. 7 Write a short note on : Data flow modelling.

Q. 8 Write the architecture body for a 2 input X-OR gate using the data flow model assuming the delay to be equal to 5 mS.

Q. 9 Write a short note on : Behavioural modelling.

Q. 10 Write the architecture body using behavioural model for the SRFF using NAND gates. Assume the delay per gate to be equal to 10 mS.

Q. 11 Explain the following in relation with VHDL programming 1. Data objects 2. Generic.

Q. 12 State advantages, disadvantages and applications of VHDL.

Q. 13 Write short note on : Data objects.

Q. 14 Write short note on : VHDL attributes.

□□□

CHAPTER 11

Module 6

Digital Logic Families

Syllabus :

Introduction, Terminologies like Propagation delay, Power consumption, Fan-in and Fan-out, Current and Voltage parameters, Noise Margin with respect to TTL and CMOS Logic and their comparison.

11.1 Introduction :

- Various digital ICs available in market belong to various types. These types are known as "families".
- Based on the components and devices internally used, the digital IC families are named as RTL (Resistor Transistor Logic), TTL (Transistor Transistor Logic), DTL (Diode Transistor Logic), CMOS etc.

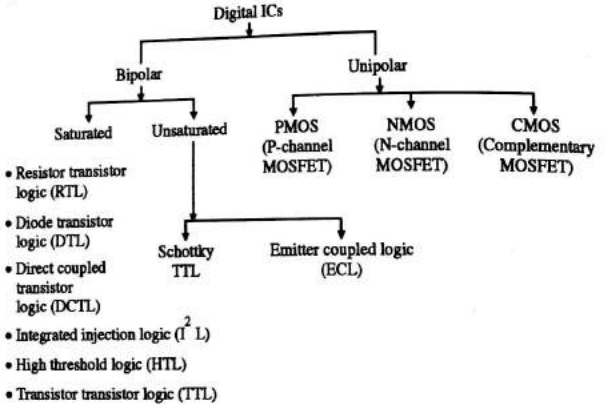
11.1.1 Classification Based on Circuit Complexity :

- The logical circuits of different types are available in the integrated circuits. Depending on the level of complexity the integrated circuit are classified into four categories as follows :
  1. SSI : Small Scale Integration.
  2. MSI : Medium Scale Integration.
  3. LSI : Large Scale Integration.
  4. VLSI : Very Large Scale Integration.
- The number of components (diodes, transistors, gates etc.) in SSI will be the lowest and that in VLSI will be the highest.
  1. Small Scale Integration (SSI) < 10 components.
  2. Medium Scale Integration (MSI) < 100 components.
  3. Large Scale Integration (LSI) > 100 components.
  4. Very Large Scale Integration (VLSI) > 1000 components.

Scanned by CamScanner

11.2 Classification of Logic Families :

11.2.1 Classification Based on Devices Used :



(C-205) **Fig. 11.2.1 : Classification of logic families**

- The two basic techniques for manufacturing ICs are :
  1. Bipolar technology
  2. Unipolar devices-Metal Oxide Semiconductor (MOS) technology.
- The classification of logic families is shown in Fig. 11.2.1.

Bipolar families :

- The bipolar families of logic circuits use the bipolar transistors fabricated on the chip. That means all the gates belonging to the bipolar family use the transistorised circuits.
- In the bipolar category there are three basic families called, Diode Transistor Logic (DTL), Transistor Transistor Logic (TTL) and Emitter Coupled Logic (ECL).
- DTL uses diodes and transistors, TTL uses transistors as the main elements. TTL has become the most popular family in SSI (Small Scale Integration) and MSI (Medium Scale Integration) chips, while ECL is the fastest logic family which is used for high speed applications.
- In the "Bipolar saturated" logic families, the bipolar transistors are used as the main device. It is used as a switch and operated in the saturation or cutoff regions.
- TTL is an example of saturated bipolar logic.
- In the unsaturated bipolar logic, the bipolar transistors are not driven into hard saturation. This increases the speed of operation. So the unsaturated bipolar ICs such as Schottky TTL and ECL (emitter coupled logic) are much faster as compared to TTL.
- All these ICs are fabricated on silicon chips using different fabrication technologies.

Unipolar families :

- The MOS family use fabricates MOS Field Effect Transistors (MOSFETs) fabricated on the chip. So all the gates belonging to the MOS family use the MOSFET based circuits.
- In the MOS category there are three logic families namely PMOS (p-channel MOSFETs) family, NMOS (n-channel MOSFETs) family and CMOS (complementary MOSFETs) family.
- PMOS is the oldest and slowest type. NMOS is used for the LSI (large scale integration) field i.e. for the microprocessors and memories.
- CMOS which uses a push pull arrangement of n-channel and p-channel MOSFETs, is extensively used where low power consumption is needed such as in pocket calculators.

11.3 Characteristics of Digital ICs :

Eventhough there are various logic families, the general characteristics, their definitions, and terminologies used for all of them have been standardized. So let us discuss some of the most important general characteristics that are applicable to all the digital ICs first and then discuss them for some particular families.

11.3.1 Voltage and Current Parameters :

MU : May 04, Dec. 09

University Questions

Q. 1 Sketch a 2 Input TTL NAND gate circuit. Define I_{OL}, I_{IL}, t_{PHL}, V_{IH}, V_{OL}, V_{IL} **(May 04, 10 Marks)**

Q. 2 Write short note on current and voltage parameters of logic gates. **(Dec. 09, 10 Marks)**

Voltage parameters (Threshold levels) :

Ideally the input voltage levels of 0 V and + 5 V (for TTL) are called as logic 0 and 1 levels respectively. However practically we won't always observe or obtain the voltage levels matching exactly to these values. Therefore it is necessary to define the worst case input voltages.

1. **$V_{IL\,(max)}$ - Worst case low level input voltage :**

   This is the maximum value of input voltage which is to be considered as a logic 0 level. If the input voltage is higher than $V_{IL\,(max)}$, then it will not be treated as a low (0) input level.

2. **$V_{IH\,(min)}$ - Worst case high level input voltage :**

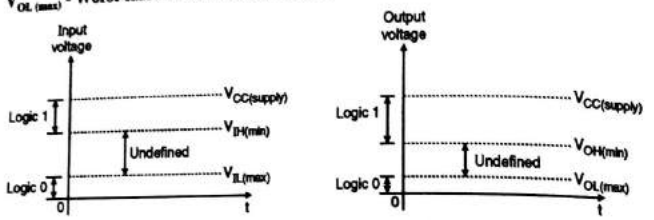
   This is the minimum value of the input voltage which is to be considered as a logic 1 level. If the input voltage is lower than $V_{IH\,(min)}$, then it will not be treated as a High (1) input.

3. **$V_{OH\,(min)}$ - Worst case high level output voltage :**

   This is the minimum value of the output voltage which will be considered as a logic HIGH (1) level. If the output voltage is lower than this level then it won't be treated as a HIGH (1) output.

Scanned by CamScanner

$V_{OL\,(max)}$ - Worst case low level output voltage :

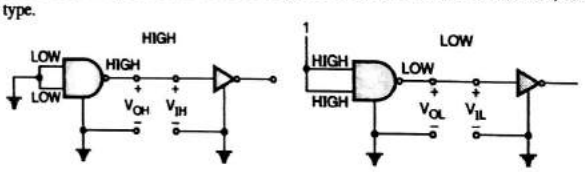


(a) Input voltage parameters **(b) Output voltage parameters**

(C-206) **Fig. 11.3.1 : Voltage parameters**

This is the maximum value of the output voltage which will be considered as a logic LOW (0) level. If the output voltage is higher than this value then it won't be treated as a LOW (0) output. All the voltage parameters are shown in Fig. 11.3.1.

The voltage parameters can be shown on the digital circuit consisting of gates as shown in Fig. 11.3.2. Note that, the NAND and NOT gates shown, can be of TTL, ECL, CMOS or any other type.



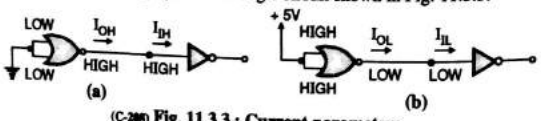
(C-207) **Fig. 11.3.2 : Voltage parameters on a logic circuit**

urrent parameters :

- **I_{IL} - Low level input current :** It is the current that flows into the input terminals when a low level input voltage in the specified range is applied.
- **I_{IH} - High level input current :** It is the current that flows into the input terminals when a high level input voltage in the specified range is applied.
- **I_{OL} - Low level output current :** This is the current that flows out of the output when the output voltage happens to be in the specified low (0) voltage range and a specified load is applied.
- **I_{OH} - High level output current :** This is the current flowing from the output when the output voltage happens to be in the specified HIGH (1) voltage range and a specified load is applied.

If the output current flows into the output terminal then it is called as a **sinking current** and if the output current flows out of the output terminal then it is called as a **sourcing current.**

The current parameters are displayed on the logic circuit shown in Fig. 11.3.3.



(C-208) **Fig. 11.3.3 : Current parameters**

Note that the actual current directions can be opposite to those shown in Fig. 11.3.3; depending on the logic family.

Note that current flowing into a node or device is considered positive and current flowing out of node or device is considered negative.

11.3.2 Fan-in and Fan-out :

MU : Dec. 13, May 14, Dec. 16

University Questions

Q. 1 Define Fan-in and Fan out. **(Dec. 13, 2 Marks)**

Q. 2 Define parameters for CMOS family : 1. Fan out 2. Fan in. **(May 14, 5 Marks)**

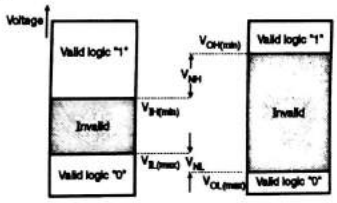
Q. 3 Compare TTL and CMOS with respect to speed, power dissipation, fan-in and fan-out and also define these terms. **(Dec. 16, 4 Marks)**

Fan-In : Fan-in is defined as the number of inputs a gate has. For example a two input gate will have a fan-in equal to 2.

Fan-out : Fan-out is defined as the maximum number of inputs of the same IC family that a gate can drive without falling outside the specified output voltage limits. Higher fan out indicates higher the current supplying capacity of a gate. For example, a fan out of 5 indicates that the gate can drive (supply current to) at the most 5 inputs of the same IC family.

11.3.3 Noise Margin :

- To understand the meaning of the term "Noise Margin" or "Noise Immunity", refer to the input and output voltage profiles shown in Fig. 11.3.4.
- Noise is an unwanted electrical disturbance which may induce some voltage in the connecting wires used between two gates or from a gate output to load.
- **Noise immunity** is defined as the ability of a logic circuit to tolerate the noise without causing the output to change undesirably.
- A quantitative measure of noise immunity of a logic family is known as **noise margin.**



(a) Input profile **(b) Output profile**

(C-210) **Fig. 11.3.4**

- In order to avoid the effects of noise voltage, the designers adjust the voltage levels $V_{OH\,(min)}$ and $V_{IH\,(min)}$ to different levels with some difference between them as shown in Fig. 11.3.4.
- The difference between $V_{OH\,(min)}$ and $V_{IH\,(min)}$ is known as the **high level noise margin V_{NH}.**
- Similarly the difference between $V_{IL\,(max)}$ and $V_{OL\,(max)}$ is called as the **low level noise margin V_{NL}.**

High level noise margin, $V_{NH} = V_{OH\,(min)} - V_{IH\,(min)}$

Low level noise margin, $V_{NL} = V_{IL\,(max)} - V_{OL\,(max)}$

- When a high logic output is driving a logic circuit input, any negative noise spike greater than V_{NH} can force the voltage to reduce into the invalid range.
- Similarly when a low logic output is driving a logic circuit input, any positive noise spikes greater than V_{NL} can force the voltage to go into the invalid state.

11.3.4 Propagation Delay (Speed of Operation) :

MU : May 04

University Questions

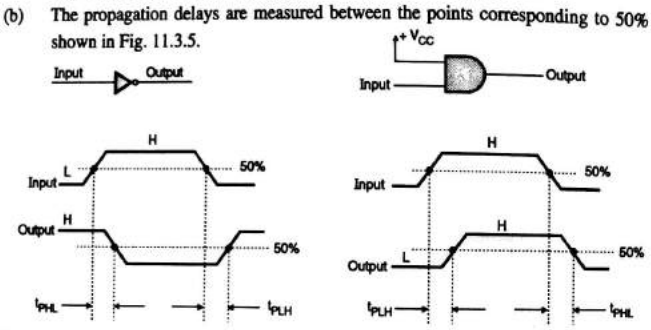
Q. 1 Sketch a 2 input TTL NAND gate circuit. Define I_{OL}, I_{IL}, t_{PHL}, V_{IH}, V_{OL}, V_{IL}. **(May 04, 10 Marks)**

- The output of a logic gate does not change its state instantaneously when the state of its input is changed.
- There is a time delay between these two time instants, which is called as the propagation delay.

Scanned by CamScanner

- Thus propagation delay is defined as time delay between the instant of application of an input pulse and the instant of occurrence of the corresponding output pulse. This is shown in Fig. 11.3.5.
- From Fig. 11.3.5 it is observed that there are two propagation delays.

1. t_{PHL} : It is the propagation delay measured when the output is making a transition from HIGH (1) to LOW (0) state.
2. t_{PLH} : This is the propagation delay measured when the output makes a transition from LOW (0) to HIGH (1) state.
   (a) The values of t_{PHL} and t_{PLH} are not always equal. If they are not equal then the one which is higher is considered as the propagation delay time of the gate.
   (b) The propagation delays are measured between the points corresponding to 50% levels as shown in Fig. 11.3.5.



(a) Propagation delays for an inverter **(b) Propagation delays for an AND gate**

(C-211) **Fig. 11.3.5**

- Ideally propagation delay should be zero and practically it should be as short as possible.
- The values of propagation delays are used to measure how fast a logic circuits is.
- For example, a logic circuit with a propagation time of 5 nS will be a faster logic circuit than the one with 10 nS propagation time, under the specified load conditions.

11.3.5 Power Dissipation :

MU : Dec. 16

University Questions

Q. 1 Compare TTL and CMOS with respect to speed, power dissipation, fan-in and fan-out and also define these terms. **(Dec. 16, 4 Marks)**

- Due to applied voltage and currents flowing through the logic ICs, some power will be dissipated in it, in the form of heat.
- This power is in milliwatts. Care should be taken to reduce the power dissipation taking place in the logic IC in order to protect the IC against damage due to excessive temperature, to reduce the loading on power supplies etc.
- Another importance of power dissipation is that the product of power dissipation and propagation time is always constant.
- Therefore if we reduce the power dissipation may lead then the propagation delay will increase in order to keep their product constant.
- Usually there is only one power supply terminal on any IC. It is denoted by V_{CC} for the TTL ICs and V_{DD} for the CMOS ICs.

- The power drawn by an IC from the power supply is given by,

$$P = V_{CC} \times I_{CC}$$

where I_{CC} is the current drawn from the power supply.

11.3.6 Operating Temperature :

- The temperature range acceptable for the consumer and industrial applications is 0° to 70° C and that for the military applications is – 55° C to 125° C.
- The performance of gates will be in the specified limits only if the temperature is in these ranges.

11.3.7 Figure of Merit (Speed Power Product SPP) :

- The figure of merit of a logical family is the product of power dissipation and propagation delay. It is called as the speed power product. The speed is specified in seconds and power is specified in W.

$$\therefore \text{ Figure of merit } = \text{ Propagation delay time} \times \text{Power dissipation.}$$

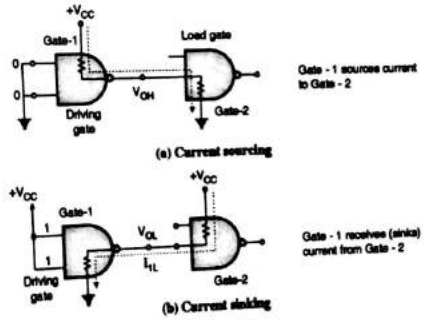
- Ideally the value of figure of merit is zero and practically, it should be as low as possible.
- Figure of merit is always a compromise between speed and power dissipation. That means if we try to reduce the propagation delay then the power dissipation will increase and vice-versa.

11.3.8 Invalid Voltage Levels :

- The operation of a logic circuit will be proper if and only if its input voltage levels are kept outside the invalid voltage range.
- That means the input voltage should be either lower than $V_{IL\,(max)}$ or higher than $V_{IH\,(min.)}$.
- The invalid input voltage will produce an unpredictable output response. Therefore it should be avoided.
- When the output is overloaded, there is a possibility of output voltage going into the invalid range.

11.3.9 Current Sourcing and Current Sinking :

- The current sourcing action is illustrated in Fig. 11.3.6(a). Gate-1 acts as a source and Gate-2 acts as load.
- The output of Gate-1 is high. It supplies a current I_{IH} to the input of Gate-2. This is called as the current sourcing action.
- The current sinking action has been demonstrated in Fig. 11.3.6(b). Gate-2 which is the load gate acts as a load.



(C-1000) **Fig. 11.3.6**

- As soon as the output of Gate-1 goes low, the current starts flowing into the output terminal of Gate-1 as shown in Fig. 11.3.6(b). Thus when gate-1 is accepting the current through its output terminal, it is said that the current sinking is taking place.

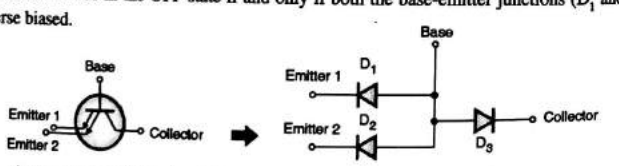
Scanned by CamScanner

11.4 TTL Logic :

- The long form of TTL is transistor logic. The digital ICs in the TTL family use only transistors as their basic building block.
- TTL ICs were first developed in 1965 and they were known as "standard TTL". This version of TTL circuits is not practically used now due to availability of advanced versions.
- The standard TTL has been improved to a great extent over the years.
- TTL devices are still used as "glue" logic which connects more complex devices in digital systems.
- The bipolar TTL family as a whole is now becoming obsolete, but we can study it in order to understand basic important concepts in general about the logic families.

11.4.1 The Multiple Emitter Transistor :

- In the TTL circuits that we are going to discuss, use a very special type of transistor.
- The normal transistor has only three terminals namely collector, base and emitter. But this special transistor has more than one emitters, as shown in Fig. 11.4.1(a) and its equivalent circuit is shown in Fig. 11.4.1(b). The number of emitters is equal to the number of inputs of the gate.
- The multiple emitter input transistor can have upto eight emitters, for an eight input NAND gate.
- In the equivalent circuit of Fig. 11.4.1(b), diodes D_1 and D_2 represent the two base to emitter junctions whereas D_3 represents the collector to base junction.
- In the forthcoming discussions, we are going to replace the multiple emitter transistor by its equivalent circuit shown in Fig. 11.4.1(b).
- This transistor can be turned ON by forward biasing either (or both) the diodes D_1 and D_2.
- This transistor will be in the OFF state if and only if both the base-emitter junctions (D_1 and D_2) are reverse biased.



(a) Multiple emitter transistor (b) Equivalent circuit

(C-1011) **Fig. 11.4.1**

Standard TTL : The 7400 TTL series is known as the standard TTL series. The TTL gates that we are going to discuss belong to this family.

11.4.2 Two Input TTL-NAND Gate (Totempole Output) :

MU : May 04, May 06, May 07, Dec. 07, Dec. 08, May 09, Dec. 10, Dec. 11

University Questions

Q. 1 Sketch a 2 input TTL NAND gate circuit. Define I_{OL}, I_{IL}, t_{PHL}, V_{IH}, V_{OL}, V_{IL}. **(May 04, 10 Marks)**

Q. 2 Draw neat circuit diagram of two input TTL NAND gate and explain its operation. **(May 06, 10 Marks)**

Q. 3 Draw neat diagram of two-input TTL NAND gate and explain its operation. Also draw transfer characteristics. **(May 07, 10 Marks)**

Q. 4 Draw neat diagram of two input TTL NAND gate and explain its operation. **(Dec. 07, 8 Marks)**

Q. 5 Draw a 2-input TTL NAND gate and explain its operation. **(Dec. 08, Dec. 10, 10 Marks)**

Q. 6 Draw the 2-input TTL NAND gate and explain. List important characteristics of TTL family. **(May 09, 10 Marks)**

Q. 7 Explain with a neat diagram 2 input TTL NAND gate in detail. **(Dec. 11, 10 Marks)**

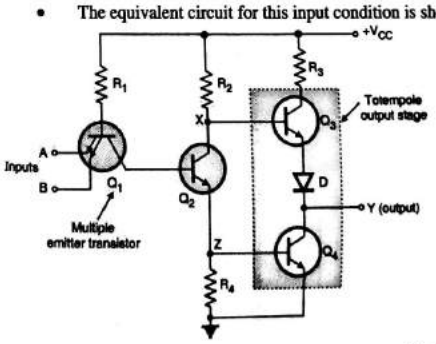
A two input TTL-NAND gate is shown in Fig. 11.4.2. A and B are the two inputs while Y is the output terminal of this NAND gate.

Operation : In order to understand the operation of this circuit, let us replace transistor Q_1 by its equivalent circuit as shown in Fig. 11.4.3.

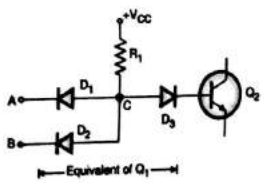
1. A and B are the input terminals. The input voltages A and B can be either LOW (zero volt ideally) or HIGH (+ V_{CC} ideally).
2. **A and B both LOW :** If A and B both are connected to ground, then both the B-E junctions of transistor Q_1 are forward biased.
   - Hence diodes D_1 and D_2 in Fig. 11.4.3 will conduct to force the voltage at point C in Fig. 11.4.3 to 0.7 V.
   - This voltage is insufficient to forward bias base-emitter junction of Q_2 due to the presence of D_3. Hence Q_2 will remain OFF.
   - Therefore its collector voltage V_X rises to V_{CC}.
   - As transistor Q_3 is operating in the emitter follower mode, output Y will be pulled up to high voltage.

   $\therefore$ Y = 1 (HIGH) ...For A = B = 0 (LOW)

   - The equivalent circuit for this input condition is shown in Fig. 11.4.4(a).



(C-1012) **Fig. 11.4.2 : Two input TTL NAND gate**



(C-1012) **Fig. 11.4.3 : Transistor Q_1 is replaced by its equivalent**

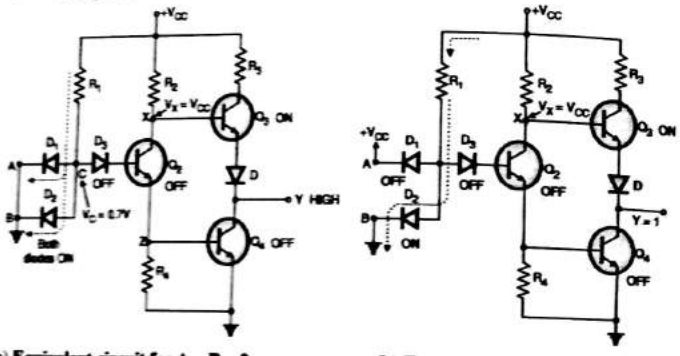
3. **Either A or B LOW :** If any one input (A or B) is connected to ground with the other terminal left open or connected to + V_{CC}, then the corresponding diode (D_1 or D_2) will conduct.
   - This will pull down the voltage at "C" to 0.7 V. (Fig. 11.4.3)
   - This voltage is insufficient to turn ON D_3 and Q_2. So it remains OFF.

Scanned by CamScanner

- So collector voltage V_X of Q_2 will be equal to V_{CC}. This voltage acts as base voltage for Q_3.
- As Q_3 acts as an emitter follower, output Y will be pulled to V_{CC}.

$$\therefore Y = 1 \begin{cases} \text{if } A = 0 \text{ and } B = 1 \\ \text{if } A = 1 \text{ and } B = 0 \end{cases}$$

- The equivalent circuit for this mode is shown in Fig. 11.4.4(b).



(a) Equivalent circuit for A = B = 0 **(b) Equivalent circuit for A = 1, B = 0**

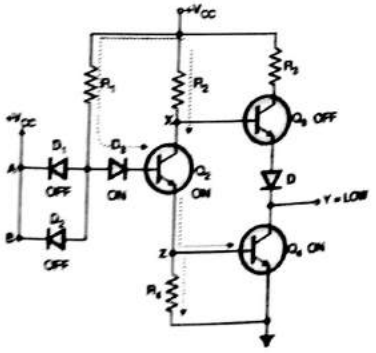
(C-1013) **Fig. 11.4.4**

A and B both HIGH : If A and B both are connected to + V_{CC}, then both the diodes D_1 and D_2 will be reverse biased and do not conduct. Therefore voltage at point "C" i.e. at the anode of D_3 increases to a sufficiently high value.

- Therefore diode D_3 is forward biased and base current is supplied to transistor Q_2 via R_1 and D_3 as shown in Fig. 11.4.4(c).
- As Q_2 conducts, the voltage at X will drop down and Q_3 will be OFF, whereas voltage at Z (across R_2) will increase to a sufficient level to turn ON Q_4.
- As Q_4 goes into saturation, the output voltage Y will be pulled down to a low voltage.

$\therefore Y = 0$ ...For A = B = 1

- The equivalent circuit for this mode of operation is shown in Fig. 11.4.4(c).



(C-1014) **Fig. 11.4.4(c) : Equivalent circuit for A = B = 1**

- This discussion reveals that the circuit operates as a NAND gate.

11.4.3 Totem-pole (Active Pull up) Output Stage :

University Questions MU : Dec. 09

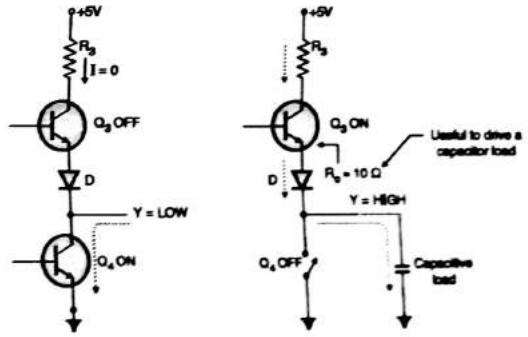
Q. 1 Write short note on totempole output stage of TTL gate. **(Dec. 09, 10 Marks)**

- The arrangement of Q_3 and Q_4 on the output side of a TTL NAND gate is called as the totem-pole arrangement. It is possible in TTL gates to speed up the charging of output capacitance without corresponding increase in the power dissipation with help of **Totem-pole** output stage.
- The Totem-pole output is also known as **active pull-up.**

Advantages of totem-pole output stage :

The advantages of using the totem-pole output stage are as follows :

1. With Q_3 in the circuit, the current flowing through R_3 will be equal to zero when the output Y = 0, that means when Q_4 is ON, as shown in Fig. 11.4.5(a). This is important because it reduces the power dissipation taking place in the circuit.
2. Another advantage of totem-pole arrangement is when the output Y is HIGH. Here Q_3 is ON and acting in the emitter follower mode. It will therefore have a very low output impedance (typically 10 Ω). Therefore the output time constant will be very short for charging up any capacitive load on the output as shown in Fig. 11.4.5(b).



(a) No power dissipation in R_3 **(b) Low output resistance**

(C-1015) **Fig. 11.4.5**

Disadvantages of Totem-pole output :

1. Q_4 in the totem-pole output turns OFF more slowly than Q_3 turns ON.
2. So before Q_4 is completely turned OFF, Q_3 will come into conduction. So for a very short duration of few nanoseconds, both the transistors will be simultaneously ON.
3. This is called as **cross conduction** and it will draw relatively large current (30 to 40 mA) from the 5 V supply.

Scanned by CamScanner

unction of diode D :

To understand the function of "D" consider the circuit without D, as shown in Fig. 11.4.6.

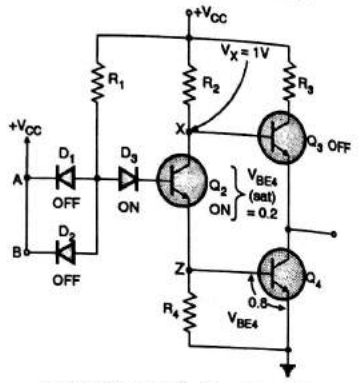
Assume that A = B = 1. So D_1 and D_2 are OFF, D_3 ON and Q_2 is in saturation.

Hence Q_4 is ON (saturation). Therefore voltage at the collector of Q_2 is given by,

$$V_X = V_{BE4\,(sat)} + V_{CE2\,(sat)}$$
$$= 0.8 + 0.2 = 1 \text{ Volt}$$

V_X = 1 V will be sufficient to turn ON Q_3 which actually should not happen. So in order to keep Q_3 in the OFF state diode D is added in the circuit.

It is important to avoid simultaneous conduction of Q_3 and Q_4, because it will lead to cross conduction and will increase power dissipation.



(C-1016) **Fig. 11.4.6 : Function of D**

Thus D_3 is used for successfully avoiding the cross conduction.

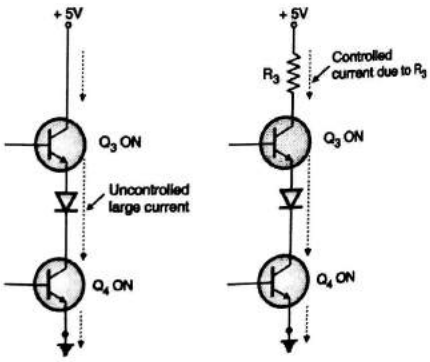
nction of R_3 :

As discussed earlier, transistor Q_4 in the totem pole arrangement turns OFF more slowly than Q_3 turns ON.

So for few nanoseconds both Q_3 and Q_4 will conduct simultaneously when the output Y is changing state from low to high. This is called as **cross conduction.**

During the cross conduction, if R_3 is not used then there will be no current limiting element in series with Q_3 and Q_4 and a heavy current will be drawn from the source as shown in Fig. 11.4.7(a) which can damage the IC.

This can be avoided by limiting the current by inserting resistor R_3 in series with Q_3, as shown in Fig. 11.4.7(b).

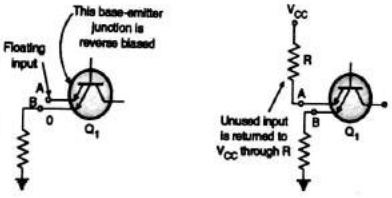


(a) R_3 not used **(b) R_3 used**

(C-1017) **Fig. 11.4.7 : Function of R_3**

11.4.4 Unconnected Inputs :

- If any input of a TTL gate is left open, disconnected or floating, then the corresponding base emitter junction of the input transistor Q_1 is not forward biased as shown in Fig. 11.4.8(a).
- Therefore the open or floating input is equivalent to a logical 1 is applied to that input.
- Hence in TTL ICs all the unconnected inputs are treated as logical 1s.
- However, the unused inputs should either be connected to some used input (s) or returned to V_{CC} through a suitable resistor as shown in Fig. 11.4.8(b).

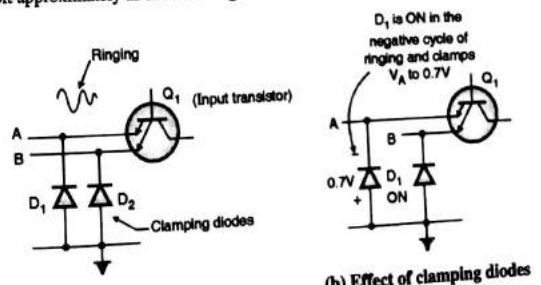


(a) Unused input **(b) Standard connection**

(C-1018) **Fig. 11.4.8**

11.4.5 Clamping Diodes :

- The TTL inputs should not be subjected to negative voltages. Under normal operating conditions this gets managed easily.
- But if fast voltage transitions are applied at the input, then there is a possibility of ringing (sinusoidal oscillations with positive and negative half cycles). Due to ringing, the inputs will be subjected to negative voltages.
- To suppress this ringing, clamping diodes are generally connected externally in all the TTL circuits as shown in Fig. 11.4.9(a).
- These are fast recovery diodes. They are forward biased during the negative half cycles of the ringing sinusoidal waveform. Hence the negative input voltage will be restricted (clamped) to – 0.7 Volt approximately as shown in Fig. 11.4.9(b).

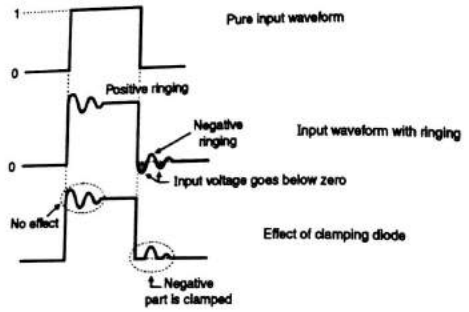


(a) Clamping diodes **(b) Effect of clamping diodes**

(C-1019) **Fig. 11.4.9**

Scanned by CamScanner

- The clamping action can be better understood by referring to Fig. 11.4.10.



(C-1020) **Fig. 11.4.10 : Effect of clamping diodes**

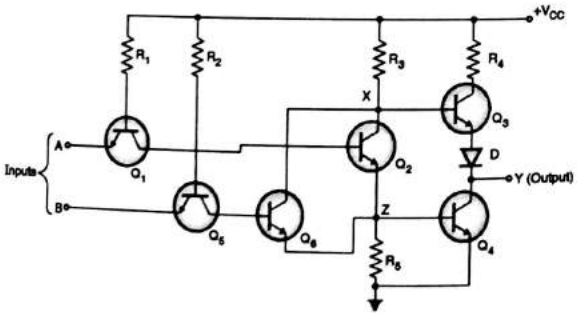
11.4.6 TTL NOR Gate :

MU : Dec. 06

University Questions

Q. 1 Explain two input TTL Nor gate and draw the circuit. **(Dec. 06, 10 Marks)**

- In order to get the 2-input NOR operation, the manufacturers modify the 2-input NAND gate as shown in Fig. 11.4.11.
- In this circuit, transistor Q_5 and Q_6 are newly added transistor. Rest of the circuit is same as that of the TTL-NAND gate.



(C-1021) **Fig. 11.4.11 : TTL-NOR gate**

Operation :

1. **A = B = 0 :**
   - Both transistors Q_1 and Q_5 conduct. Hence Q_2 and Q_6 both remain off. Hence potential at point "X" goes high and that at point "Z" is low.
   - Hence Q_3 is on and Q_4 is off. Hence output Y = 1.
2. **A = 0, B = 1 :**
   - Transistor Q_1 conducts but Q_5 remains off. Hence Q_2 remains off but Q_6 conducts.
   - Since Q_2 and Q_6 are in parallel, the conducting Q_6 will bring the potential at "X" low and that at "Z" will go up.
   - Hence Q_3 will be off and Q_4 turns on. Hence output Y = 0.
3. **A = 1, B = 0 :**
   - Q_1 remains off but Q_5 turns on. Hence Q_2 will conduct and Q_6 remains off.
   - The potential at X comes down and that at Z goes up. So Q_3 will be off and Q_4 will be on. Hence Y = 0.
4. **A = B = 1 :**
   - Both Q_1 and Q_5 are off. Hence Q_2 and Q_6 conduct. Potential at X decreases and that at Z is increased.
   - So Q_3 is off and Q_4 turns on. Hence Y = 0.

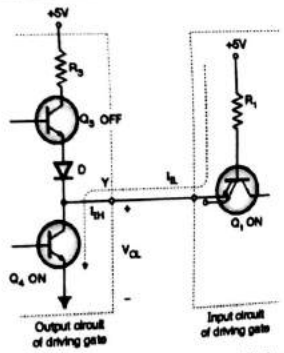
11.4.7 5400 Series :

- The temperature range for the devices in 7400 series is from 0 to 70°C, over a supply voltage range of 4.75 to 5.25 V.
- So 7400 series is used for the commercial applications.
- But 5400 TTL series is developed specially for the military applications.
- The devices of 5400 series operate over the temperature range of – 55 to 125°C and over the supply voltage range of 4.5 to 5.5 Volts.

11.5 Sourcing and Sinking in TTL :

11.5.1 Current Sinking Action :

- The current sinking will take place when output Y = 0 as shown in Fig. 11.5.1(a).
- The current flows from the input stage of load gate into the output stage of driving gate.
- Transistor Q_4 of the driving gate acts as a closed switch because it is in saturation. This will forward bias the base emitter junction of Q_1 of the load gate.



(C-1022) **Fig. 11.5.1(a) : With Y = 0 the driving gate sinks current from the load gate**

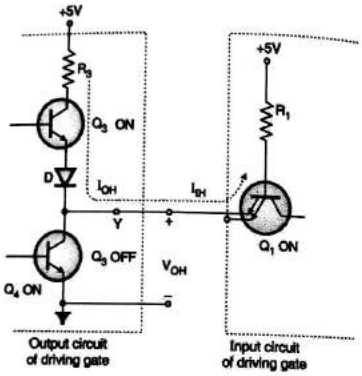
Scanned by CamScanner

11.5.2 Current Sourcing Action :

- Current sourcing will take place when the output of the driving gate (Y) becomes HIGH, i.e. Y = 1. Refer Fig. 11.5.1(b).
- Q_3 will be ON and Q_4 will be OFF. So the base emitter junction of Q_1 of the load gate will be reverse biased.
- Therefore the reverse leakage current of Q_1 will be supplied (sourced) by the driving gate as shown in Fig. 11.5.1(b).

Note that this current is extremely small, typically of the order of 10 μA.

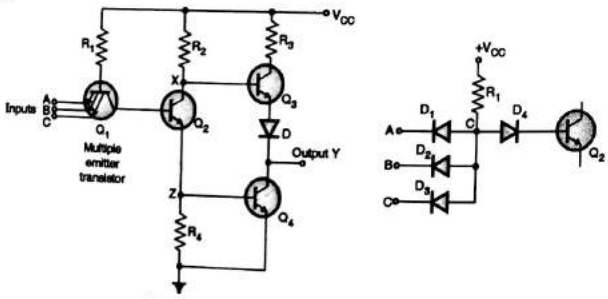
Transistor Q_3 is called as current sourcing or pull up transistor and Q_4 is called as current sinking or pull down transistor.



(C-1023) **Fig. 11.5.1(b) : Current sourcing in TTL**

11.5.3 Three Input TTL NAND Gate :

The circuit diagram of a three input NAND gate is as shown in Fig. 11.5.2. Note that the multiple emitter transistor has three emitter terminals which act as the inputs A, B and C to the NAND gate.



(b) Equivalent circuit of the multiple emitter transistor Q_1

(C-1685) **Fig. 11.5.2 : Three input TTL NAND gate**

The principle of operation of this circuit is exactly same as that of the two input NAND gate discussed in section 11.4.2.

- If at least one of the inputs is low (0) then at least one of the diodes D_1, D_2, D_3 in Fig. 11.5.2(b) will be conducting. Hence the voltage at point C will be clamped to 0.7 V. Hence Q_2 will be off. So Q_3 will conduct and the output Y = 1 (HIGH)

$$\therefore \; Y = 1 \text{ if at least one input is 0.}$$

- If A = B = C = 1 then D_1, D_2, D_3 will be off. So Q_2 will be turned on. So Q_3 will be turned off and Q_4 will be turned on. So the output Y = 0 (LOW).

$$\therefore \; Y = 0 \text{ if all the inputs are 1.}$$

- Table 11.5.1 shows the truth table and the status of various transistors in the circuit.

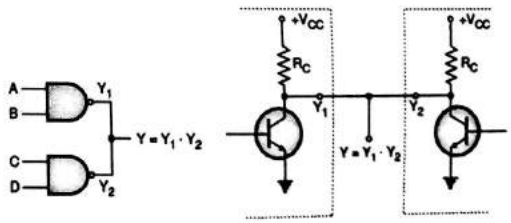
Table 11.5.1 : Truth table of the 3-input NAND gate

Input			Status of various transistors			Output
A	B	C	Q_2	Q_3	Q_4	Output Y
0	0	0	OFF	ON	OFF	1
0	0	1	OFF	ON	OFF	1
0	1	0	OFF	ON	OFF	1
0	1	1	OFF	ON	OFF	1
1	0	0	OFF	ON	OFF	1
1	0	1	OFF	ON	OFF	1
1	1	0	OFF	ON	OFF	1
1	1	1	ON	OFF	ON	0

11.6 Wired AND Connection (TTL) :

- If we connect the outputs of two gates directly to each other as shown in Fig. 11.6.1, then they are said to be wired ANDed with each other.
- If the outputs are wired ANDed, then the final output is given by,

$$Y = Y_1 \cdot Y_2 = (\overline{A \cdot B}) \cdot (\overline{C \cdot D}) = \overline{AB + CD}$$



(a) Wired AND connection (b) Wired AND connection with internal output stages shown

(C-1025) **Fig. 11.6.1**

Can we wire AND the gates having Totempole output stage ?

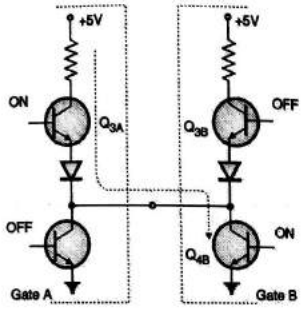
- The answer is No. Wired AND connection should not be used for the totem-pole output circuits because of the current spike problem.

Scanned by CamScanner

- TTL circuits with open collector outputs are available which can be used for wired AND connection.

Why cannot we connect the totempole outputs together ?

- The outputs of two TTL gates with totem pole output are shown to be connected together in Fig. 11.6.2.
- There are two reasons why such connection should be avoided.
- Assume that output of Gate A is HIGH so Q_{3A} is ON and output of Gate B is LOW so Q_{4B} is conducting. Since both transistors operate in saturation, they are equivalent to small resistances. Hence Q_{4B} acts as a very low resistive load and draws a very large current which is higher than its rated current.

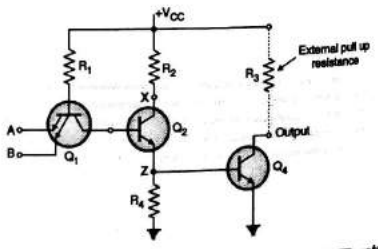


(C-1026) **Fig. 11.6.2 : Totempole outputs tied together**

- Hence the transistor may get damaged due to overheating, after sometime.
- Second problem caused by these high currents is that the current through Q_{4B} will produce a voltage drop V_{CE4B} to raise V_{OL} between 0.5 to 1 V. This is greater than the allowable $V_{OL(max)}$.
- Due to these two reasons, the totempole outputs should never be directly connected together.

11.7 Open Collector Outputs (TTL) :

- We have seen that the gates having totem-pole output cannot be wired ANDed.
- Such a connection becomes possible if another type output stage called open collector output is used.
- The circuit diagram of a 2-input NAND gate is shown in Fig. 11.7.1. You will realize that this is the same TTL NAND gate which we have discussed earlier but with R_3 and Q_3 removed.



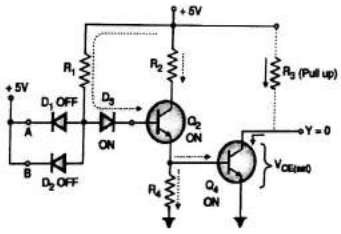
(C-1027) **Fig. 11.7.1 : Open collector 2 input NAND gate**

- The collector point of Q_4 is brought out as output as shown in Fig. 11.7.1, therefore it is called as open collector output.
- For proper operation it is necessary to connect an external resistance R_3 between V_{CC} and the open collector output as shown in Fig. 11.7.1. This resistance is called as pull up resistance.

Operation :

1. **With A = B = 0 :**
   - With A = B = 0, both the BE junctions of Q_1 are forward biased. So Q_2 remains OFF.
   - Hence no current flows through R_4. So $V_Z \approx 0$ V.

   - Therefore Q_4 is OFF and its collector voltage is equal to V_{CC}. So Y = 1 when A = B = 0.
2. **With A = 0, B = 1 OR A = 1, B = 0 :**
   - One of the BE junctions is forward biased (the one corresponding to 0 input).
   - So Q_2 is OFF and Q_4 also is OFF. So its collector voltage is equal to V_{CC}.
   - Therefore output Y = 1 when any one input is low.
3. **With A = B = 1 :**
   - When both the inputs are high, transistor Q_1 is turned OFF.
   - So Q_2 will be turned ON. Sufficient voltage is developed across R_4. Base current is applied to Q_4 and Q_4 goes into saturation.
   - So the output voltage is equal to $V_{CE\,(sat)}$ of Q_4 which is very small. Thus Y = 0 when A = B = 1. The equivalent circuit for this mode is shown in Fig. 11.7.2.



(C-1028) **Fig. 11.7.2 : Equivalent circuit of open collector 2-input NAND gate with A = B = 1**

Calculation of pull up resistor R_3 :

- The value of R_3 depends on the amount of current the gate is supposed to supply (source). That means R_3 depends on the value of **Fan-out.**
- Let the fan out be 5.

$$\therefore \text{ Fan out } = \frac{I_{OL}}{I_{IL}}, \quad \therefore 5 = \frac{I_{OL}}{I_{IL}}$$

- For a standard TTL, the value of I_{IL} is 1.6 mA.

$$\therefore I_{OL} = 5 \times I_{IL} = 5 \times 1.6 \text{ mA} = 8 \text{ mA}$$

- This current will flow through R_3. When Q_4 goes into saturation. Apply KVL to the collector circuit of Q_4 to write,

$$V_{CC} - I_{OL} R_3 - V_{CE4\,(sat)} = 0$$

$$\therefore R_3 = \frac{V_{CC} - V_{CE4\,(sat)}}{I_{OL}} = \frac{5 - 0.2}{8 \times 10^{-3}} = 600\ \Omega$$

11.7.1 Disadvantages of Open Collector Output :

1. The value of pull up resistance is high (few kΩ). Therefore if the load capacitance is large then the RC time constant (R_3C) becomes large. This slows down the switching speed of Q_4. Therefore the gates having an open collector output will be slow.
2. Second disadvantage is increased power dissipation. When Q_4 is ON, a large current flows through the pull up R_3. Hence power dissipation is increased. This problem is eliminated if we use totem-pole output arrangement.

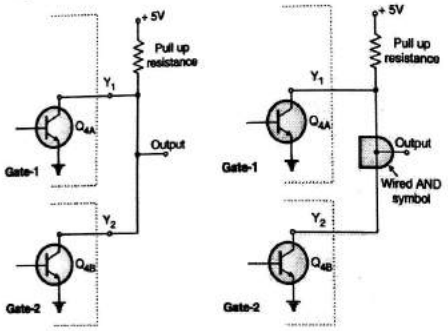
11.7.2 Advantage :

The main advantage of open collector output is that **Wired ANDing** becomes possible.

Scanned by CamScanner

11.7.3 Wired ANDing :

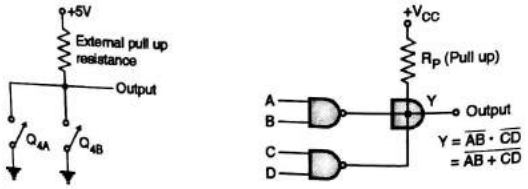
- Wired ANDing means connecting the outputs of gates together to obtain the AND function. It is possible to connect the outputs of two or more gates together as shown in Fig. 11.7.3(a).
- The wired ANDing is represented schematically as shown in Fig. 11.7.3(b).



(a) Wired ANDing of two open collector TTL gates **(b) Symbol of wired ANDing**

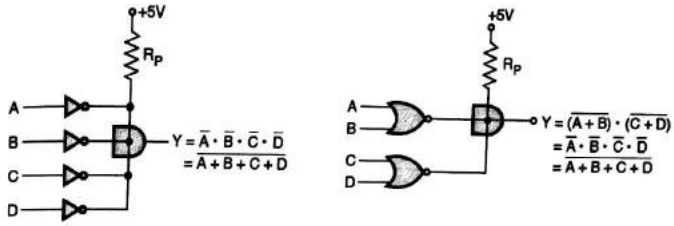
(C-1029) **Fig. 11.7.3 : Schematic representation of wired ANDing**

- Q_{4A} and Q_{4B} are the output transistors of Gate-A and Gate-B respectively. A common pull up resistance is used for all the output transistors.
- The transistors Q_{4A} and Q_{4B} are operated as switches. They operate in either saturation or cut off regions.
- Therefore the wired ANDing equivalent circuit is as shown in Fig. 11.7.4(a).
- The equivalent circuit of Fig. 11.7.4(a) indicates that the wired AND output will be "0" if any one or all the output transistor (s) are conducting. The output will be high (1) if and only if all the transistors are in their OFF state.
- This is the reason why such a connection is called as wired AND connection.
- Figs. 11.7.4(b), (c) and (d) shows that it is possible to wire AND the outputs of any gate such as NAND, NOR or inverter. The corresponding output Boolean equations are given alongwith the diagrams.



(a) Equivalent circuit of wired ANDing **(b) Wired ANDing of NAND gate outputs**

(C-1030) **Fig. 11.7.4 (Contd...)**



(c) Wired ANDing of inverter outputs **(d) Wired ANDing of the NOR outputs**

(C-1031) **Fig. 11.7.4**

11.7.4 Comparison of Totem-pole and Open Collector Outputs :

Table 11.7.1 gives the comparison of totem-pole and open collector outputs of TTL.

Table 11.7.1 : Comparison of totem-pole and open collector outputs

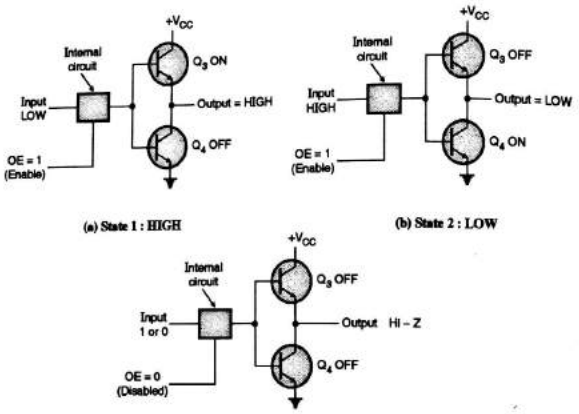
Sr. No.	Parameter	Totem-pole	Open collector
1.	Circuit components on the output side	Q_3 (pull up transistor), D and Q_4 (pull down transistor) are used.	Only the pull down transistor Q_4 is used.
2.	Wired ANDing	Cannot be done	Easily be done
3.	External pull up resistor	Not required	Required to be connected.
4.	Power Dissipation	Low due to presence of pull up transistor Q_3.	High due to current flowing through R_3.
5.	Speed	Operating speed is high.	Operating speed is low.

11.8 Tristate (Three state) TTL Devices :

- We have discussed the totem-pole and open collector output TTL configurations in previous sections. Now let us study the third type of output circuitry used in TTL as well as CMOS logic families. It is called as the **tristate configuration.**
- It is called as tristate because it allows three possible output states namely :

  1. HIGH 2. LOW and 3. High Impedance (Hi – Z) state.
- An additional input called Output Enable (OE) is introduced. Both the pull up and pull down transistors (Q_3 and Q_4) are being used.
- The output stage block diagram of a tri-state inverter is shown in Fig. 11.8.1(a). The enable signal OE decides the operation of transistors Q_3 and Q_4.

Scanned by CamScanner



(C-1032) **Fig. 11.8.1**

- When OE = 1, then the inverter operates as a normal inverter. If input is 0 then output will be 1 (HIGH) as shown in Fig. 11.8.1(a). The pull up transistor Q_3 will be ON.
- Similarly with OE = 1 and input high (1), the output will be low because the pull down transistor Q_4 will be ON as shown in Fig. 11.8.1(b).

High impedance state :

- If the enable input OE = 0 (zero), then irrespective of the status of input, both the transistors will remain off as shown in Fig. 11.8.1(c).
- This state of operation is called as high impedance (Hi – Z) state. In this state the output terminal is essentially open circuit i.e. not connected anywhere.

11.8.1 Advantages of Tristate :

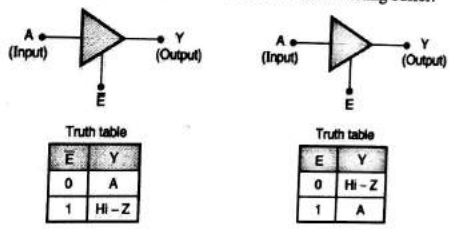
- The outputs of tristate ICs can be connected together i.e. they can make use of a common wire. But in doing so there is no reduction in switching speed (this problem is faced for open collector ICs).
- There is no reduction in speed because the tristate output when enabled, operates as a totem-pole TTL. So we get all the advantages of totem-pole output such as low output impedance and high speed.
- However note that when tristate outputs are connected together, we should enable only one of them at a time.
- Otherwise, two or more active outputs will fight to take control of the common line. This will result in very high currents which can damage the ICs and produce invalid output voltage levels.

11.8.2 Tristate Buffers :

University Questions MU : Dec. 02, May 03, Dec. 03, May 06

Q. 1 Write short note on tristate registers. **(Dec. 02, May 03, Dec. 03, May 06, 6 Marks)**

- A tristate buffer is a circuit which either connects or isolates its input from its output. In other words it controls the flow of signal from input to output.
- Tristate buffers are of two types, namely the non-inverting buffers and inverting buffer.
- The inverting tristate buffers will invert the input data while passing it to the output.
- Fig. 11.8.2 shows the symbols and truth tables for a non-inverting buffer.



Truth table

$\bar{E}$	Y
0	A
1	Hi – Z

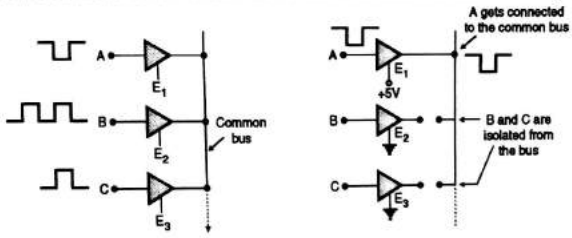
Truth table

E	Y
0	Hi – Z
1	A

(C-1033) **Fig. 11.8.2 : Tristate non-inverting buffers, symbols and truth tables**

11.8.3 Applications of Tristate Buffers (Bus Organization) :

- Tristate buffers are used in those applications in which several signals are simultaneously connected to a common bus.
- The basic idea about the common bus and the use of tristate buffers is illustrated in Fig. 11.8.3(a).



(a) A, B, C are connected to a common bus via tristate buffers **(b) A is getting transmitted on the bus**

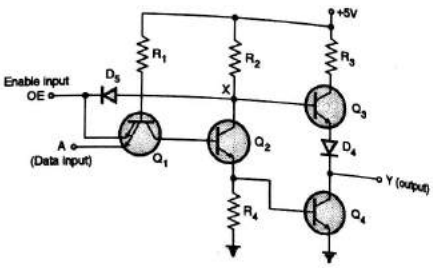
(C-1034) **Fig. 11.8.3**

- Three tristate buffers are used for connecting the inputs A, B and C to the common bus.
- As shown in Fig. 11.8.3(b), the enable signal E_1 is connected to +5 V. Therefore A gets connected to the bus.
- Since enable terminals E_2 and E_3 are connected to ground, the second and third buffer of Fig. 11.8.3(b) will go into the high impedance state. So B and C will be isolated from the common bus.

Scanned by CamScanner

11.8.4 A TRI-STATE Inverter :

Fig. 11.8.4 shows the circuit diagram of a Tri-State Logic (TSL) inverter. It has an enable input (OE) and a data input A and one output. The output stage is totempole type which has been discussed earlier.



(C-1035) **Fig. 11.8.4 : A TSL inverter**

Operation :

When OE = LOW :

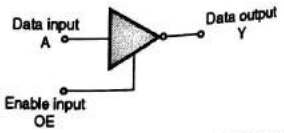
- When OE = 0, diode D_5 is forward biased and is turned ON. Therefore the voltage at point X is equal to 0.7 Volt, irrespective of the state of data input A.
- $V_X = 0.7$ V is not sufficient to turn ON Q_3, so it remains OFF.
- "OE" is also connected to one emitter of transistor Q_1. As OE = "0", transistor Q_1 is ON, hence Q_2 is OFF (as discussed for the NAND gate).
- As Q_2 is OFF, current through R_4 is zero. So Q_4 is in the OFF state.
- Thus with OE = LOW (0), both the output transistors Q_3 and Q_4 will be OFF and the inverter will be in the **High-Impedance** state.

When OE = HIGH :

- When OE = 1 (HIGH), the diode D_5 will be reverse biased. So it will be ineffective.
- Also the corresponding emitter base junction of Q_1 will be reverse biased. So the circuit of TSL inverter now acts as a TTL NAND gate with only one input A.
- If A = 0, then Q_1 is ON, so Q_2 is OFF, so $V_X = V_{CC}$, hence Q_3 turns ON and output Y = 1.
- If A = 1, then Q_1 is OFF, so Q_2 is ON, so V_X is low, but sufficient voltage appears across R_4, so Q_4 goes into saturation and Y = 0.
- Thus with OE = 1, the given circuit operates as a normal inverter.
- Table 11.8.1 summarizes the operation of the TSL inverter.

Table 11.8.1 : Operation of TSL inverter

Enable input	Data input	Output
OE = 1	A = 0	Y = 1
OE = 1	A = 1	Y = 0
OE = 0	A = 0 or 1	Hi-Impedance



(C-1036) **Fig. 11.8.5 : Logic symbol of TSL inverter**

Logic symbol :

The logic symbol of a TSL inverter is shown in Fig. 11.8.5.

11.9 Standard TTL Characteristics : MU : Dec. 02, May 03, Dec. 03, Dec. 05, May 09

University Questions

Q. 1 Define the various typical characteristics of a TTL gate. **(Dec. 02, May 03, Dec. 03, Dec. 05, May 09, 4 Marks)**

Q. 2 Write short notes on : TTL characteristics. **(Dec. 03, 6 Marks)**

The Texas instrument first introduced two TTL series namely 54 series and 74 series.

Supply voltage and temperature ranges :

- Table 11.9.1 gives the supply voltage and temperature ranges for two TTL families.

Table 11.9.1

TTL series	Supply voltage range	Temperature range
74 series	4.75 V to 5.25 V	0° C to 70° C
54 series	4.5 V to 5.5 V	– 55° C to 125° C

Voltage levels :

- Table 11.9.2 shows the input and output logic voltage levels for the TTL 74 series.

Table 11.9.2 : Voltage levels for TTL 74 series

Voltages	Minimum	Typical	Maximum
V_{OL}	–	0.2 V	0.4 V
V_{OH}	2.4 V	3.4 V	–
V_{IL}	–	–	0.8 V
V_{IH}	2.0 V	–	–

Noise margin :

- We have already defined the noise margins as,
  High level noise margin, $V_{NH} = V_{OH\,(min)} - V_{IH\,(min)}$
  and Low level noise margin, $V_{NL} = V_{IL\,(max)} - V_{OL\,(max)}$
- Substituting the values from Table 11.9.2, the noise margin for the TTL logic families can be calculated.

$$\therefore V_{NH} = 2.4 - 2 = 0.4 \text{ V}. \qquad V_{NL} = 0.8 - 0.4 = 0.4 \text{ V}.$$

- Thus noise margin for the TTL family is 0.4 V. This means that as long as the induced noise voltage is less than 0.4 V, the operation of the TTL ICs will be unaffected. The noise margins have been shown in section 11.3.3.

Power dissipation :

- The average power dissipation for the standard TTL 74 series is approximately 10 mW.
- It is dependent on the parameters such as tolerances, signal level etc.

Propagation delay :

- The propagation delay of a standard TTL gate is approximately 10 nanoseconds (1 nanosecond = 10^{-9} seconds).

Fan out :

A standard TTL gate is capable of driving at the most 10 other TTL gates simultaneously, maintaining its output voltage within the specified limits. Hence fan out of a TTL gate is 10.

All the important characteristics of TTL logic family are listed in Table 11.9.3.

Scanned by CamScanner

Table 11.9.3 : Important parameters of TTL logic family

Sr. No.	Parameter	Values
1.	Supply voltage	74 series : (4.75 to 5.25 V) 54 series : (4.5 to 5.5 V)
2.	Temperature range	74 series : 0 to 70° C 54 series : – 55 to 125° C
3.	Voltage levels	$V_{IL\,(max)} = 0.8$ V, $V_{OL\,(max)} = 0.4$ V $V_{IH\,(min)} = 2$ V, $V_{OH\,(min)} = 2.4$ V
4.	Noise margin	0.4 V
5.	Power dissipation	10 mW
6.	Propagation delay	10 nanosec.
7.	Fan out	10
8.	Figure of merit	100

11.9.1 Advantages of TTL :

1. TTL circuits are fast.
2. Low propagation delay.
3. Power dissipation is not dependent on frequency.
4. Compatible to all the other families.
5. Latch ups do not take place.
6. These are not susceptible to the damage due to static charges.
7. Higher current sourcing and sinking capabilities.

11.9.2 Disadvantages of TTL :

1. Large power dissipation.
2. Fan out is lower than that of CMOS.
3. Less component density.
4. Can operate only on +5 V power supply.
5. Poor noise immunity.

11.10 MOS - Logic Family :

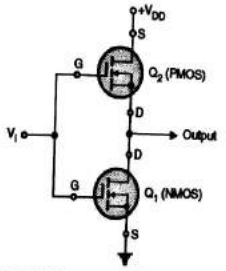
- The MOS - logic family uses MOSFET as the basic device the way TTL uses BJT.
- The MOSFETs are of two types namely the depletion MOSFETs and the enhancement type MOSFETs.
- In logic circuits, the enhancement MOSFETs are used. We use the E-MOSFET as a switch.

11.11 CMOS Logic :

- The 74C00 CMOS series is a group of CMOS circuits which are pin-for-pin and function for function compatible with the TTL 7400 devices.
- For example 74C32 is a quad 2-input OR gate in the CMOS family whereas 7402 is a quad 2 input TTL gate in the 7400 TTL family.

11.11.1 CMOS Inverter :

CMOS inverter with positive input and output voltages is shown in Fig. 11.11.1(a).



(a) CMOS inverter with positive voltages

V_I	Q_1	Q_2	Output
0 V (logic 0)	OFF	ON	V_{DD} (logic 1)
V_{DD} (logic 1)	ON	OFF	0 (logic 0)

(b) Summary of operation

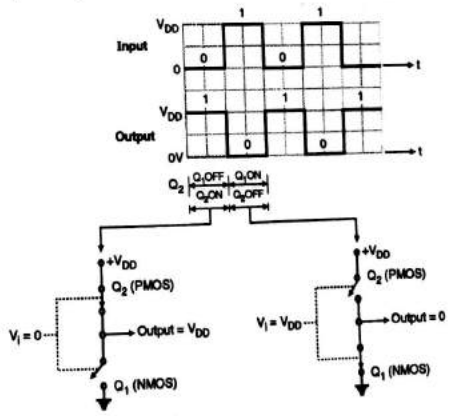
(C-1053) **Fig. 11.11.1**

Operation :

1. **With $V_I = 0$ Volt (logic 0) :**
   - With $V_I = 0$ Volt, the gate to source voltage V_{GS} of Q_1 (NMOS) will be 0 Volt, hence it will be OFF. But V_{GS} of Q_2 (PMOS) will be equal to $-V_{DD}$. So Q_2 will be ON.
   - Hence the output voltage will be equal to $+V_{DD}$ i.e. logic 1.
2. **With $V_I = V_{DD}$ (logic 1) :**
   - With $V_I = +V_{DD}$, the gate to source voltages of the two MOSFETs are as follows :
     Q_1 (NMOS) : $V_{GS1} = V_{DD}$, Q_2 (PMOS) : $V_{GS2} = 0$ Volt
   - Hence Q_1 will be turned ON and Q_2 will be OFF. So output gets connected to ground i.e. it will be "0". Thus the inversion will take place.
   - The operation of this inverter has been summarized in Fig. 11.11.1(b).

Waveforms and equivalent circuits :

The input output voltage waveforms for the inverter are shown in Fig. 11.11.2.

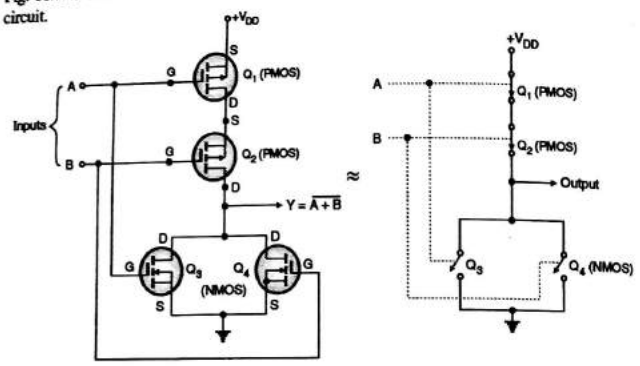


(C-1054) **Fig. 11.11.2 : Input output voltage waveforms and equivalent circuits for a CMOS inverter**

Scanned by CamScanner

11.11.2 CMOS NOR Gate :

- Fig. 11.11.3 shows the CMOS 2-input NOR gate. It is obtained by modifying the CMOS inverter circuit.



(a) CMOS NOR gate **(b) Equivalent circuit**

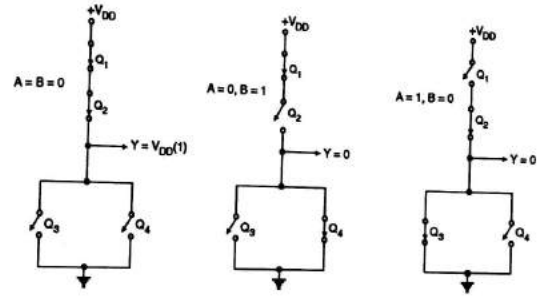
(C-1055) **Fig. 11.11.3**

- Q_1 and Q_2 are p-channel MOSFETs connected in series and Q_3 Q_4 are n-channel MOSFETs connected in parallel with each other.
- Input A is connected to the gates of Q_1 and Q_3 while input B is connected to the gates of Q_2 and Q_4.

Output $Y = V_{DD} = 1$ if Q_1 and Q_2 both ON and Q_3 Q_4 both OFF. And output $Y = 0$ if Q_1 or Q_2 or both are OFF and Q_3 or Q_4 both are ON.

Operation :

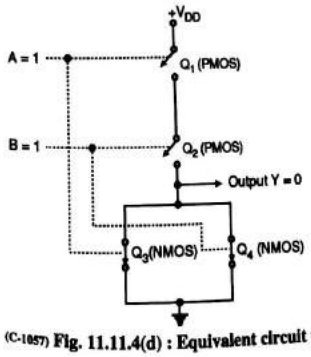
1. **With A = B = 0 :**
   - $V_{GS1} = -V_{DD}$, $V_{GS2} = -V_{DD}$, Hence Q_1 and Q_2 will be ON.
   - $V_{GS3} = 0$, $V_{GS4} = 0$, So Q_3 and Q_4 will be OFF.
   - The equivalent circuit for this mode is shown in Fig. 11.11.4(a) which shows that output $Y = V_{DD}$ (logical 1).
   - ∴ For A = 0, B = 0, output Y = 1.
2. **With A = 0, B = 1 :**
   - If A = 0 and B = 1, then Q_1 will remain ON, but Q_2 will turn OFF.
   - Q_3 will remain OFF but Q_4 will be ON.
   - The equivalent circuit for this mode is shown in Fig. 11.11.4(b) which shows that output Y = 0 Volt (logical 0).
   - Thus Y = 0 for A = 0 and B = 1.



(a) For A = B = 0 **(b) For A = 0, B = 1** **(c) For A = 1, B = 0**

(C-1056) **Fig. 11.11.4 : Equivalent circuits**

3. **With A = 1, B = 0 :**
   - If A = 1 and B = 0 then Q_1 will be OFF and Q_2 will turn ON.
   - Q_3 will be turned ON but Q_4 will turn OFF.
   - Equivalent circuit of this mode is shown in Fig. 11.11.4(c) which shows that Y = 0 Volt (logical 0).
   - Thus Y = 0 for A = 1, B = 0.
4. **For A = B = 1 :**
   - If A and B both are high (1), then Q_1 and Q_2 both will be OFF.
   - Q_3 and Q_4 both will be ON. Hence output Y = 0.
   - The equivalent circuit for this mode is shown in Fig. 11.11.4(d) and Table 11.11.1 summarizes the operation.



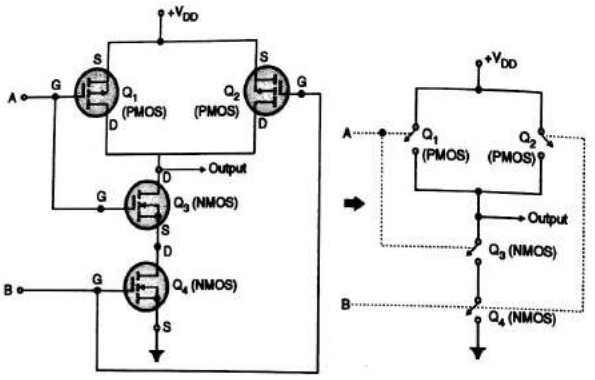
(C-1057) **Fig. 11.11.4(d) : Equivalent circuit for A = B = 1**

Table 11.11.1 : Summary of operation

Inputs		Transistors				Output
A	B	Q_1	Q_2	Q_3	Q_4	Y
0	0	ON	ON	OFF	OFF	V_{DD} (1)
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

Scanned by CamScanner

11.11.3 CMOS NAND Gate :



(a) 2-input CMOS NAND gate **(b) Equivalent circuit**

(C-1658) **Fig. 11.11.5**

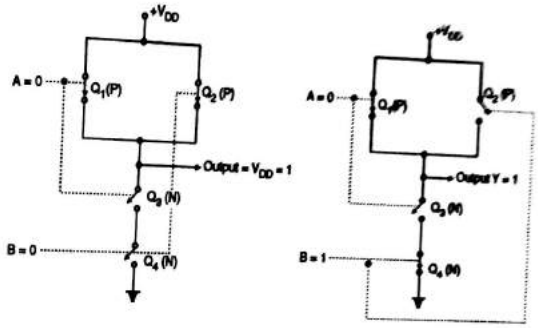
- The two input CMOS NAND gate is shown in Fig. 11.11.5(a) and its equivalent circuit by replacing each MOSFET by a switch is shown in Fig. 11.11.5(b).
- Q_1 and Q_2 are p-channel MOSFETs. They are connected in parallel with each other.
- Q_3 and Q_4 are n-channel MOSFETs. They are connected in series with each other.
- Input A is connected to gates of Q_1 and Q_3. So A controls the status of MOSFETs Q_1 and Q_3.
- Input B is connected to gates of Q_2 and Q_4. So B controls the status of MOSFETs Q_2 and Q_4.

Operation :

1. **A = B = 0 :**
   - With A = 0 and B = 0, both the PMOSFETs i.e. Q_1 and Q_2 will be ON. But both the N-MOSFETs i.e. Q_3 and Q_4 will be OFF.
   - As seen from the equivalent circuit of Fig. 11.11.6(a), the output $Y = +V_{DD}$ (logic 1).

   So Y = 1 if A = B = 0

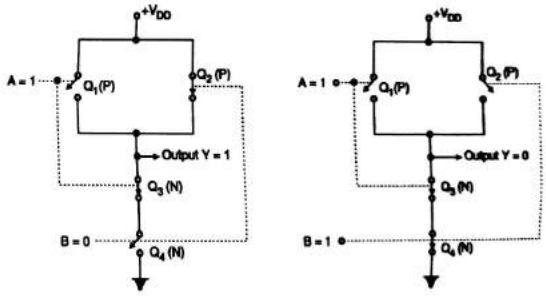
2. **With A = 0 and B = 1 :**
   - With A = 0 and B = 1, Q_1 will continue to be ON and Q_3 continues to be OFF. But Q_2 will now turn OFF and Q_4 will be turned ON.
   - The equivalent circuit of this mode is shown in Fig. 11.11.6(b) which shows that output $Y = +V_{DD}$ i.e. logic 1. So Y = 1 if A = 0 and B = 1.



(a) A = B = 0 **(b) A = 0, B = 1**

(C-1659) **Fig. 11.11.6 : Equivalent circuits**

3. **With A = 1 and B = 0 :**



(c) A = 1, B = 0 **(d) A = 1, B = 1**

(C-1660) **Fig. 11.11.6 : Equivalent circuit**

- With A = 1, Q_1 will be turned OFF and Q_3 will turn ON.
- And with B = 0, Q_2 will be turned ON and Q_4 will be turned OFF.
- As seen from the equivalent circuit of Fig. 11.11.6(c), the output $Y = +V_{DD}$ (logic 1).

  So Y = 1 if A = 1 and B = 0.

4. **With A = 1, B = 1 :**
   - With A = B = 1 both P-MOSFETs i.e. Q_1 and Q_2 will be OFF and both the N-MOSFETs i.e. Q_3 and Q_4 will be ON.
   - The equivalent circuit of this mode is shown in Fig. 11.11.6(d).
   - It shows that output Y = 0 (LOW).

   So Y = 0 if A = B = 1.
   - The operation of 2-input CMOS NAND gate is summarized in Table 11.11.2.

Scanned by CamScanner

Table 11.11.2 : Summary of operation of a CMOS NAND gate

Inputs		Transistors				Output Y
A	B	Q_1	Q_2	Q_3	Q_4	
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

11.11.4 CMOS Series :

The popular CMOS series are 4000/14000 series, 74C series, 74 HC/HCT (High speed CMOS), 74 AC/ACT (Advanced CMOS).

11.12 CMOS Characteristics :

Some of the important CMOS characteristics are as follows :

11.12.1 Power Supply Voltage :

- The 4000/14000 series and 74C series CMOS ICs are capable of operating over a wide range of power supply voltage (typically 3 V to 15 V).
- That means the CMOS ICs from these series are extremely versatile. They can operate on 3 V batteries as well as 5 V TTL compatible power supplies.
- These ICs can operate on higher power supply voltages, when higher noise margin is required.
- The other CMOS families such as 74 HC/HCT, 74 AC/ACT and 74 AHC/AHCT operate over a voltage range of 2 to 6 V.

11.12.2 Logic Voltage Levels :

- The logic voltage levels for different CMOS series have different values. Table 11.12.1 lists various logic voltage levels for different CMOS series.

Table 11.12.1 : Logic voltage levels (in volts) with $V_{DD} = V_{CC} = +5$ V

Parameter	CMOS							TTL			
	4000 B	74 HC	74 HCT	74 AC	74 ACT	74 AHC	74 AHCT	74	74LS	74AS	74 ALS
$V_{IH(min)}$	3.5	3.5	2.0	3.5	2.0	3.85	2.0	2.0	2.0	2.0	2.0
$V_{IL(max)}$	1.5	1.0	0.8	1.5	0.8	1.65	0.8	0.8	0.8	0.8	0.8
$V_{OH(min)}$	4.95	4.9	4.9	4.9	4.9	4.4	3.15	2.4	2.7	2.7	2.7
$V_{OL(max)}$	0.05	0.1	0.1	0.1	0.1	0.44	0.1	0.4	0.5	0.5	0.4
V_{NH}	1.45	1.4	2.9	1.4	2.9	0.55	1.15	0.4	0.7	0.7	0.7
V_{NL}	1.45	0.9	0.7	1.4	0.7	1.21	0.7	0.4	0.3	0.3	0.4

Important points from Table 11.12.1 :

1. All the voltage levels shown in Table 11.12.1 correspond to a supply voltage of + 5 Volts.
2. Note that $V_{OH} \approx 5$ V and $V_{OL} \approx 0$ V.

11.12.3 Noise Margins :

- Table 11.12.1 contains the high level and low level noise margins V_{NH} and V_{NL} for each CMOS and TTL series.
- These are calculated as follows :

$$V_{NH} = V_{OH(min)} - V_{IH(min)}$$
$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

- It can be observed that the CMOS devices will have higher noise margins as compared to those of TTL. So CMOS ICs should be preferred to TTL for operation in the noisy environment.
- The noise margins in CMOS will increase further if the supply voltages are increased further.

11.12.4 Power Dissipation :

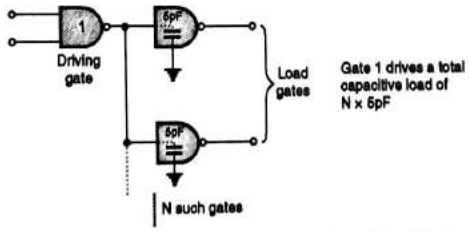
- The power dissipation of CMOS devices is extremely low when they are in the static (stable) state.
- Typically the dc power dissipation is 2.5 nW per gate when $V_{DD} = 5$ V and 10 nW for $V_{DD} = 10$ V. This is too small as compared to TTL gates ($P_D = 10$ mW).
- Hence CMOS devices are preferred for the battery operated systems.
- But power dissipation is low under the dc operating conditions and at low frequencies. But power dissipation increases as the circuit switching frequency is increased.
- The relation between power dissipation and frequency is demonstrated in Table 11.12.2.

Table 11.12.2 : Relation between P_D and frequency

Frequency	0 (dc)	100 kHz	1 MHz
Power dissipation P_D	10 nW	0.1 mW	1 mW

11.12.5 Fan Out :

- The input resistance of CMOS devices is very high (10^{12} Ω). So their input current is very very small, almost zero. Therefore one CMOS gate can drive a large number of other CMOS gates. Hence fan out of CMOS devices will be large as compared to fan out of TTL.
- Typically the fan out is restricted to 50 for operation below 1 MHz.
- Why is fan out restricted to only 50 ? This is because the input capacitance of each CMOS gate is about 5 pF. These capacitors act as load on the driving gate as shown in Fig. 11.12.1. The charging current for these capacitor has to be supplied by the driving gate. This current should not be too large. This will limit the fan-out to 50.



(C-1061) **Fig. 11.12.1 : Fan out and switching speed are dependent on the input capacitance of load gates**

Scanned by CamScanner

11.12.6 Switching Speed :

- The output resistance of CMOS is low in both the states (0 or 1) of output. So eventhough it has to drive large capacitive loads, the switching speed can still be faster than the NMOS or PMOS devices.
- The NAND gate of 4000 series has following values of propagation delays,

  Average t_{pd} = 50 nS ..... at V_{DD} = 5 V

  Average t_{pd} = 25 nS ..... at V_{DD} = 10 V
- The average time delay for various CMOS ICs are given in Table 11.12.3.

Table 11.12.3 : Switching speeds for various CMOS ICs at V_{DD} = 5 V

Family	4000	74 HC/HCT	74 AC/ACT	74 AHC
Propagation Delay t_{pd}	50 nS	8 nS	4.7 nS	4.3 nS

11.12.7 Unconnected Inputs :

- Note that the CMOS inputs should never be left floating.
- All the CMOS inputs should be either connected to 0 V (ground) or V_{DD}, or to another inputs.
- This is necessary to avoid permanent damage of CMOS ICs.
- Sometimes there are some unused gates on a chip. The inputs of such gates also should be connected to ground or + V_{DD}.
- The CMOS ICs are damaged due to induced voltages at the floating inputs due to noise or static charges. Such voltages can bias the P-MOS or N-MOS in their conduction state that may cause overheating and damage.

11.12.8 Static Sensitivity :

- The MOS ICs are susceptible to the damages due to static electricity. Such static electricity (voltages upto about 30 kV) gets generated by our simple day to day actions.
- For example, if a person walks on a carpet, then he will generate a voltage of 30 kV.
- After that if he touches the gate of a MOSFET, then the oxide layer will breakdown, due to large voltage and the device will be permanently damaged.
- Such a damage is called as damage due to **Electro-Static Discharge (ESD),** and we have to use a resistor diode protection network for protection.

11.12.9 Latch Ups :

- Due to the peculiar construction of the MOSFETs, there exist parasitic (unwanted) PNP and NPN transistors in the MOSFET structures.
- Under some operating conditions, these parasitic transistor will turn ON and will latch up i.e. remain ON permanently. This may force a large current through the CMOS device and damage it permanently.
- Latch ups can take place when the maximum voltage rating of the device is exceeded. Latch ups can also take place due to high voltage spikes or ringing at the input.
- To avoid such latch ups, we have to use well designed regulated power supply with minimum spikes on the V_{DD} line. The power supply with current limit will help to minimize the device current if latch up takes place.
- Manufacturers provide an internal protection circuit for avoiding the latch ups.

11.13 Advantages of CMOS :

1. Low power dissipation.
2. High fan out (typically 50).

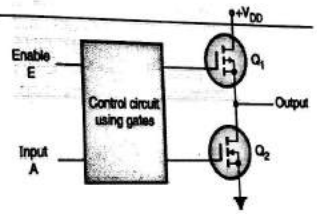
3. High noise margin for higher values of V_{DD}.
4. Capable of working over a wide range of supply voltage.
5. Switching speeds comparable to those of TTL.
6. High packaging density since (more devices can be accommodated in the same space) MOS devices need less space.

11.14 Disadvantages of CMOS :

1. Propagation delays longer than those of TTL (25 to 100 nS).
2. Slower than TTL.
3. CMOS ICs can get damaged due to static charge.
4. Latch ups can take place which can damage the device.
5. Need protection circuitry.

11.15 Tristate Logic Output :

- Fig. 11.15.1 shows an inverting driver alongwith logic gates.
- When both the MOSFETs are in the OFF state, the output is in its high impedance state.
- When the enable input E = 1, the inverter operates in a normal way. Either Q_1 is ON or Q_2 is ON, depending on state of the input A.



(C-1065) **Fig. 11.15.1 : A tristate CMOS inverter**

- But if E = 0, then neither Q_1 nor Q_2 is ON. Then irrespective of input A, both the MOSFETs will be OFF and the inverter will go into the high impedance state.

11.16 Interfacing :

- The meaning of word interfacing is to connect one or more outputs of one system to the one or more inputs of another system or circuits which have different electrical characteristics.
- If the electrical characteristics of the two circuits or systems are different then we cannot have a direct connection between them.
- So an "interface" circuit is required to be inserted between the "driver" circuit and the "load" circuit as shown in Fig. 11.16.1.
- The task of the interface circuit is to accept the output of the driver circuit and "condition" it so that it becomes compatible with the load circuit.

(C-1067) **Fig. 11.16.1 : Interfacing between two different systems**

- ICs from the same family are designed in such a way that they can be connected together without any special considerations.
- But when we connect the output of one type of IC to the input of another type of IC, we have to be concerned about the differences related to the voltage and current levels of both the ICs.

11.16.1 TTL to CMOS Interfacing :

MU : May 05, Dec. 05, Dec. 06

University Questions

Q.1 Explain the interfacing of CMOS and TTL gates. **(May 05, Dec. 05, Dec. 06, 10 Marks)**

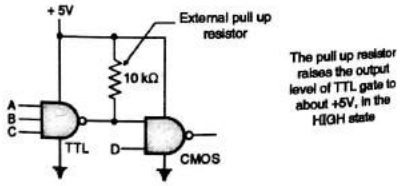
Scanned by CamScanner

- When we interface different types of ICs, it is necessary to check whether the driving IC is capable of meeting the current and voltage requirements of the load IC or not.
- For example Table 11.16.1 reveals that the output current capability of TTL ICs is much higher than the input current values of CMOS ICs. Therefore there is no problem for a TTL IC to drive CMOS as far as current is concerned.
- But there is a problem when we compare the voltage levels of TTL and CMOS. Because $V_{OH\,(min)}$ of a TTL series is very low as compared with $V_{IH\,(min)}$ required for the CMOS series like 400B, 74 HC or 74 AC.
- In such situations, something has to done to increase the level of TTL output voltage to an acceptable voltage level for CMOS.

Table 11.16.1 : Input output currents for standard devices with a supply voltage of 5 V

Parameter	CMOS		TTL		
	4000 B	74 HC/HCT	74	74 LS	74 AS
$I_{IH\,(max)}$	1 μA	1 μA	40 μA	20 μA	20 μA
$I_{IL\,(max)}$	1 μA	1 μA	1.6 mA	0.4 mA	0.5 mA
$I_{OH\,(max)}$	0.4 mA	4 mA	0.4 mA	0.4 mA	2 mA
$I_{OL\,(max)}$	0.4 mA	4 mA	16 mA	8 mA	20 mA

- The situation in which a TTL IC is driving CMOS is shown in Fig. 11.16.2.
- Note the presence of pull up resistor at the output of the TTL gate. Due to this resistor the TTL output will rise to approximately +5 V in its HIGH state.
- This will provide the sufficient voltage level at the input of the CMOS gate.
- Such a pull up resistance is not required if the CMOS gate belongs to 74 HCT or 74 ACT family, because these families can accept the TTL outputs directly. In other words they are TTL compatible CMOS families.

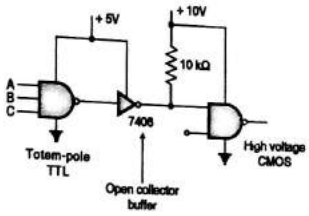


(C-1068) **Fig. 11.16.2 : TTL driving CMOS**

11.16.2 TTL Driving High Voltage CMOS :

- But if the CMOS IC is operated with a supply voltage V_{DD} greater than 5 V then the interfacing becomes a little bit difficult.
- For example with V_{DD} = 10 V, then CMOS input requires $V_{IH\,(min)}$ = 7 V.
- The output stages of many TTL devices cannot operate at a voltage higher than + 5 V. Therefore the technique of using the pull up resistance can no more be employed.
- The interfacing of TTL to a high voltage CMOS is shown in Fig. 11.16.3.
- When it is not possible to pull up the totem-pole output to + V_{DD}, an open collector buffer is used as interface between TTL totem-pole and high voltage CMOS gate.

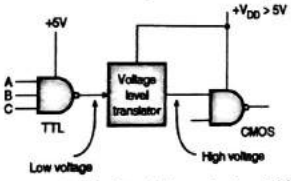
- IC 7406 is an inverting CMOS buffer. It has an output voltage rating of 30 V. So we can connect it as an interfacing buffer along with a pull up resistance of 10 kΩ is shown in Fig. 11.16.3.
- Note that the pull up resistance has been connected to the 10 Volts supply. Therefore the buffer output and therefore the CMOS input could reach 10 Volts level in its HIGH state. Thus the TTL can be successfully interfaced with a high voltage CMOS.



(C-1069) **Fig. 11.16.3 : TTL to high voltage CMOS interfacing, using open collector buffer**

11.16.3 Interfacing using Level-Shifter (TTL to High Voltage CMOS) :

- Another way to interface TTL with high voltage CMOS is to use a voltage level translator (a level shifter) for interfacing as shown in Fig. 11.16.4.
- The voltage translator circuit accepts a low voltage from TTL and translates it into a high voltage output for CMOS IC inputs.
- Note that the supply voltage to the level shifter is the supply voltage of the CMOS IC.
- The example of level-shifter is IC 4504 B.



(C-1070) **Fig. 11.16.4 : Interfacing using level-shifter**

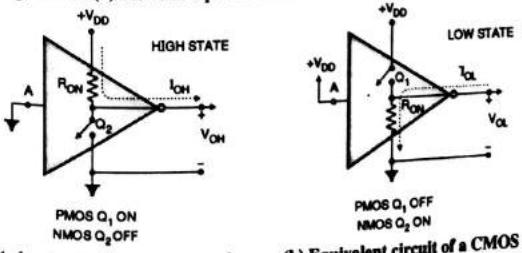
11.16.4 CMOS to TTL Interface :

MU : May 05, Dec. 05, Dec. 06

University Questions

Q. 1 Explain the interfacing of CMOS and TTL gates. **(May 05, Dec. 05, Dec. 06, 10 Marks)**

- Before we consider the problem of CMOS interfacing with TTL, let us review the output characteristics of CMOS in two different states.
- Fig. 11.16.5(a) shows the equivalent output circuit of a CMOS inverter in the HIGH state output, whereas Fig. 11.16.5(b) shows its equivalent in the LOW state output.



(a) **Equivalent circuit of a CMOS inverter for HIGH state** (b) **Equivalent circuit of a CMOS inverter for low state**

(C-1071) **Fig. 11.16.5**

Scanned by CamScanner

quivalent circuit in HIGH state :

In the HIGH state of output, the P-channel MOSFET Q_1 will be ON, so it has been replaced by the ON state resistance R_{ON}. Whereas the N-channel MOSFET Q_2 will be OFF. So it has been replaced by an open switch as shown in Fig. 11.16.5(a).

Therefore the CMOS output is equivalent to a V_{DD} source with a source resistance of R_{ON}.

quivalent circuit in LOW state :

The equivalent circuit of a CMOS inverter in its low state of output is shown in Fig. 11.16.5(b). In this state, the P-MOSFET Q_1 is OFF and represented by an open switch.

Whereas the N-channel MOSFET is ON. So it has been replaced by resistance R_{ON}. So the CMOS inverter now acts as a low resistance connected to ground and sinks current.

Table 11.16.2 presents the various voltage and current levels for the standard CMOS (4000B) and TTL (74) series for comparison.

Table 11.16.2

Sr. No.	Parameters for driving gate (CMOS) of 4000 B series	Parameters for the load gate (TTL) of 74 series
1.	$V_{OH\,(min)} = 4.95$ V	$V_{IH\,(min)} = 2$ V
2.	$V_{OL\,(max)} = 0.05$ V	$V_{IL\,(max)} = 0.8$ V
3.	$I_{OH\,(max)} = 0.4$ mA	$I_{IH\,(max)} = 40\ \mu A$
4.	$I_{OL\,(max)} = 0.4$ mA	$I_{IL\,(max)} = 1.6$ mA

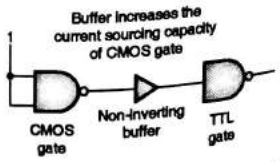
For low state output (2, 4) — For high state output (1, 3)

11.16.5 CMOS Driving TTL in the HIGH State :

- Now let us understand what happens when a CMOS IC drives TTL. Let the CMOS output voltage be high (1). So consider the first and third parameters in Table 11.16.2 i.e. V_{OH} and I_{OH} for CMOS and V_{IH} and I_{IH} for TTL.
- Looking at these parameters we understand that the CMOS output can supply sufficient voltage as well as current to the TTL inputs. So there is no need of using any special interface for high state. We can directly connect CMOS output to TTL input.

11.16.6 CMOS Driving TTL in the LOW State :

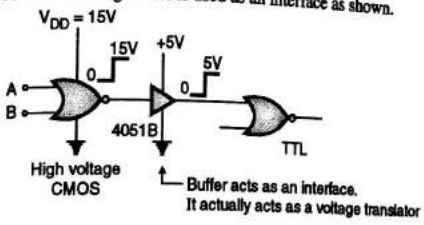
- Now let the CMOS output be low (0). So consider the second and fourth parameters in Table 11.16.2 i.e. V_{OL}, I_{OL} for CMOS gate and V_{IL} and I_{IL} for TTL gate.
- Looking at these parameters we understand that the low state output voltage V_{OL} of CMOS can satisfy the V_{IL} requirement of TTL.
- But the low level output current of a CMOS gate (I_{OL}) is not sufficient to drive even a single TTL gate with $I_{IL\,(max)} = 1.6$ mA.
- Therefore to improve the current sourcing capability of CMOS we should use a non-inverting buffer between the CMOS and TTL gates as shown in Fig. 11.16.6.



(C-1072) **Fig. 11.16.6 : CMOS driving TTL in low state**

11.16.7 High Voltage CMOS Driving TTL :

- The high voltage CMOS gates operate at high supply voltages (typically 15 V). So their output voltage in the HIGH state will be $V_{OH\,(max)} \approx 15$ V.
- Almost all the TTL ICs have a maximum input voltage $V_{IH\,(max)} = 7$ V.
- Therefore an interface as shown in Fig. 11.16.7 will be required between high voltage CMOS gates and TTL. A level shifting buffer is used as an interface as shown.

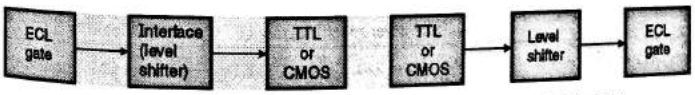


(C-1073) **Fig. 11.16.7 : High voltage CMOS driving TTL**

- The buffer 4051B is connected to a + 5 V supply. It is a non-inverting buffer. So it converts a 15 V HIGH output of the CMOS gate into a 5 V input for the TTL gate.
- Thus the 4051B buffer acts like a level translator.

11.16.8 Interfacing CMOS with ECL and TTL :

- The logic voltage levels in ECL are completely different from those of TTL and CMOS.
- Therefore to interface ECL gates with the other logic families such as CMOS and TTL we need to use the interface in the form of a voltage translator i.e. level shifter.
- Voltage translators are actually the voltage level shifters. They are available in the ECL family.
- The TTL/CMOS to ECL and ECL to TTL/CMOS interfacing is shown in Fig. 11.16.8.



(a) ECL to TTL/ CMOS (b) TTL/CMOS to ECL

(C-1074) **Fig. 11.16.8 : Interfacing**

11.17 Comparison of CMOS and TTL :

MU : May 08, May 11, May 12, Dec. 12, Dec. 15, May 16, Dec. 16

University Questions

Q. 1 Compare TTL and CMOS logic families. **(May 08, May 11, May 12, Dec. 12, 10 Marks)**

Q. 2 Compare TTL and CMOS with respect to speed, power dissipation, fan-in and fan-out. **(Dec. 15, 2 Marks)**

Q. 3 Compare TTL and CMOS logic. **(May 16, 5 Marks)**

Q. 4 Compare TTL and CMOS with respect to speed, power dissipation, fan-in and fan-out and also define these terms. **(Dec. 16, 4 Marks)**

Scanned by CamScanner

Table 11.17.1 : Comparison of CMOS and TTL

Sr. No.	Parameter	CMOS	TTL
1.	Device used	N-channel MOSFET and P-channel MOSFET	Bipolar junction transistor
2.	$V_{IH\,(min)}$	3.5 V (V_{DD} = 5 V)	2 V
3.	$V_{IL\,(max)}$	1.5 V	0.8 V
4.	$V_{OH\,(min)}$	4.95 V	2.7 V
5.	$V_{OL\,(max)}$	0.05 V	0.4 V
6.	High level noise margin	V_{NH} = 1.45 V	0.4 V
7.	Low level noise margin	V_{NL} = 1.45 V	0.4 V
8.	Noise immunity	Better than TTL	Less than CMOS
9.	Propagation delay	105 nS (Metal gate CMOS)	10 nS. (Standard TTL)
10.	Switching speed	Less than TTL.	Faster than CMOS
11.	Power dissipation per gate.	P_D = 0.1 mW. Hence used for battery backup applications	10 mW
12.	Speed power product.	10.5 pJ	100 pJ
13.	Dependence of P_D on frequency	P_D increases with increase in frequency.	P_D does not depend on frequency.
14.	Fan out	Typically 50.	10
15.	Unconnected inputs	Unused inputs should be returned to GND or V_{DD}. They should never be left floating.	Inputs can remain floating. The floating inputs are treated as logic 1s.
16.	Component density	More than TTL since MOSFETs need smaller space while fabricating an IC.	Less than CMOS since BJT needs more space.
17.	Operating areas	MOSFETs are operated as switches. i.e. in the ohmic region or cut off region.	Transistors are operated in saturation or cut off regions.
18.	Power supply voltage	Flexible from 3 V to 15 V.	Fixed equal to 5 V.

11.17.1 Comparison of CMOS, TTL and ECL :

MU : May 06, Dec. 09, May 10, Dec. 11, Dec. 12, Dec. 14

University Questions

Q. 1 Compare the different logic families with the following parameters; Fan in, Fan out, Noise Margin, Speed, Power Dissipation, etc. **(May 06, 6 Marks)**

Q. 2 Compare TTL, CMOS and ECL families with respect to basic gate, voltage levels, fan-in, fan-out, propagation delay, power dissipation and noise margin. **(Dec. 09, 10 Marks)**

Q. 3 Compare TTL, CMOS and ECL families with respect to gate, voltage level, fan-in, fan-out, propagation delay, power dissipation and noise margin. **(May 10, 10 Marks)**

Q. 4 Compare the different logic families with respect to the following parameters : Fan in, Fan out, Noise margin, speed and power dissipation. **(Dec. 11, 10 Marks)**

Q. 5 Compare the different logic families with respect to the following parameters : fan in, fan out, noise margin, speed and power dissipation. **(Dec. 12, 10 Marks)**

Q. 6 Compare the performance of TTL, CMOS and ECL logic. **(Dec. 14, 8 Marks)**

Sr. No.	Parameter	TTL	CMOS	ECL
1.	Components used.	Transistors diodes and resistors	MOSFETs	Resistors and transistors
2.	Fan out.	Moderate	Highest	High
3.	Propagation delay.	10 nS	70 nS	2 nS
4.	Noise margin.	Moderate	High	Low
5.	Power dissipation per gate	10 mW	0.1 mW	40-50 mW
6.	Speed power product	100 pJ	0.7 pJ	40-100 pJ
7.	Circuit complexity	Complex	Moderately complex	Complex
8.	Basic gate	NAND	NAND/NOR	OR/NOR
9.	Applications	Lab and demonstration equipment	Portable equipments as they consume less power	High speed switching applications due to high speed.

Review Questions

Short Answer Questions :

Q. 1 Which are the different logic families ? Write their characteristics.

Q. 2 What is passive pull-up load and active pull-up load ?

Q. 3 Give the advantages of active pull-up over passive pull-up.

Q. 4 Explain the use of multi-emitter inputs.

Scanned by CamScanner

Q. 5 Define the following terms regarding a logic family :

1. Noise margin 2. Propagation delay

Q. 6 Compare the performance of TTL, CMOS and ECL logic.

Q. 7 Which are the different logic families ? Write their characteristics.

Q. 8 What is passive pull-up load and active pull-up load ?

Q. 9 Give the advantages of active pull-up over passive pull-up.

Q. 10 Explain the use of multi-emitter inputs.

Q. 11 Define the following terms regarding a logic family :

1. Noise margin 2. Propagation delay

Q. 12 Compare the performance of TTL, CMOS and ECL logic.

Q. 13 Explain the features of complementary symmetry logic (CMOS).

Q. 14 State the application of CMOS logic inverter.

Q. 15 What are the precautions taken while using CMOS ICs ?

Q. 16 Mention the advantages and disadvantages of TTL, CMOS and ECL IC families.

Q. 17 List any four characteristics of logic gates.

Q. 18 Compare between TTL and CMOS logic.

Q. 19 Classify the IC according to their scale of integration.

Q. 20 Explain the terms SSI, MSI, LSI, VLSI.

Q. 21 Explain what is meant by TTL ?

Q. 22 Draw the circuit diagram of two input TTL NAND gate and explain the function of each component in it.

Long Answer Questions :

Q. 1 Explain with diagram the working of High-Threshold Logic (HTL). State the advantage of HTL over DTL ?

Q. 2 Explain briefly the operation of TTL NAND gate in tri-state output configurations.

Q. 3 Give the important characteristics of logic families and explain their importance.

Q. 4 Explain the ECL circuit.

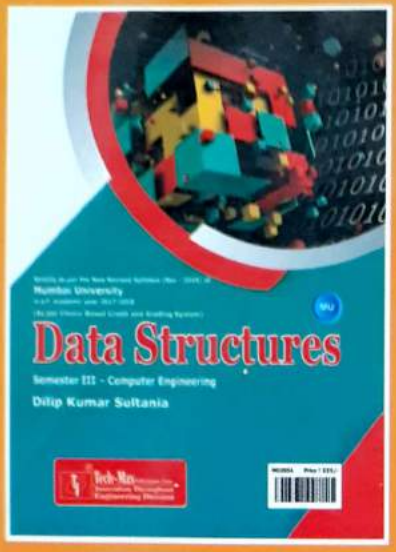
Q. 5 Explain 7400 TTL series. Sketch pin-outs of different ICs of 7400 series and state their functions.

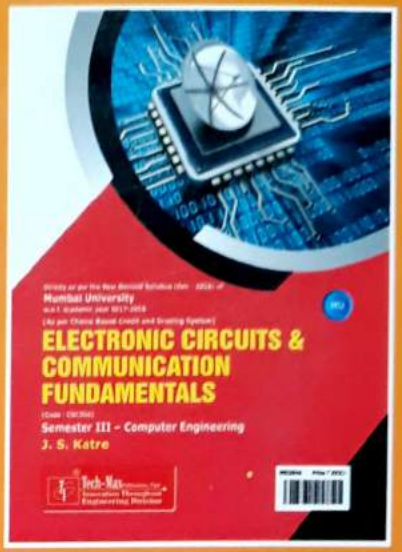
Q. 6 State specifications of standard TTL family.

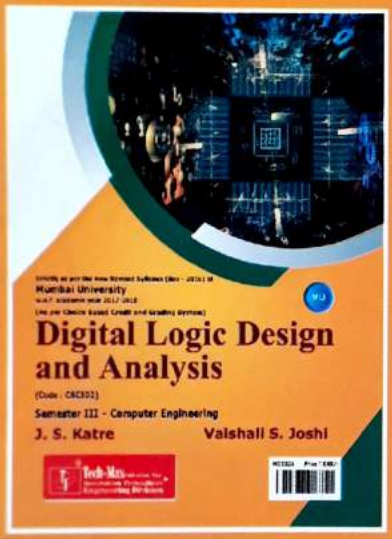
□□□

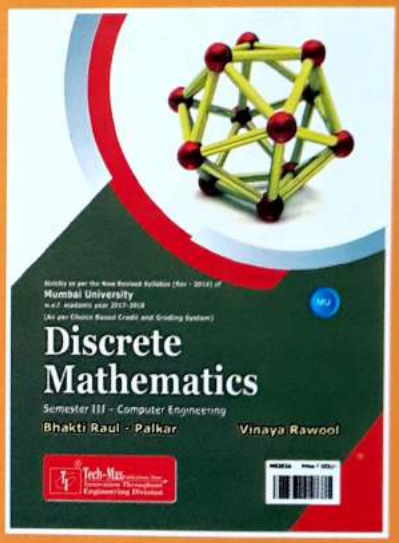
Scanned by CamScanner

Semester III - Computer Engineering









Tech-Max Publications

Office address : B/5, First Floor, Maniratna Complex, Aranyeshwar Corner, Pune - 411009.Maharashtra State, India.
Tel. : 91-20-24217965, 91-20-24225065
E-mail : info@techmaxbooks.com

ISBN : 978-93-86742-07-0

Price ₹ 455/-

http://www.facebook.com/techmax.publications
http://www.twitter.com/techmaxbooks

Books are available at :
KRISHNA BOOKS COLLECTIONS
Ground Floor, Krishna Niwas Building, Behind BEST Niwas Building, Near Napoo Hall, Chandavarkar Road, Matunga East, Mumbai 400019.
Tel No. : 022-24109080 / 022-24102571
Mobile No. : 7498018909 9833082745 / 9833082761

SCAN TO VISIT
Tech-Max Publications
www.techmaxbooks.com

Scanned by CamScanner