

SUBJECT CODE : 210245

As per Revised Syllabus of
SAVITRIBAI PHULE PUNE UNIVERSITY
Choice Based Credit System (CBCS)
S.E. (Comp.) Semester - I

DIGITAL ELECTRONICS AND LOGIC DESIGN

(For IN SEM Exam - 30 Marks)

Atul P. Godse

M.S. Software Systems (BITS Pilani)

B.E. Industrial Electronics

Formerly Lecturer in Department of Electronics Engg.
Vishwakarma Institute of Technology
Pune

Dr. Mrs. Deepali A. Godse

M.E., Ph.D. (Computer Engg.)

Head of Information Technology Department,
Bharati Vidyapeeth's College of Engineering for Women,
Pune

Dr. Vinod Vijaykumar Kimbahune

Ph.D in Computer Engineering

Associate Professor

Smt. Kashibai Navale College of Engineering.
Vadgaon(bk)

Dr. Sunil M. Sangve

Ph.D in Computer Science and Engineering

Professor and Head, Computer Engineering

Zeal College of Engineering and Research,
Narhe, Pune



DIGITAL ELECTRONICS AND LOGIC DESIGN

(For IN SEM Exam - 30 Marks)

Subject Code : 210245

S.E. (Computer) Semester - I

First Edition : August 2020

© Copyright with A. P. Godse, Dr. D. A. Godse

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-90041-79-4



9 789390 041794

SPPU 19

PREFACE

The importance of **Digital Electronics and Logic Design** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Digital Electronics and Logic Design**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors

A. P. Godse

Dr. D. A. Godse

Dr. V. V. Kimbahune

Dr. S. M. Sange

Dedicated to God

SYLLABUS

Digital Electronics and Logic Design (210245)

Credit Scheme	Examination Scheme and Marks
03	Mid_Semester (TH) : 30 Marks

Unit I Minimization Technique

Logic Design Minimization Technique : Minimization of Boolean function using K-map (up to 4 variables) and Quine Mc-Clusky Method, Representation of signed number-sign magnitude representation ,1's complement and 2's complement form, Sum of product and Product of sum form, Minimization of SOP and POS using K-map. (**Chapters - 1, 2**)

Unit II Combinational Logic Design

Code converter : BCD, Excess - 3, Gray code, Binary Code. Half - Adder, Full Adder, Half Subtractor, Full Subtractor, Binary Adder (IC 7483), BCD adder, Look ahead carry generator, Multiplexers (MUX) : MUX (IC 74153, 74151), Cascading multiplexers, Demultiplexers (DEMUX) - Decoder (IC 74138, IC 74154), Implementation of SOP and POS using MUX, DMUX, Comparators (2 bit), Parity generators and Checker.

(Chapter - 3)

TABLE OF CONTENTS

Unit - I

Chapter - 1 Review of Number Systems	(1 - 1) to (1 - 24)
1.1 Number Systems	1 - 2
1.2 Representation of Numbers of Different Radix	1 - 2
1.2.1 Decimal Number System	1 - 2
1.2.2 Binary Number System.....	1 - 3
1.2.3 Octal Number System.....	1 - 3
1.2.4 Hexadecimal Number System	1 - 4
1.2.5 Format of a Binary Number	1 - 4
1.2.6 Counting in Radix (Base) r	1 - 5
1.3 Conversion of Numbers from One Radix to Another Radix.....	1 - 5
1.3.1 Binary to Octal Conversion.....	1 - 5
1.3.2 Octal to Binary Conversion.....	1 - 6
1.3.3 Binary to Hexadecimal Conversion	1 - 6
1.3.4 Hexadecimal to Binary Conversion	1 - 7
1.3.5 Octal to Hexadecimal Conversion	1 - 7
1.3.6 Hexadecimal to Octal Conversion	1 - 8
1.3.7 Converting Any Radix to Decimal	1 - 9
1.3.8 Conversion of Decimal Number to Any Radix Number	1 - 9
1.4 Representation of Signed Numbers.....	1 - 13
1.4.1 Sign-Magnitude Representation	1 - 13
1.4.2 1's Complement Representation.....	1 - 14
1.4.3 2's Complement Representation.....	1 - 14
1.4.4 Binary Subtraction using 1's Complement Method.....	1 - 15
1.4.5 Binary Subtraction using 2's Complement Method.....	1 - 16
1.5 BCD (Binary Coded Decimal) Codes.....	1 - 18
1.5.1 BCD Addition	1 - 18
1.5.2 Excess-3 Code	1 - 20

1.5.3 Gray Code	1 - 21
1.5.3.1 Gray to Binary Conversion	1 - 22
1.5.3.2 Binary to Gray Conversion	1 - 23

Unit - II

Chapter - 2 Logic Design Minimization Techniques (2 - 1) to (2 - 70)

2.1 Boolean Expressions	2 - 2
2.1.1 Sum of Product Form.....	2 - 3
2.1.2 Product of Sum Form.....	2 - 3
2.1.3 Canonical SOP and Canonical POS Forms	2 - 4
2.1.4 Canonical Expressions in Canonical SOP or POS Form	2 - 4
2.1.4.1 Steps to Convert SOP to Canonical SOP Form	2 - 4
2.1.4.2 Steps to Convert POS to Canonical POS Form	2 - 6
2.1.5 M Notations : Minterms and Maxterms	2 - 8
2.1.6 Complements of Canonical Forms.....	2 - 10
2.2 Karnaugh (K-Map) Minimization.....	2 - 11
2.2.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps	2 - 11
2.2.2 Plotting a Karnaugh Map	2 - 14
2.2.2.1 Representation of Truth Table on Karnaugh Map	2 - 14
2.2.2.2 Representing Standard SOP on K-Map	2 - 15
2.2.2.3 Representing Standard POS on K-Map	2 - 16
2.2.3 Grouping Cells for Simplification	2 - 17
2.2.3.1 Grouping Two Adjacent Ones (Pair)	2 - 18
2.2.3.2 Grouping Four Adjacent Ones (Quad)	2 - 20
2.2.3.3 Grouping Eight Adjacent Ones (Octet)	2 - 21
2.2.4 Illegal Grouping.....	2 - 22
2.3 Simplification of SOP Expression	2 - 22
2.3.1 Essential Prime Implicants	2 - 29
2.3.2 Incompletely Specified Functions (Don't Care Terms)	2 - 30
2.3.2.1 Describing Incomplete Boolean Function	2 - 30
2.3.2.2 Don't Care Conditions in Logic Design	2 - 31
2.3.2.3 Minimization of Incompletely Specified Functions	2 - 31
2.4 Simplification of POS Expression	2 - 34
2.5 Summary of Rules for K-Map Simplification	2 - 38

2.6 Limitations of Karnaugh Map	2 - 39
2.7 Quine-McCluskey or Tabular Method.....	2 - 39
2.7.1 Algorithm for Generating Prime Implicants	2 - 40
2.7.2 Quine McCluskey using Don't Care Terms	2 - 44
2.7.3 Prime Implicant Table and Redundant Prime Implicants	2 - 48
2.7.4 Advantages and Disadvantages of Quine McCluskey Method	2 - 49
2.8 Implementation of Boolean Function using Logic Gates	2 - 50
2.8.1 Implementation of SOP Boolean Expression	2 - 50
2.8.2 Implementation of POS Boolean Expression	2 - 50
2.9 Universal Gates.....	2 - 52
2.9.1 NAND Gate	2 - 52
2.9.2 NOR Gate.....	2 - 54
2.10 NAND-NAND Implementation	2 - 57
2.11 NOR-NOR Implementation	2 - 64

Chapter - 3 Combinational Logic Design	(3 - 1) to (3 - 72)
3.1 Introduction	3 - 2
3.1.1 Analysis Procedure	3 - 2
3.1.2 Design Procedure	3 - 5
3.2 Code Converter.....	3 - 6
3.3 Adders.....	3 - 16
3.3.1 Half Adder	3 - 16
3.3.2 Full Adder	3 - 17
3.4 Subtractors	3 - 19
3.4.1 Half Subtractor	3 - 19
3.4.2 Full-Subtractor	3 - 20
3.5 Parallel Adder	3 - 22
3.6 Parallel Subtractor	3 - 22
3.7 Parallel Adder / Subtractor	3 - 23
3.8 Four-bit Parallel Adder (7483/74283).....	3 - 24
3.9 BCD Adder.....	3 - 25

3.10 Look-Ahead Carry Adder.....	3 - 28
3.11 Multiplexers (MUX).....	3 - 32
3.11.1 2 : 1 Multiplexer	3 - 33
3.11.2 4 : 1 Multiplexer.....	3 - 34
3.11.3 8 : 1 Multiplexer	3 - 35
3.11.4 Quadruple 2 to 1 Multiplexer	3 - 35
3.11.5 The 74151 Multiplexer.....	3 - 36
3.11.6 The 74XX153 Dual 4 to 1 Multiplexer.....	3 - 37
3.11.7 Expanding Multiplexers	3 - 38
3.11.8 Implementation of Combinational Logic using MUX.....	3 - 40
3.11.9 Applications of Multiplexer	3 - 48
3.11.10 Multiplexer ICs	3 - 49
3.12 Demultiplexers (DEMUX).....	3 - 49
3.12.1 Types of Demultiplexers.....	3 - 50
3.12.1.1 1 : 4 Demultiplexer	3 - 50
3.12.1.2 1 : 8 Demultiplexer	3 - 51
3.12.2 Expanding Demultiplexers	3 - 52
3.12.3 Implementation of Combinational Logic using Demultiplexer	3 - 53
3.12.4 Applications of Demultiplexer	3 - 55
3.12.5 Demultiplexer ICs	3 - 55
3.13 Decoder	3 - 55
3.13.1 Binary Decoder	3 - 56
3.13.2 The 74X138 3-to-8 Decoder.....	3 - 57
3.13.3 Expanding Cascading Decoders.....	3 - 58
3.13.4 Realization of Boolean Function using Decoder.....	3 - 60
3.13.5 Applications of Decoder	3 - 61
3.13.6 Decoder ICs	3 - 61
3.14 Magnitude Comparator using 7485.....	3 - 62
3.14.1 IC 7485 (4-bit Comparator)	3 - 64
3.15 Parity Generator and Checker using 74180.....	3 - 67

UNIT - I

1

Review of Number Systems

Contents

- 1.1 *Number Systems*
- 1.2 *Representation of Numbers of Different Radix*
- 1.3 *Conversion of Numbers from One Radix to Another Radix*
- 1.4 *Representation of Signed Numbers*
- 1.5 *BCD (Binary Coded Decimal) Codes*

1.1 Number Systems

- Number system is a basis for counting various items.
 - The decimal number system has 10 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
 - Modern computers communicate and operate with binary numbers which use only the digits 0 and 1.
 - When decimal quantities are represented in the binary form, they take more digits.
 - For large decimal numbers people have to deal with very large binary strings and therefore, they do not like working with binary numbers. This fact gave rise to three new number systems : **Octal**, **Hexadecimal** and **Binary Coded Decimal (BCD)**.

Review Questions

1. Explain various number systems.
 2. Name the number system used in computers.

1.2 Representation of Numbers of Different Radix

1.2.1 Decimal Number System

- In decimal number system we can express any decimal number in units, tens, hundreds, thousands and so on.
 - When we write a decimal number say, 5678.9, we know it can be represented as $5000 + 600 + 70 + 8 + 0.9 = 5678.9$
 - The decimal number 5678.9 can also be written as 5678.9_{10} , where the 10 subscript indicates the **radix or base**.
 - The position of a digit with reference to the decimal point determines its value/weight. The sum of all the digits multiplied by their weights gives the total number being represented.
 - The leftmost digit, which has the greatest weight is called the **most significant digit** and the rightmost digit, which has the least weight, is called the **least significant digit**.
 - Fig. 1.2.1 shows decimal digit and its weights expressed as a power of 10.

	10^3	10^2	10^1	10^0	10^{-1}
	5	6	7	8	.
In power of 10	$5 \cdot 10^3$	$6 \cdot 10^2$	$7 \cdot 10^1$	$8 \cdot 10^0$	$9 \cdot 10^{-1}$

Fig. 1.2.1 Representation of a decimal number

1.2.2 Binary Number System

- Binary system with its two digits is a **base-two system**.
- The two binary digits (bits) are 1 and 0.
- In binary system, weight is expressed as a power of 2.
- The Fig. 1.2.2 (a) shows representation of binary number 1101.101 in power of 2.
- By adding each digit of a binary number in a power of 2 we can find the decimal equivalent of the given binary number.

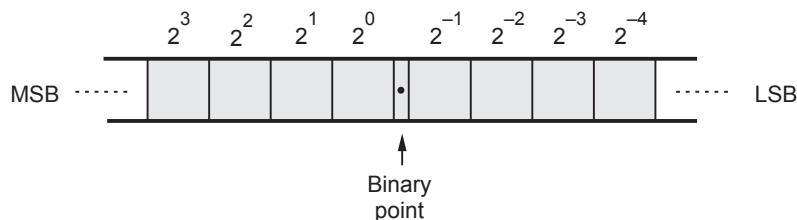


Fig. 1.2.2 Binary position values as a power of 2

$$\begin{array}{ccccccccccccc}
 & 2^3 & & 2^2 & & 2^1 & & 2^0 & & & 2^{-1} & & 2^{-2} & & 2^{-3} \\
 N = & \boxed{1} & | & \boxed{1} & | & 0 & | & \boxed{1} & | & \cdot & | & \boxed{1} & | & 0 & | & \boxed{1} \\
 & 1 & 2^3 & | & 1 & 2^2 & | & 0 & 2^1 & | & 1 & 2^0 & | & \cdot & | & 1 & 2^{-1} & | & 0 & 2^{-2} & | & 1 & 2^{-3} \\
 N = & 1 & 2^3 + 1 & 2^2 + 0 & 2^1 + 1 & 2^0 & + & 1 & 2^{-1} + 0 & 2^{-2} + 1 & 2^{-3} & = (13.625)_{10}
 \end{array}$$

Fig. 1.2.2 (a)

1.2.3 Octal Number System

- The octal number system uses first eight digits of decimal number system : 0, 1, 2, 3, 4, 5, 6 and 7. As it uses 8 digits, its base is 8.
- For example, the octal number 5632.471 can be represented in power of 8 as shown in Fig. 1.2.2 (b).

$$\begin{array}{ccccccccccccc}
 & 8^3 & & 8^2 & & 8^1 & & 8^0 & & & 8^{-1} & & 8^{-2} & & 8^{-3} \\
 N = & \boxed{5} & | & \boxed{6} & | & 3 & | & \boxed{2} & | & \cdot & | & \boxed{4} & | & 7 & | & \boxed{1} \\
 & 5 & 8^3 & | & 6 & 8^2 & | & 3 & 8^1 & | & 2 & 8^0 & | & \cdot & | & 4 & 8^{-1} & | & 7 & 8^{-2} & | & 1 & 8^{-3} \\
 N = & 5 & 8^3 + 6 & 8^2 + 3 & 8^1 + 2 & 8^0 & + & 4 & 8^{-1} + 7 & 8^{-2} + 1 & 8^{-3} & = (2970.611328)_{10}
 \end{array}$$

Fig. 1.2.2 (b)

- By adding each digit of an octal number in a power of 8 we can find the decimal equivalent of the given octal number.

1.2.4 Hexadecimal Number System

- The hexadecimal number system has a base of 16 having 16 characters : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- Table 1.2.1 shows the relationship between decimal, binary, octal and hexadecimal.
- For example, 3FD.84 can be represented in power of 16 as shown below.
- By adding each digit of a hexadecimal number in a power of 16 we can find decimal equivalent of the given hexadecimal number.

Decimal	Binary	Octal	Hexadecimal
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	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table 1.2.1 Relation between decimal, binary, octal and hexadecimal numbers

$$\begin{array}{ccccccccc}
 & 16^2 & 16^1 & 16^0 & & 16^{-1} & 16^{-2} \\
 N = & \boxed{3} & \boxed{F} & \boxed{D} & \cdot & \boxed{8} & \boxed{4} \\
 & 3 \cdot 16^2 & F \cdot 16^1 & D \cdot 16^0 & \cdot & 8 \cdot 16^{-1} & 4 \cdot 16^{-2} \\
 N = & 3 \cdot 16^2 + F \cdot 16^1 + D \cdot 16^0 & + & 8 \cdot 16^{-1} + 4 \cdot 16^{-2} \\
 & = 3 \cdot 16^2 + 15 \cdot 16^1 + 13 \cdot 16^0 & + & 8 \cdot 16^{-1} + 4 \cdot 16^{-2} & = (1021.515625)_{10}
 \end{array}$$

1.2.5 Format of a Binary Number

- A single digit in the binary number is called **bit**.
- The following figure shows the format of binary number. Four binary digits form a **nibble**, eight binary digits form a **byte**, sixteen binary digits form a **word** and thirty-two binary digits form a **double-word**.

$b_{31} b_{30} b_{29} b_{28}$	$b_{27} b_{26} b_{25} b_{24}$	$b_{23} b_{22} b_{21} b_{20}$	$b_{19} b_{18} b_{17} b_{16}$	$b_{15} b_{14} b_{13} b_{12}$	$b_{11} b_{10} b_9 b_8$	$b_7 b_6 b_5 b_4$	$b_3 b_2 b_1 b_0$	Bit				
Nibble 7	Nibble 6	Nibble 5	Nibble 4	Nibble 3	Nibble 2	Nibble 1	Nibble 0					
Byte 3		Byte 2		Byte 1		Byte 0						
Word 1				Word 0								
Double word												

Nibble : 4-bits can represent $2^4 = 16$ distinct values

Byte : 8-bits can represent $2^8 = 256$ distinct values

Word : 16-bits can represent $2^{16} = 65536$ distinct values

Double word : 32-bits can represent $2^{32} = 4294967296$ distinct values

1.2.6 Counting in Radix (Base) r

- Each number system has r set of characters. For example, in decimal number system r equals to 10 has 10 characters from 0 to 9, in binary number system r equals to 2 has 2 characters 0 and 1 and so on.
- In general we can say that, a number represented in radix r, has r characters in its set and r can be any value. This is illustrated in Table 1.2.2.

Radix (Base) r	Characters in set
2 (Binary)	0, 1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4
6	0, 1, 2, 3, 4, 5
7	0, 1, 2, 3, 4, 5, 6
8 (Octal)	0, 1, 2, 3, 4, 5, 6, 7
9	0, 1, 2, 3, 4, 5, 6, 7, 8
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
:	:
16 (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Table 1.2.2 Radix and character set

1.3 Conversion of Numbers from One Radix to Another Radix

1.3.1 Binary to Octal Conversion

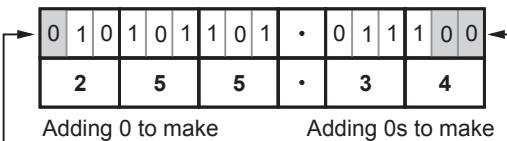
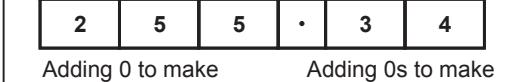
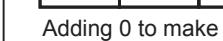
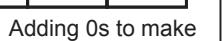
- The base for octal number is 8 and the base for binary number is 2.
- The base for octal number is the third power of the base for binary numbers. Therefore, by grouping 3 digits of binary numbers and then converting each group digit to its octal equivalent we can convert binary number to its octal equivalent.

Example 1.3.1 Convert 10101101.0111 to octal equivalent.

Solution :

Step 1 : Make group of 3-bits starting from LSB for integer part and MSB for fractional part, by adding 0s at the end, if required.

Step 2 : Write equivalent octal number for each group of 3-bits.

Step 1. 	Step 2. 	Binary (Base 2) Octal (Base 8)
Adding 0 to make a group of 3-bits 	Adding 0s to make a group of 3-bits 	

$$\therefore (10101101.0111)_2 = (255.34)_8$$

1.3.2 Octal to Binary Conversion

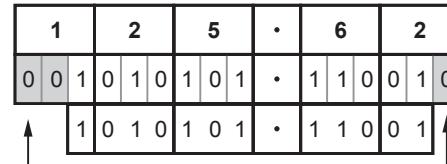
- Conversion from octal to binary is a reversal of the process explained in the previous section. Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

Example 1.3.2 Convert $(125.62)_8$ to binary.

Solution :

Step 1 : Write equivalent 3-bit binary number for each octal digit.

Step 2 : Remove any leading or trailing zeros.

Step 1. 	Octal (Base 8)
Step 2. 	

$$\therefore (125.62)_8 = (1010101.11001)_2$$

1.3.3 Binary to Hexadecimal Conversion

- The base for hexadecimal numbers is 16 and the base for binary numbers is 2.
- The base for hexadecimal number is the fourth power of the base for binary numbers. Therefore, by grouping 4 digits of binary numbers and then converting each group digit to its hexadecimal equivalent we can convert binary number to its hexadecimal equivalent.

Example 1.3.3 Convert $1101101110 \cdot 1001101$ to hexadecimal equivalent.

Solution :

Step 1 : Make group of 4-bits starting from LSB for integer part and MSB for fractional part, by adding 0s at the end, if required.

Step 2 : Write equivalent hexadecimal number for each group of 4-bits.

Step 1.	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>.</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table>	0	0	1	1	0	1	1	0	1	1	1	0	.	1	0	0	1	1	0	1	0	Step 2.	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>3</td><td>6</td><td>E</td><td>.</td><td>9</td><td>A</td></tr> </table>	3	6	E	.	9	A
0	0	1	1	0	1	1	0	1	1	1	0	.	1	0	0	1	1	0	1	0										
3	6	E	.	9	A																									
Adding 0s to make a group of 4-bits			Adding 0 to make a group of 4-bits																											

$$\therefore (1101101110.1001101)_2 = (36E.9A)_{16}$$

1.3.4 Hexadecimal to Binary Conversion

- Conversion from hexadecimal to binary is a reversal of the process explained in the previous section. Each digit of the hexadecimal number is individually converted to its binary equivalent to get hexadecimal to binary conversion of the number.

Example 1.3.4 Convert $(8A9.B4)_{16}$ to binary.

Solution :

Step 1 : Write equivalent 4-bit binary number of each hexadecimal digit.

Step 2 : Remove any leading or trailing zeros.

Step 1.	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>8</td><td>A</td><td>9</td><td>.</td><td>B</td><td>4</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>.</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </table>	8	A	9	.	B	4	1	0	0	0	1	0	1	0	1	0	0	1	.	1	0	1	1	0	1	0	0
8	A	9	.	B	4																							
1	0	0	0	1	0	1	0	1	0	0	1	.	1	0	1	1	0	1	0	0								
Trailing zeros																												

$$\therefore (8A9.B4)_{16} = (1000 1010 1001.101101)_2$$

1.3.5 Octal to Hexadecimal Conversion

The easiest way to convert octal number to hexadecimal number is given below.

- Convert octal number to its binary equivalent.
- Convert binary number to its hexadecimal equivalent.

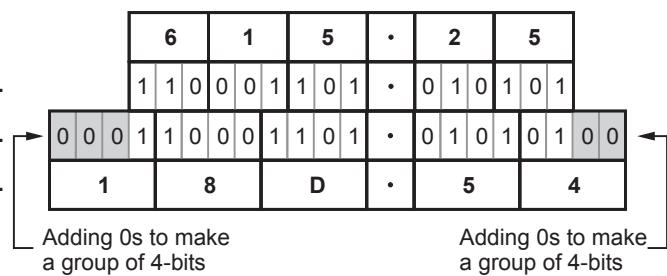
Example 1.3.5 Convert $(615.25)_8$ to its hexadecimal equivalent.

Solution :

Step 1 : Write equivalent 3-bit binary number for each octal digit.

Step 2 : Make group of 4-bits starting from LSB for integer part and MSB for fractional part by adding 0s at the end, if required.

Step 3 : Write equivalent octal number for each group to 4-bits.

<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>6</td><td>1</td><td>5</td><td>.</td><td>2</td><td>5</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>8</td><td>D</td><td>.</td><td>5</td><td>4</td></tr> </table>	6	1	5	.	2	5	1	1	0	0	1	1	0	0	0	1	1	0	1	8	D	.	5	4	Octal (Base 8) Binary (Base 2) Binary (Base 2) Hex (Base 16)
6	1	5	.	2	5																				
1	1	0	0	1	1																				
0	0	0	1	1	0																				
1	8	D	.	5	4																				
Step 1. Step 2. Step 3.																									
	Adding 0s to make a group of 4-bits Adding 0s to make a group of 4-bits																								

1.3.6 Hexadecimal to Octal Conversion

The easiest way to convert hexadecimal number to octal number is given below.

1. Convert hexadecimal number to its binary equivalent.
2. Convert binary number to its octal equivalent.

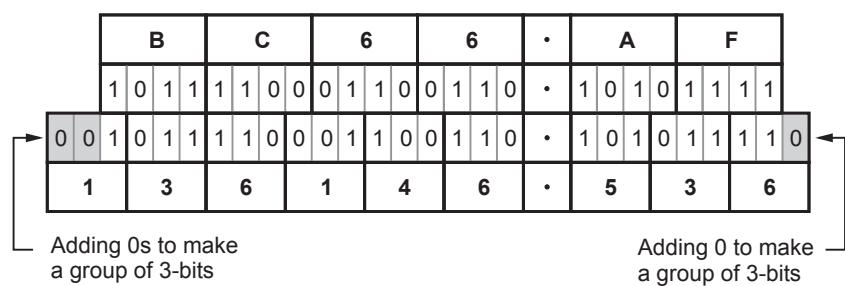
Example 1.3.6 Convert $(BC66.AF)_{16}$ to its octal equivalent.

Solution :

Step 1 : Write equivalent 4-bit binary number for each hexadecimal digit.

Step 2 : Make group of 3-bits starting from LSB for integer part and MSB for fractional part by adding 0s at the end, if required.

Step 3 : Write equivalent octal number for each group of 3-bits.

<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>B</td><td>C</td><td>6</td><td>6</td><td>.</td><td>A</td><td>F</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>3</td><td>6</td><td>1</td><td>4</td><td>6</td><td>.</td></tr> </table>	B	C	6	6	.	A	F	1	0	1	1	1	0	0	0	0	1	0	1	1	0	1	3	6	1	4	6	.	Hex (Base 16) Binary (Base 2) Binary (Base 2) Octal (Base 8)
B	C	6	6	.	A	F																							
1	0	1	1	1	0	0																							
0	0	1	0	1	1	0																							
1	3	6	1	4	6	.																							
Step 1. Step 2. Step 3.																													
	Adding 0s to make a group of 3-bits Adding 0 to make a group of 3-bits																												

1.3.7 Converting Any Radix to Decimal

- In general, numbers can be represented as

$$N = A_{n-1}r^{n-1} + A_{n-2}r^{n-2} + \dots + A_1r^1 + A_0r^0 + A_{-1}r^{-1} + A_{-2}r^{-2} + \dots C_{-m}r^{-m}$$

where N = Number in decimal

A = Digit

r = Radix or base of a number system

n = The number of digits in the integer portion of number

m = The number of digits in the fractional portion of number

- From this general equation we can convert number with any radix into its decimal equivalent. This is illustrated using following example.

Example 1.3.7 Convert $(3102.12)_4$ to its decimal equivalent.

$$\begin{aligned}\text{Solution : } N &= 3 \times 4^3 + 1 \times 4^2 + 0 \times 4^1 + 2 \times 4^0 + 1 \times 4^{-1} + 2 \times 4^{-2} \\ &= 192 + 16 + 0 + 2 + 0.25 + 0.125 = 210.375_{10}\end{aligned}$$

Example 1.3.8 Determine the value of base x , if : $(193)_x = (623)_8$

$$\text{Solution : } (193)_x = (623)_8$$

Converting octal into decimal : $6 \times 8^2 + 2 \times 8 + 3 = (403)_{10} = (623)_8$

$$\therefore (193)_x = 1 \times x^2 + 9 \times x + 3 \times x^0 = (403)_{10}$$

$$\therefore x^2 + 9x + 3 = 403 \quad \therefore x = 16 \text{ or } x = -25$$

Negative is not applicable

$$\therefore x = 16$$

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

1.3.8 Conversion of Decimal Number to Any Radix Number

Step 1 : Convert integer part.

Step 2 : Convert fractional part.

- The conversion of integer part is accomplished by successive division method and the conversion of fractional part is accomplished by successive multiplication method.

Steps in Successive Division Method

- Divide the integer part of decimal number by desired base number, store quotient (Q) and remainder (R).
- Consider quotient as a new decimal number and repeat step 1 until quotient becomes 0.
- List the remainders in the reverse order.

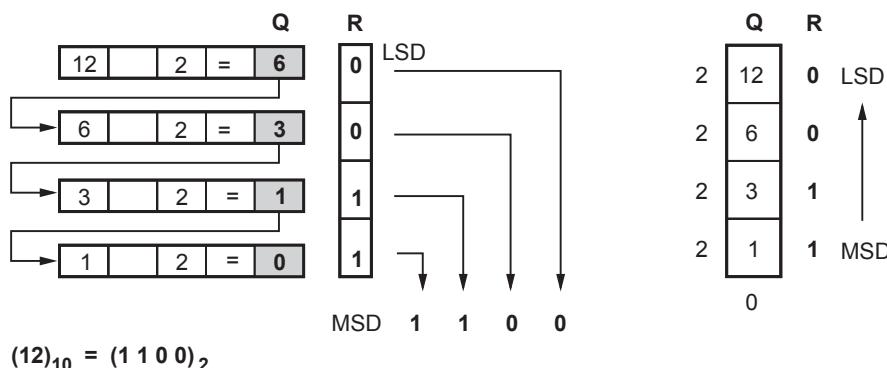
Steps in Successive Multiplication Method

1. Multiply the fractional part of decimal number by desired base number.
2. Record the integer part of product as carry and fractional part as new fractional part.
3. Repeat steps 1 and 2 until fractional part of product becomes 0 or until you have many digits as necessary for your application.
4. Read carries downwards to get desired base number.

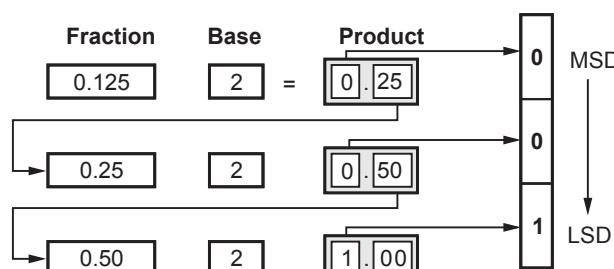
Example 1.3.9 Convert 12.125 decimal into binary.

Solution :

Integer part : Conversion of integer part by successive division method.



Fractional part : Conversion of fractional part by successive multiplication method.

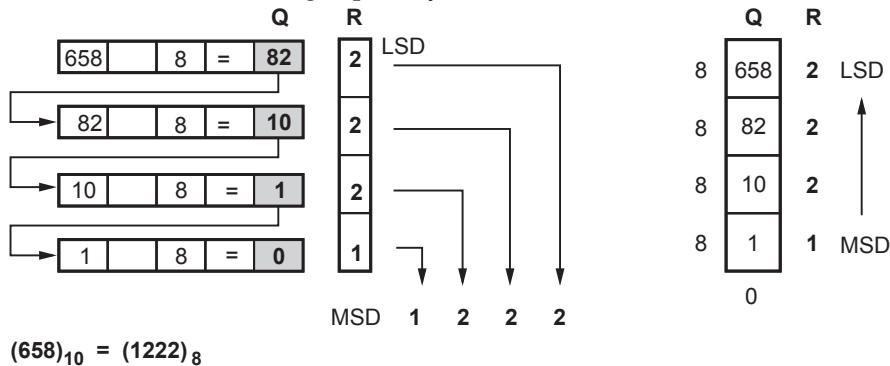


$$(12.125)_{10} = (1100.001)_2$$

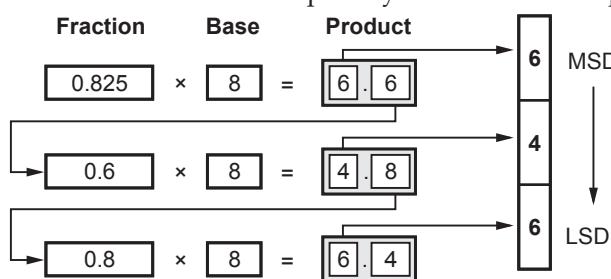
Example 1.3.10 Convert 658.825 decimal into octal.

Solution :

Integer part : Conversion of integer part by successive division method.



Fractional part : Conversion of fractional part by successive multiplication method.



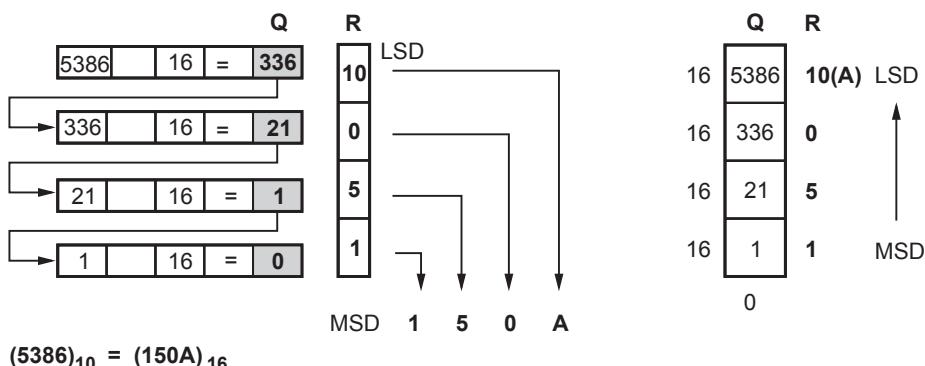
$$(658.825)_{10} = (1222.646)_{8}$$

In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 8 until we have as many digits as necessary for our application.

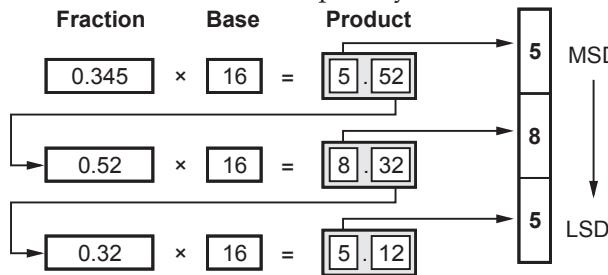
Example 1.3.11 Convert 5386.345 decimal into hexadecimal.

Solution :

Integer part : Conversion of integer part by successive division method.



Fractional part : Conversion of fractional part by successive multiplication method.



$$(0.345)_{10} = (0.585)_{16}$$

$$(5386.345)_{10} = (150A.585)_{16}$$

In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 16 until we have as many digits as necessary for our application.

Example 1.3.12 The solution to the quadratic equation $x^2 - 11x + 22 = 0$ is $x = 3$ and $x = 6$. What is the base of the numbers ?

Solution : Given quadratic equation is $x^2 - 11x + 22 = 0$ and it is given that $x = 3$ and $x = 6$

$$\therefore (x - 3)(x - 6) = x^2 - 11x + 22 \quad \dots(1)$$

The numbers 3 and 6 are same in any base, whose value is more than 6.

$$\text{i.e., } (3)_b = (3)_{10}$$

$$(6)_b = (6)_{10} \quad \text{where } b = \text{base}$$

Solving equation (1) we have,

$$(x^2 - 9x + 18)_{10} = (x^2 - 11x + 22)_b$$

Comparing coefficients of x we have

$$(-9)_{10} = (-11)_b$$

$$\therefore (9)_{10} = (11)_b$$

$$\therefore 9 = b^1 + b^0 = b + 1$$

$$\therefore b = 8$$

or comparing constants we have

$$(18)_{10} = (22)_b$$

$$\therefore 18 = 2b + 2$$

$$\therefore b = 8$$

Example 1.3.13 Determine the value of b for the following :

$$i) (292)_{10} = (1204)_b \quad ii) (16)_{10} = (100)_b$$

Solution : i) $(292)_{10} = 1 \times b^3 + 2 \times b^2 + 0 \times b^1 + 4 \times b^0$

$$292 = b^3 + 2b^2 + 4$$

$$\therefore b = 6$$

$$ii) (16)_{10} = 1 \times b^2 + 0 \times b^1 + 0 \times b^0 = b^2$$

$$\therefore b = 4$$

1.4 Representation of Signed Numbers

- In practice, we use plus sign to represent positive number and minus sign to represent negative number.
- However, because of hardware limitations, in computers, both positive and negative numbers are represented with only binary digits.
- There are three ways to represent signed numbers :
 - Sign-Magnitude representation
 - 1's complement representation
 - 2's complement representation

1.4.1 Sign-Magnitude Representation

- The leftmost bit (sign bit) in the number represents sign of the number.

- The sign bit is 0 for positive numbers and it is 1 for negative numbers.

- Fig. 1.4.1 shows the sign magnitude format for 8-bit signed number.

- Here are some examples of sign-magnitude numbers.

$$1. + 6 = 0000\ 0110 \quad 2. - 14 = 1000\ 1110$$

$$3. + 24 = 0001\ 1000 \quad 4. - 64 = 1100\ 0000$$

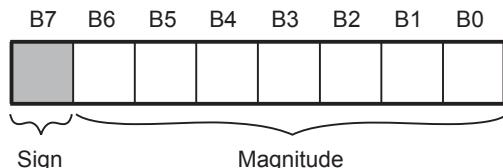
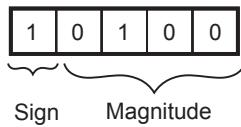


Fig. 1.4.1

Maximum positive number :	0 1 1 1 1 1 1 1	= +127
Maximum negative number :	1 1 1 1 1 1 1 1	= -127

Example 1.4.1 A binary machine handles negative numbers in the true magnitude form. How will -4 be stored in a register with a sign bit and 4 bits representing magnitude ?

Solution :



1.4.2 1's Complement Representation

- The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.

Example 1.4.2 Find 1's complement of $(11010100)_2$.

Solution :

1	1	0	1	0	1	0	0	Number
↓	↓	↓	↓	↓	↓	↓	↓	NOT operation
0	0	1	0	1	0	1	1	1's complement of number

1.4.3 2's Complement Representation

- The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as,

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

Example 1.4.3 Find 2's complement of $(11000100)_2$.

Solution :

1	1	0	0	0	1	0	0	Number
					1	1		Carry
0	0	1	1	1	0	1	1	1's complement of number
+							1	Add 1
0	0	1	1	1	1	0	0	2's complement of number

Example 1.4.4 Represent the decimal number -200 and 200 using 2's complement binary form.

Solution : To represent 200 and -200 we need 9-bit number system.

+ 200	=	0 1 1 0 0 1 0 0 0	
- 200	=	1 0 0 1 1 0 1 1 1	1's complement
	+		Add 1
		1 0 0 1 1 1 0 0 0	2's complement

Fig. 1.5.2

The Table 1.4.1 lists all possible 4-bit signed binary numbers in the three representations. Looking at the Table 1.4.1 we understand following points :

- Positive numbers in all three representations are identical and have 0 in the leftmost position.
- All negative numbers have a 1 in the leftmost bit position.
- The signed-2's complement system has only one representation for 0, which is always positive.
- The signed-magnitude and 1's complement systems have either a positive 0 or a negative 0.
- With four bits, we can represent 16 binary numbers.

Decimal	Signed-2's complement	Signed-1's complement	Signed magnitude
+ 7	0111	0111	0111
+ 6	0110	0110	0110
+ 5	0101	0101	0101
+ 4	0100	0100	0100
+ 3	0011	0011	0011
+ 2	0010	0010	0010
+ 1	0001	0001	0001
+ 0	0000	0000	0000
- 0	-	1111	1000
- 1	1111	1110	1001
- 2	1110	1101	1010
- 3	1101	1100	1011
- 4	1100	1011	1100
- 5	1011	1010	1101
- 6	1010	1001	1110
- 7	1001	1000	1111
- 8	1000	-	-

Table 1.4.1

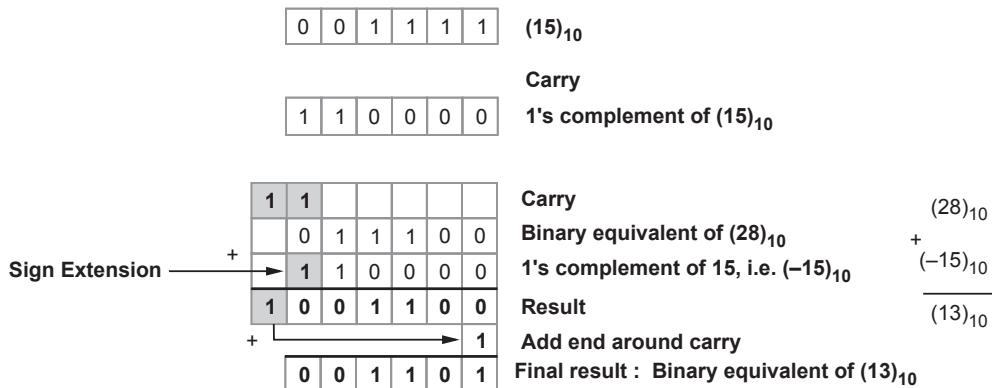
1.4.4 Binary Subtraction using 1's Complement Method

In a 1's complement subtraction, negative number is represented in the 1's complement form and actual addition is performed to get the desired result. For example, operation A – B is performed using following steps :

1. Take 1's complement of B.
2. Result $\leftarrow A + 1\text{'s complement of } B\right.$
3. If carry is generated then the result is positive and in the true form. Add carry to the result to get the final result.
4. If carry is not generated then the result is negative and in the 1's complement form.

Example 1.4.5 Perform $(28)_{10} - (15)_{10}$ using 6-bit 1's complement representation.

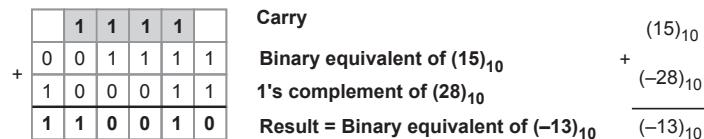
Solution : $(28)_{10} = (011100)_2$
 $(15)_{10} = (001111)_2$



Example 1.4.6 Perform $(15)_{10} - (28)_{10}$ using 6-bit 1's complement representation.

Solution : $(15)_{10} = (001111)_2$

$$(28)_{10} = (011100)_2$$



Verification 0 0 1 1 0 1 1's complement of result (Binary equivalent of (13)₁₀)

1.4.5 Binary Subtraction using 2's Complement Method

In a 2's complement subtraction, negative number is represented in the 2's complement form and actual addition is performed to get the desired result. For example, operation A – B is performed using following steps :

1. Take 2's complement of B.
2. Result $\leftarrow A + 2^{\text{'}}\text{ complement of } B$.
3. If carry is generated then the result is positive and in the true form. In this case, carry is ignored.
4. If carry is not generated then the result is negative and in the 2's complement form.

Example 1.4.7 Perform $(28)_{10} - (15)_{10}$ using 6-bit 2's complement representation.

Solution : $(28)_{10} = (011100)_2$

$$(15)_{10} = (001111)_2$$

<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> </table>	0	0	1	1	1	1	(15) ₁₀
0	0	1	1	1	1		
	Carry						
	1's complement of $(15)_{10}$						
	Add 1						
	2's complement of 15, i.e., $(-15)_{10}$						
	Carry						
	Binary equivalent of $(28)_{10}$						
	2's complement of 15, i.e. $(-15)_{10}$						
	Result : Binary equivalent of $(13)_{10}$						
Sign Extension → + Ignore Carry →	$ \begin{array}{r} 1 \quad 1 \\ + \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \\ \hline 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \end{array} $						
	$ \begin{array}{r} (28)_{10} \\ + (-15)_{10} \\ \hline (13)_{10} \end{array} $						

Example 1.4.8 Perform $(15)_{10} - (28)_{10}$ using 6-bit 2's complement representation.

Solution : $(15)_{10} = (001111)_2$

$$(28)_{10} = (011100)_2$$

<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> </table>	0	1	1	1	0	0	Binary equivalent of $(28)_{10}$															
0	1	1	1	0	0																	
	Carry																					
	1's complement of $(28)_{10}$																					
	Add 1																					
	2's complement of $(28)_{10}$, i.e., $(-28)_{10}$																					
	Carry																					
	Binary equivalent of $(15)_{10}$																					
	2's complement of $(28)_{10}$																					
	No carry, thus result is negative and in 2's complement form																					
	Verification																					
	1's complement of result																					
	Add 1																					
	- Result = Binary equivalent of $(13)_{10}$																					
No carry → 0 Verification → <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>+ </td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr> <tr><td></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> </table>		0	0	1	1	0	0	+						1		0	0	1	1	0	1	$ \begin{array}{r} (15)_{10} \\ + (-28)_{10} \\ \hline (-13)_{10} \end{array} $
	0	0	1	1	0	0																
+						1																
	0	0	1	1	0	1																

Review Questions

1. What are 1's complement and 2's complement numbers ?
2. State advantages of 2's complement number representation.
3. What are the different ways to represent a negative number ?
4. State the different ways for representing the signed binary numbers.

1.5 BCD (Binary Coded Decimal) Codes

- BCD is an abbreviation for Binary Coded Decimal.
- BCD is a numeric code in which each digit of a decimal number is represented by a separate group of 4-bits. The most common BCD code is 8-4-2-1 BCD.
- It is called 8-4-2-1 BCD because the weights associated with 4 bits are 8-4-2-1 from left to right. This means that, bit-3 has weight 8, bit-2 has weight 4 bit-1 has weight 2 and bit-0 has weight 1.
- The Table 1.5.1 shows the 4-bit 8-4-2-1 BCD code used to represent a single decimal digit.
- In multidigit coding, each decimal digit is individually coded with 8-4-2-1 BCD code, as shown in the Fig. 1.5.3. Total 8-bits are required to encode 58_{10} in 8-4-2-1 BCD.

Decimal Digit	BCD code			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 1.5.1 8-4-2-1 BCD code

Decimal	5				8			
	0	1	0	1	1	0	0	0
8-4-2-1 BCD								

Fig. 1.5.1

Advantages

1. Easy to convert between it and decimal.

Disadvantages

1. Since it requires more binary digits to represent multi-digit number than binary number system. It is less **efficient**. For example, to represent the same number (58) in binary : 111010_2 , we require only 6 digits.
2. Arithmetic operations are more complex.

1.5.1 BCD Addition

- The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Case 1 : Sum equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

$ \begin{array}{r} & 1 & 1 & & \\ + 6 & 0 & 1 & 1 & 0 \\ 3 & 0 & 0 & 1 & 1 \\ \hline 9 & 0 & 1 & 0 & 0 & 1 \end{array} $	Carry	$ \begin{array}{r} & 5 & & \\ + 2 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 & 1 \end{array} $	Carry
		Valid BCD numbers	
		BCD for 6 BCD for 3 BCD for 9	
		BCD for 5 BCD for 2 BCD for 7	

- The addition is carried out as in normal binary addition and the sums are 1 0 0 1 and 0 1 1 1 which are valid BCD codes.

Case 2 : Sum greater than 9 with carry 0

- Let us consider addition of 6 and 8 in BCD.

$ \begin{array}{r} & & 0 & 1 & 1 & 0 \\ + 6 & 1 & 0 & 0 & 0 \\ 8 & 0 & 1 & 1 & 0 \end{array} $	BCD for 6 BCD for 3	Invalid BCD number (1 1 1 0) > 9
Carry		

- The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown follows.

$ \begin{array}{r} & 6 & & \\ + 8 & 1 & 0 & 0 & 0 \\ \hline 14 & 1 & 1 & & \\ & 1 & 1 & 1 & 0 \\ & + 0 & 1 & 1 & 0 \\ \hline \text{Carry} & 1 & 0 & 1 & 0 & 0 \end{array} $	BCD for 6 BCD for 8	Invalid BCD number Add 6 for correction
BCD for 14 Decimal		

- After addition of 6 carry is produced into the second decimal position.

Case 3 : Sum equals 9 or less with carry 1

- Let us consider addition of 8 and 9 in BCD.

$ \begin{array}{r} 8 \\ + 9 \\ \hline 17 \end{array} $	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td style="text-align: center;">1</td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr> <tr><td></td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr> <tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr> </table>	1						1	0	0	0		1	0	0	1	1	0	0	0	1	Carry BCD for 8 BCD for 9 Carry <table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr> <tr><td style="text-align: center;">1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	0	0	0	1	0	0	0	1	1							
1																																						
	1	0	0	0																																		
	1	0	0	1																																		
1	0	0	0	1																																		
0	0	0	1	0	0	0	1																															
1																																						

- In this case, result (0001 0001) is valid BCD number, but it is incorrect, since carry is generated after addition of least significant digits. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown.

$ \begin{array}{r} 8 \\ + 9 \\ \hline 17 \end{array} $	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr> <tr><td></td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr> <tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr> <tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr> <tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr> <tr><td style="text-align: center;">1</td><td></td><td></td><td></td><td></td></tr> </table>	1	0	0	0	0		1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1					BCD for 8 BCD for 9 Incorrect BCD result Add 6 for correction BCD for 17
1	0	0	0	0																												
	1	0	0	1																												
0	0	0	1	0																												
0	0	0	0	0																												
0	0	0	1	0																												
1																																

Procedure of BCD Addition

- Going through these three cases of BCD addition we can summarize the BCD addition procedure as follows :
 - Add two BCD numbers using ordinary binary addition.
 - If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.
 - If the four-bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid. To correct the invalid sum.
 - Add 6 (0110_2) to the four-bit sum and ignore carry of this addition.
 - Add 1 (0001_2) to the next higher-order BCD digit.

1.5.2 Excess-3 Code

- The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.
- For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). It is a **non-weighted code**.

- Table 1.5.2 shows excess-3 codes to represent single decimal digit. It is **sequential code** because we get any code word by adding binary 1 to its previous code word as shown in the Table 1.5.2.
- In excess-3 code we get 9's complement of a number by just complementing each bit. Due to this excess-3 code is called **self-complementing code or reflective code**.

Decimal digit	Excess-3 code			
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Table 1.5.2 Excess-3 code

Example 1.5.1 Find the excess-3 code and its 9's complement for following decimal numbers.
a) 592 b) 403

Solution :

- a) By referring Table 1.5.2.

$$\begin{aligned} 592_{10} &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \\ \text{9's complement of } 592_{10} &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array} \end{aligned}$$

Complement of each other

- b) By referring Table 1.5.2.

$$\begin{aligned} 403_{10} &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\ \text{9's complement of } 403_{10} &= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline \end{array} \end{aligned}$$

Complement of each other

1.5.3 Gray Code

Gray code is a non-weighted code and is a special case of unit-distance code.

- In unit-distance code, bit patterns for two consecutive numbers differ in only one bit position. These codes are also called **cyclic codes**.
- The Table 1.5.3 shows the bit patterns assigned for gray code from decimal 0 to decimal 15.

- As shown in the Table 1.5.3 for gray code any two adjacent code groups differ only in one bit position.
- Reflective Property :** The gray code is also called **reflected code**. Notice that the two least significant bits for 4_{10} through 7_{10} are the mirror images of those for 0_0 through 3_{10} . Similarly, the three least significant bits for 8_{10} through 15_{10} are the mirror images of those for 0_{10} through 7_{10} .
- In general, the n least significant bits for 2^n through $2^{n+1} - 1$ are the mirror images of those for 0 through $2^n - 1$.

Decimal code	Gray code
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

Table 1.5.3 Gray code

1.5.3.1 Gray to Binary Conversion

Steps for gray to binary code conversion

- The Most Significant Bit (MSB) of the binary number is the same as the Most Significant Bit (MSB) of the gray code number. **So write down MSB as it is.**
- To obtain the next binary digit, perform an exclusive-OR-operation between the bit just written down and the next gray code bit. Write down the result.
- Repeat step 2 until all gray code bits have been exclusive-ORed with binary digits.

Example 1.5.2 Convert gray code 101011 into its binary equivalent.

A	B	A \oplus B Exclusive OR operation
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.5.4 Exclusive OR operation

Solution :

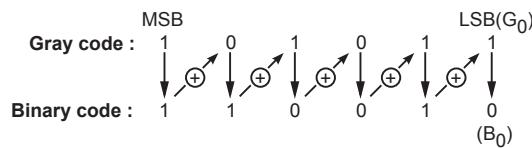


Fig. 1.5.2

$$\therefore (1010011)_{\text{gray}} = (110010)_2$$

In general we can perform gray code to binary code conversion as shown below :

$$B_n \text{ (MSB)} = G_n \text{ (MSB)}$$

$$B_2 = B_3 \oplus G_2$$

$$B_{n-1} = B_n \oplus G_{n-1} \quad \dots$$

$$B_1 = B_2 \oplus G_1$$

$$B_{n-2} = B_{n-1} \oplus G_{n-2}$$

$$B_0 = B_1 \oplus G_0$$

1.5.3.2 Binary to Gray Conversion

Steps for binary to gray code conversion

1. The MSB of the gray code is the same as the MSB of the binary number. **So write down MSB as it is.**
2. To obtain the next gray digit, perform an exclusive-OR-operation between previous and current binary bit. Write down the result.
3. Repeat step 2 until all binary bits have been exclusive-ORed with their previous ones.

Example 1.5.3 Convert 10111011 in binary into its equivalent gray code.

Solution :

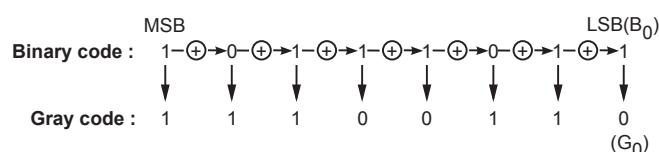


Fig. 1.5.3

$$\therefore (10111011)_2 = (11100110)_{\text{gray}}$$

Review Questions

1. What is BCD code ? State the procedure for BCD addition.
2. What is excess-3 code ?
3. What is gray code ? Explain the steps to convert binary to gray code and vice-versa.



Notes

UNIT - I

2

Logic Design Minimization Techniques

Contents

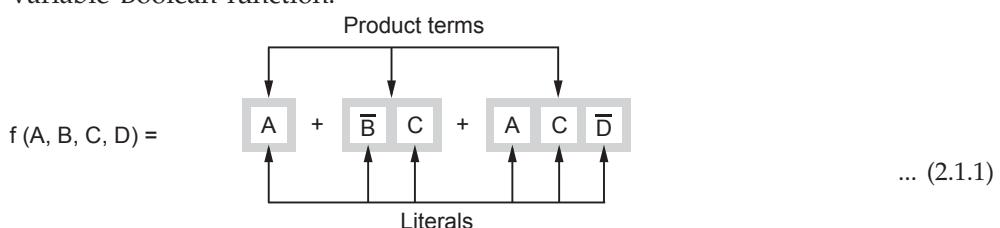
- 2.1 *Boolean Expressions*
- 2.2 *Karnaugh (K-Map) Minimization*
- 2.3 *Simplification of SOP Expression* **Dec.-09, May-13,19 ,** Marks 12
- 2.4 *Simplification of POS Expression.* **Dec.-15,** Marks 6
- 2.5 *Summary of Rules for K-Map Simplification*
- 2.6 *Limitations of Karnaugh Map*
- 2.7 *Quine-McCluskey or Tabular Method* **Dec.-09,18, May-10,14,18** · Marks 10
- 2.8 *Implementation of Boolean Function*
 using Logic Gates **Dec.-13, May-15,16,18** · Marks 4
- 2.9 *Universal Gates*
- 2.10 *NAND-NAND Implementation* **May-06,07,08,**
 **Dec.-05,06,11,15,16** Marks 8
- 2.11 *NOR-NOR Implementation* **Dec.-05,06,07** Marks 6

2.1 Boolean Expressions

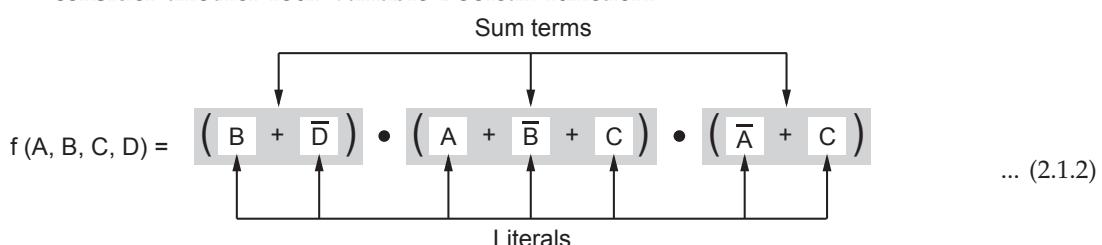
- Boolean expressions are constructed by connecting the Boolean constants and variables with the Boolean operations. These Boolean expressions are also known as **Boolean formulas**. We use Boolean expressions to describe switching function or **Boolean functions**. For example, if the Boolean expression $(A + \bar{B}) C$ is used to describe the function f , then Boolean function is written as

$$f(A, B, C) = (A + \bar{B}) C \quad \text{or} \quad f = (A + \bar{B}) C$$

- Based on the structure of Boolean expression, it can be categorized in different formulas. One such categorization are the normal formulas. Let us consider the four-variable Boolean function.



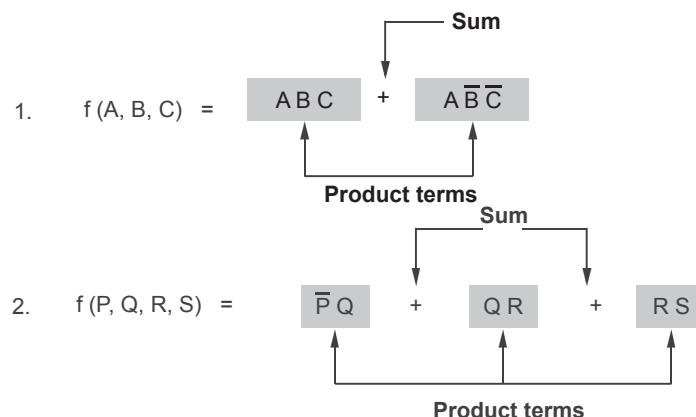
- In this Boolean function the variables are appeared either in a complemented or an uncomplemented form. Each occurrence of a variable in either a complemented or an uncomplemented form is called a **literal**. Thus, the above Boolean function 1.1.1 consists of six literals. They appear in the product terms. A **product term** is defined as either a literal or a product (also called conjunction) of literals. Function 2.1.1 contains three product terms, namely, A , $\bar{B}C$ and $AC\bar{D}$. Let us consider another four variable Boolean function.



- The above Boolean function consists of seven literals. Here, they appear in the sum terms. A **sum term** is defined as either a literal or a sum (also called disjunction) of literals. Function 2.1.2 contains three sum terms, namely, $(B + \bar{D})$, $(A + \bar{B} + C)$ and $(\bar{A} + C)$.
- The literals and terms are arranged in one of the two standard forms :
 - Sum of product form (SOP) and
 - Product of sum form (POS).
- In these standard forms, the terms that form the function may contain one, two or any number of literals.

2.1.1 Sum of Product Form

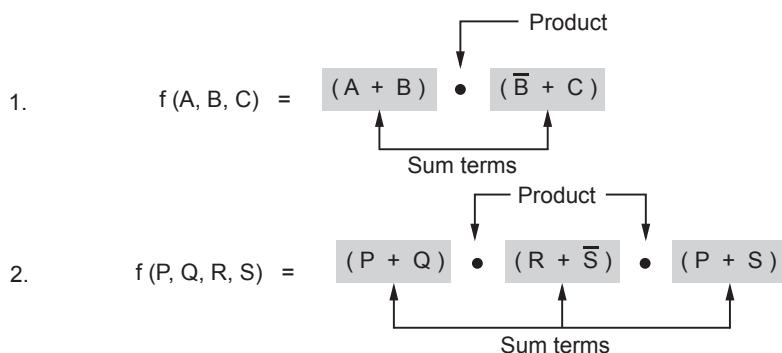
The words sum and product are derived from the symbolic representations of the OR and AND functions by + and · (addition and multiplication), respectively. But we realize that these are not arithmetic operators in the usual sense. A product term is any group of literals that are ANDed together. For example, ABC, XY and so on. A sum term is any group of literals that are ORed together such as A + B + C, X + Y and so on. A sum of products (SOP) is a group of product terms ORed together. Some examples of this form are :



Each of these sum of products expressions consist of two or more product terms (AND) that are ORed together. Each product term consists of one or more literals appearing in either complemented or uncomplemented form. For example, in the sum of products expression $ABC + A\bar{B}\bar{C}$, the first product term contains literals A, B and C in their uncomplemented form. The second product term contains B and C in their complemented (inverted) form. The sum of product form is also known as **disjunctive normal form** or **disjunctive normal formula**.

2.1.2 Product of Sum Form

A product of sums is any groups of sum terms ANDed together. Some examples of this form are :



Each of these product of sums expressions consist of two or more sum terms (OR) that are ANDed together. Each sum term consists of one or more literals appearing in either complemented or uncomplemented form. The product of sum form is also known as **conjunctive normal form** or **conjunctive normal formula**.

2.1.3 Canonical SOP and Canonical POS Forms

We can realize that in the SOP $f(A, B, C) = A\bar{B}C + A\bar{B}C + \bar{A}B\bar{C}$ form, all the individual terms do not involve all literals. For example, in expression $AB + AB\bar{C}$ the first product term do not contain literal C. If each term in SOP form contains all the literals then the SOP form is known as **canonical SOP form**. Each individual term in the canonical SOP form is called **minterm**. One canonical sum of products expression is as shown in Fig. 2.1.1.

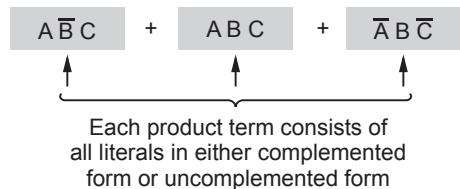


Fig. 2.1.1 Canonical SOP form

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

Fig. 2.1.2 Canonical POS form

If each term in POS form contains all the literals then the POS form is known as **canonical POS form**. Each individual term in the canonical POS form is called **maxterm**. One canonical product of sums expression is as shown in Fig. 2.1.2.

Thus we can say that, Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

2.1.4 Canonical Expressions in Canonical SOP or POS Form

Sum of product form can be converted to canonical sum of products by ANDing the terms in the expression with terms formed by ORing the literal and its complement which are not present in that term. For example for a three literal expression with literals A, B and C, if there is a term AB, where C is missing, then we form term $(C + \bar{C})$ and AND it with AB. Therefore, we get $AB(C + \bar{C}) = ABC + AB\bar{C}$.

2.1.4.1 Steps to Convert SOP to Canonical SOP Form

Step 1 : Find the missing literal in each product term if any.

Step 2 : AND each product term having missing literal/s with term/s formed by ORing the literal and its complement.

Step 3 : Expand the terms by applying distributive law and reorder the literals in the product terms.

Step 4 : Reduce the expression by omitting repeated product terms if any. Because $A + A = A$.

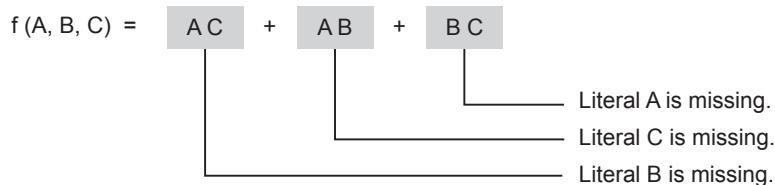
Illustrative Examples

Example 2.1.1 Convert the given expression in canonical SOP form.

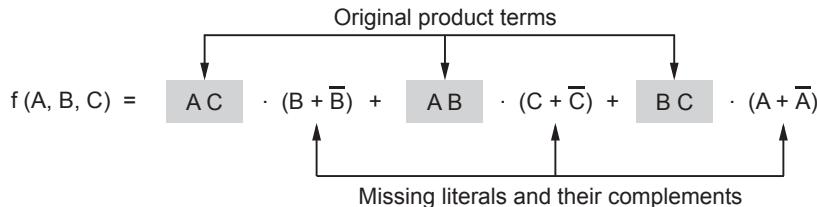
$$f(A, B, C) = AC + AB + BC$$

Solution :

Step 1 : Find the missing literal/s in each product term.



Step 2 : AND product term with (missing literal + its complement).



Step 3 : Expand the terms and reorder literals.

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

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

Note After having sufficient practice student should expand product term and reorder literals in it in a single step.

Step 4 : Omit repeated product terms.

$$\begin{aligned} f(A, B, C) &= A B C + A \bar{B} C + \boxed{A B C} + A B \bar{C} + \boxed{A B C} + \bar{A} B C \\ f(A, B, C) &= A B C + A \bar{B} C + A B \bar{C} + \bar{A} B C \end{aligned}$$

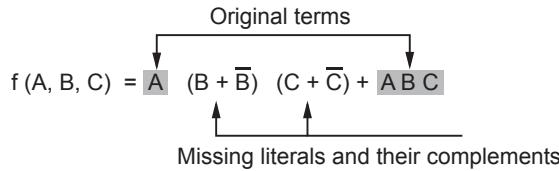
Example 1.1.2 Convert the given expression in canonical SOP form. $f(A, B, C) = A + ABC$

Solution :

Step 1 : Find the missing literal/s in each product term.

$$\begin{aligned} f(A, B, C) &= \boxed{A} + A B C \\ &\quad \uparrow \\ &\quad \text{Literals B and C are missing} \end{aligned}$$

Step 2 : AND product term with (missing literal + its complement).



Step 3 : Expand the terms and reorder literals.

$$f(A, B, C) = AB C + A B \overline{C} + A \overline{B} C + A \overline{B} \overline{C} + A B C$$

Step 4 : Omit repeated product term

$$\begin{aligned} f(A, B, C) &= AB C + A B \overline{C} + A \overline{B} C + A \overline{B} \overline{C} + \boxed{ABC} \\ &= AB C + A B \overline{C} + A \overline{B} C + A \overline{B} \overline{C} \end{aligned}$$

Repeated product term

Examples for Practice

Example 2.1.3 Express the following function in canonical SOP form

$$F_1 = AB + \overline{C}D + A\overline{B}\overline{C}$$

$$\text{Ans. : } \overline{A} \overline{B} \overline{C} + \overline{A} \overline{B} \overline{C} + A \overline{B} \overline{C} + A \overline{B} C + A \overline{B} C + A B \overline{C} + A B \overline{C} + A B C$$

Example 2.1.4 Determine the canonical SOP form of $f(x, y, z) = (x y + \bar{z})(y + x \bar{z})$

$$\text{Ans. : } xy z + xy \bar{z} + x y \bar{z} + x \bar{y} \bar{z}$$

2.1.4.2 Steps to Convert POS to Canonical POS Form

Step 1 : Find the missing literals in each sum term if any.

Step 2 : OR each sum term having missing literal/s with term/s form by ANDing the literal and its complement.

Step 3 : Expand the terms by applying distributive law and reorder the literals in the sum terms.

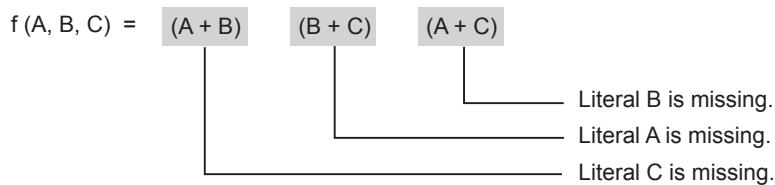
Step 4 : Reduce the expression by omitting repeated sum terms if any. Because $A \cdot A = A$.

Illustrative Examples

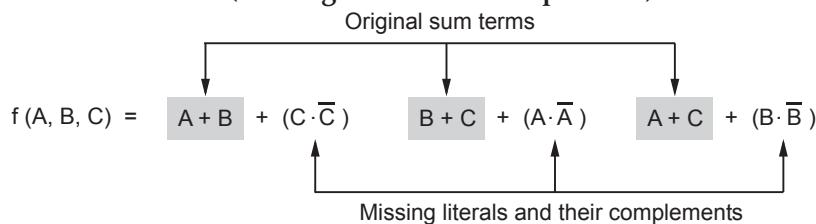
Example 2.1.5 Convert the given expression in canonical POS form.

$$f(A, B, C) = (A + B)(B + C)(A + C)$$

Solution : **Step 1 :** Find the missing literal/s in each sum term.



Step 2 : OR sum term with (missing literal • its complement).



Step 3 : Expand the terms and reorder literals.

Expand :

Since $A + BC = (A + B)(A + C)$ we have,

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + B + \bar{C})(B + C + A)(B + C + \bar{A}) \\ &\quad (A + C + B)(A + C + \bar{B}) \end{aligned}$$

Reorder :

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + B + \bar{C})(A + B + C)(\bar{A} + B + C) \\ &\quad (A + B + C)(A + \bar{B} + C) \end{aligned}$$

Step 4 : Omit repeated sum terms.

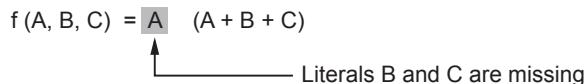
$$f(A, B, C) = (A + B + C)(A + B + \bar{C})(A + B + C)(\bar{A} + B + C)(A + B + C)(A + \bar{B} + C)$$

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

Example 2.1.6 Convert the given expression in canonical POS form.

$$Y = A \cdot (A + B + C)$$

Solution : **Step 1 :** Find the missing literal/s in each sum term.



Step 2 : OR sum term with (missing literal • its complement).

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

Step 3 : Expand the terms and reorder literals.

Since $A + BC = (A + B)(A + C)$ we have,

$$\begin{aligned} f(A, B, C) &= (A + B \cdot \bar{B} + C)(A + B \cdot \bar{B} + \bar{C})(A + B + C) \\ &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C})(A + B + C) \end{aligned}$$

Step 4 : Omit repeated sum terms.

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C}) \text{ (A + B + C)} \\ f(A, B, C) &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C}) \end{aligned}$$

Example for Practice

Example 2.1.7 Convert the given expression in canonical POS form

$$f(P, Q, R) = (P + \bar{Q})(P + R)$$

$$\text{Ans. : } (P + \bar{Q} + R)(P + \bar{Q} + \bar{R})(P + Q + R)$$

2.1.5 M Notations : Minterms and Maxterms

- Each individual term in canonical SOP form is called **minterm** and each individual term in canonical POS form is called **maxterm**. The concept of minterms and maxterms allows us to introduce a very convenient shorthand notations to express logical functions. Table 2.1.1 gives the minterms and maxterms for a three literal/variable logical function where the number of minterms as well as maxterms is $2^3 = 8$. In general, for an n-variable logical function there are 2^n minterms and an equal number of maxterms.

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	$\bar{A} B C = m_3$	$A + \bar{B} + \bar{C} = M_3$
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	$A B \bar{C} = m_6$	$\bar{A} + \bar{B} + C = M_6$
1	1	1	$A B C = m_7$	$\bar{A} + \bar{B} + \bar{C} = M_7$

Table 2.1.1 Minterms and maxterms for three variables

- As shown in Table 2.1.1 each minterm is represented by m_i and each maxterm is represented by M_i , where the subscript i is the decimal number equivalent of the natural binary number. With these shorthand notations logical function can be represented as follows :

$$\begin{aligned}
 1. \quad f(A, B, C) &= \overline{A} \overline{B} \overline{C} + \overline{A} \overline{B} C + \overline{A} B \overline{C} + A B \overline{C} \\
 &= m_0 + m_1 + m_3 + m_6 \\
 &= \sum m(0, 1, 3, 6)
 \end{aligned}$$

$$\begin{aligned}
 2. \quad f(A, B, C) &= (A + B + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C) \\
 &= M_1 \bullet M_3 \bullet M_6 \\
 &= \prod M(1, 3, 6)
 \end{aligned}$$

where \sum denotes **sum of product** while \prod denotes **product of sum**.

- We know that logical expression can be represented in the truth table form. It is possible to write logic expression in canonical SOP or POS form corresponding to a given truth table. The logic expression corresponding to a given truth table can be written in a canonical sum of products form by writing one product term for each input combination that produces an output of 1. These product terms are ORed together to create the canonical sum of products. The product terms are expressed by writing complement of a variable when it appears as an input 0, and the variable itself when it appears as an input 1. Consider, for example, the truth Table 2.1.2.

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

← \overline{ABC}
 ← \overline{ABC}
 ← ABC

Table 2.1.2

- The product corresponding to input combination 010 is \overline{ABC} , the product corresponding to input combination 011 is \overline{ABC} and product corresponding to input combination 110 is ABC . Thus the canonical sum of products form is

$$\begin{aligned}
 f(A, B, C) &= \overline{ABC} + \overline{ABC} + ABC \\
 &= m_2 + m_3 + m_6
 \end{aligned}$$

- The logic expression corresponding to a truth table can also be written in a canonical product of sums form by writing one sum term for each output 0. The sum terms are expressed by writing complement of a variable when it appears as an input 1 and the variable itself when it appears as an input 0. Consider, for example, the truth Table 2.1.3.

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	0
1	1	0	1
1	1	1	1

← $A + \overline{B} + C$
 ← $\overline{A} + B + \overline{C}$

Table 2.1.3

The sum corresponding to input combinations 010 is $A + \bar{B} + C$, and the sum corresponding to input 101 is $\bar{A} + B + \bar{C}$. Thus, the standard product of sum form is

$$\begin{aligned} f(A, B, C) &= (A + \bar{B} + C)(\bar{A} + B + \bar{C}) \\ &= M_2 \bullet M_5 \end{aligned}$$

2.1.6 Complements of Canonical Forms

The POS and SOP functions derived from the same truth table are logically equivalent. In terms of minterms and maxterms we can then write

$$\begin{aligned} f(A, B, C) &= m_0 + m_1 + m_3 + m_4 + m_6 + m_7 = M_2 + M_5 \\ \therefore f(A, B, C) &= \sum m(0, 1, 3, 4, 6, 7) = \pi M(2, 5) \end{aligned}$$

From the above expressions we can easily notice that there is a complementary type of relationship between a function expressed in terms of maxterms. Using this complementary relationship we can find logical function in terms of maxterms if function in minterms is known or vice-versa. For example, for a four variables if

$$f(A, B, C, D) = \sum m(0, 2, 4, 6, 8, 10, 12, 14)$$

then $f(A, B, C, D) = \pi M(1, 3, 5, 7, 9, 11, 13, 15)$

Example 2.1.8 Express $F = A + B'C$ as sum of minterms.

$$\begin{aligned} \text{Solution : } A + \bar{B}C &= A(B + \bar{B})(C + \bar{C}) + (A + \bar{A})\bar{B}C \\ &= (AB + A\bar{B})(C + \bar{C}) + (A\bar{B}C) + \bar{A}\bar{B}C \\ &= A BC + A \bar{B}C + A \bar{B}\bar{C} + A \bar{B}C + A \bar{B}\bar{C} + \bar{A}\bar{B}C \\ &= A BC + A \bar{B}C + A \bar{B}\bar{C} + A \bar{B}C + \bar{A}\bar{B}C \\ \therefore F &= \sum m(1, 4, 5, 6, 7) \end{aligned}$$

Example 2.1.9 Express the Boolean function $F = XY + \bar{X}Z$ in product of maxterm.

$$\begin{aligned} \text{Solution : } F &= XY + \bar{X}Z \\ &= XY(Z + \bar{Z}) + \bar{X}Z(Y + \bar{Y}) = XYZ + XY\bar{Z} + \bar{X}YZ + \bar{X}\bar{Y}Z \\ \therefore F &= \sum m(7, 6, 3, 1) = \Pi M(0, 2, 4, 5) \\ &= (X + Y + Z)(X + \bar{Y} + Z)(\bar{X} + Y + Z)(\bar{X} + Y + \bar{Z}) \end{aligned}$$

Example 2.1.10 Express the Boolean function as

(1) POS form (2) SOP form

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

Solution : 1. POS form : $D = (\bar{A} + B)(\bar{B} + C)$

$$\begin{aligned} \text{2. SOP form : } D &= \bar{A}\bar{B} + \bar{A}C + B\bar{B} + BC \\ &= \bar{A}\bar{B} + \bar{A}C + BC \quad \because B\bar{B} = 0 \end{aligned}$$

Review Questions

1. Define switching function.
2. Define literal, product term and sum term.
3. Explain sum of product form.
4. What do you mean by canonical SOP and POS forms.
5. Explain how to convert SOP or POS expressions in their canonical forms.
6. What do you mean by minterms and maxterms ?

2.2 Karnaugh (K-Map) Minimization

In the previous section we have seen that for simplification of Boolean expressions by Boolean algebra we need better understanding of Boolean laws, rules and theorems. During the process of simplification we have to predict each successive step. For these reasons, we can never be absolutely certain that an expression simplified by Boolean algebra alone is the simplest possible expression. On the other hand, the map method gives us a systematic approach for simplifying a Boolean expression. The map method, first proposed by Veitch and modified by Karnaugh, hence it is known as the **Veitch diagram** or the **Karnaugh map**.

2.2.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps

The basis of this method is a graphical chart known as Karnaugh map (K-map). It contains boxes called cells. Each of the cell represents one of the 2^n possible products that can be formed from n variables. Thus, a 2-variable map contains $2^2 = 4$ cells, a 3-variable map contains $2^3 = 8$ cells and so forth. Fig. 2.2.1 shows outlines of 1, 2, 3 and 4-variable maps.

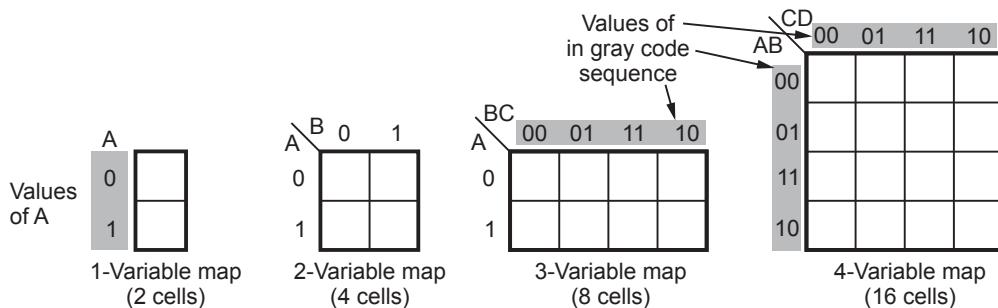


Fig. 2.2.1 Outlines of 1, 2, 3 and 4-variable Karnaugh maps

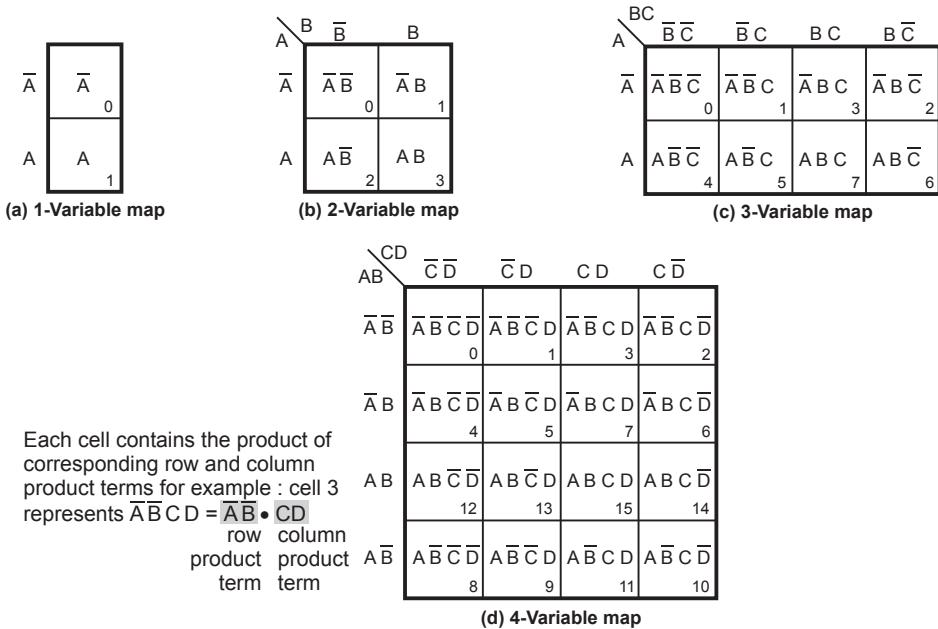


Fig. 2.2.2 1, 2, 3 and 4-variable maps with product terms

Product terms are assigned to the cells of a Karnaugh map by labeling each row and each column of the map with a variable, with its complement, or with a combination of variables and complements. The product term corresponding to a given cell is then the product of all variable in the row and column where the cell is located. Fig. 2.2.2 shows the way to label the rows and columns of a 1, 2, 3 and 4-variable maps and the product terms corresponding to each cell.

It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complemented or to an uncomplemented form). For example, in Fig. 2.2.2 (b) the only change that occurs in moving along the bottom row from $\bar{A}\bar{B}$ to AB is the change from \bar{B} to B . Similarly, the only change that occurs in moving down the right column from $\bar{A}\bar{B}$ to AB is the change from \bar{A} to A . Irrespective of number of

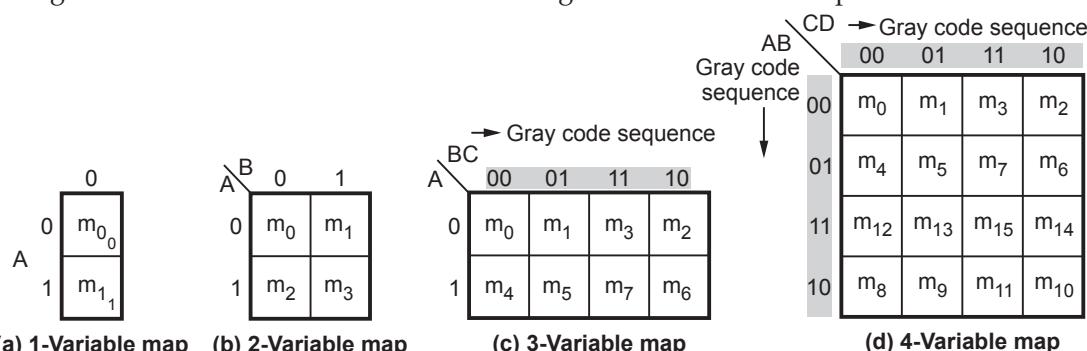


Fig. 2.2.3 Another way to represent 1, 2, 3 and 4-variable maps for SOP expressions

variables the labels along each row and column must conform to the single-change rule. We know that the gray code has same properties (only one variable change when we proceed to next number or previous number) hence gray code is used to label the rows and columns of K-map as shown in Fig. 2.2.3.

The Fig. 2.2.3 shows label of the rows and columns of a 1, 2, 3 and 4-variable maps using gray code and the product terms corresponding to each cell. Here, instead of writing actual product terms, corresponding shorthand minterm notations are written in the cell and row and columns are marked with gray code instead of variables.

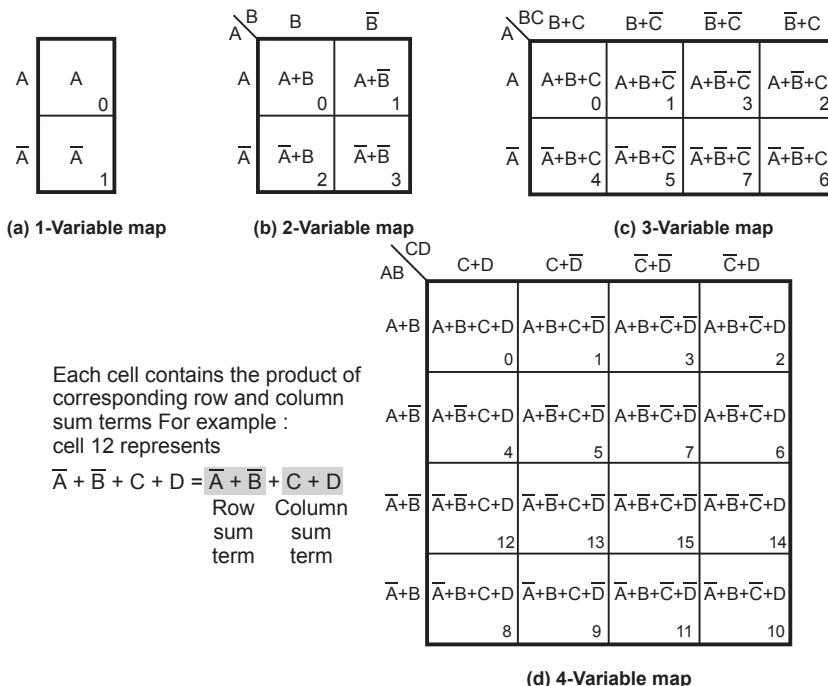


Fig. 2.2.4 1, 2, 3 and 4-variable maps with sum terms

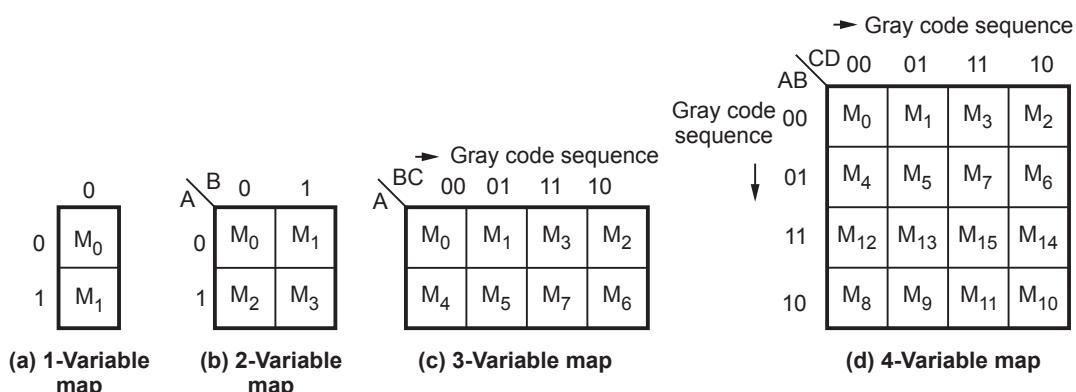


Fig. 2.2.5 Another way to represent 1, 2, 3 and 4-variable map for POS expressions

In case of POS expressions we assign maxterms (sum terms) to the cells of a Karnaugh map. The Fig. 2.2.4 shows the way to label the rows and columns of a 1, 2, 3 and 4-variable maps and the sum terms corresponding to each cell. The Fig. 2.2.4 shows label of the rows and columns of 1, 2, 3 and 4-variable maps using gray code and the sum terms corresponding to each cell. Here, instead of writing actual sum terms, corresponding shorthand maxterm notations are written in the cell and row and columns are marked with gray code instead of variables.

2.2.2 Plotting a Karnaugh Map

We know that logic function can be represented in various forms such as truth table, SOP Boolean expression and POS Boolean expression. In this section we will see the procedures to plot the given logic function in ansy form on the Karnaugh map.

2.2.2.1 Representation of Truth Table on Karnaugh Map

Cell : The smallest unit of a Karnaugh map, corresponding to one line of a truth table. The input variables are the cell's co-ordinates and the output variable is the cell's contents.

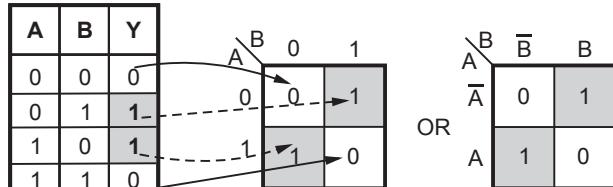


Fig. 2.2.6 (a) Representation of 2-variable truth table on K-map

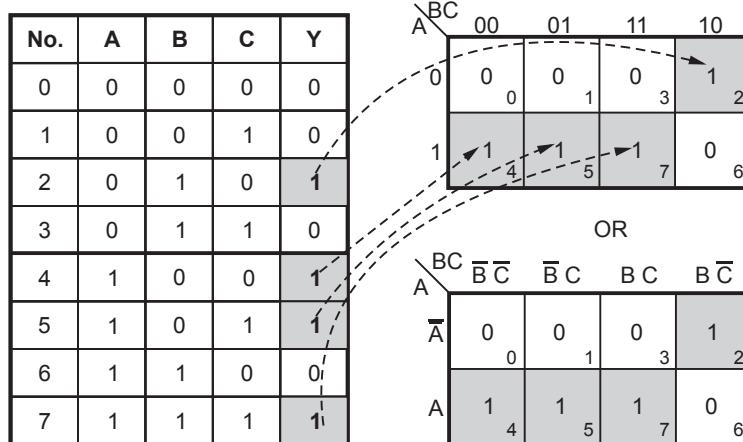
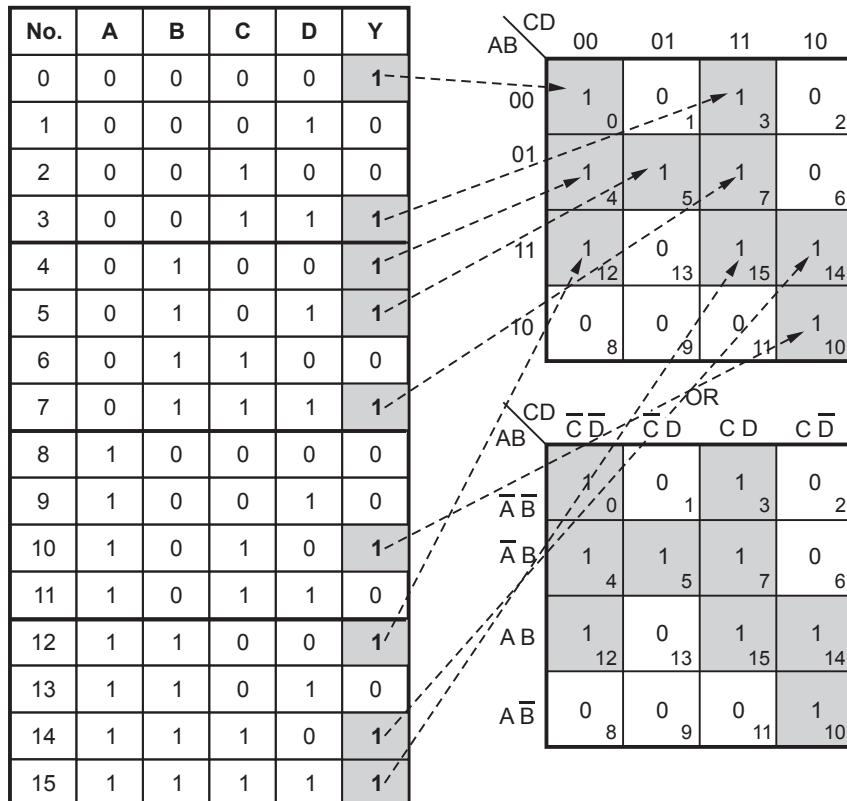


Fig. 2.2.6 (b) Representation of 3-variable truth table on K-map

Fig. 2.2.6 shows K-maps plotted from truth tables with 2, 3 and 4-variables. Looking at the Fig. 2.2.6 we can easily notice that the terms which are having output 1, have the corresponding cells marked with 1s. The other cells are marked with zeros.

Note The student can verify the data in each cell by checking the data in the column Y for particular row number and the data in the same cell number in the K-map.



(c) Representation of 4-variable truth table on K-map

Fig. 2.2.6 Plotting truth table on K-map

2.2.2.2 Representing Standard SOP on K-Map

A Boolean expression in the sum of products form can be plotted on the Karnaugh map by placing a 1 in each cell corresponding to a term (minterm) in the sum of products expression. Remaining cells are filled with zeros. This is illustrated in the following examples.

Illustrative Examples

Example 2.2.1 Plot Boolean expression $Y = AB\bar{C} + ABC + \bar{A}\bar{B}C$ on the Karnaugh map.

Solution : The expression has 3-variables and hence it can be plotted using 3-variable as in Fig. 2.2.7.

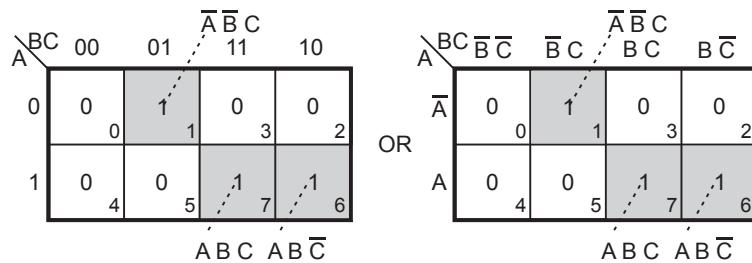


Fig. 2.2.7

Example 2.2.2 Plot Boolean expression

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

Solution : The expression has 4-variables and hence it can be plotted using 4-variable map as shown in Fig. 2.2.8.

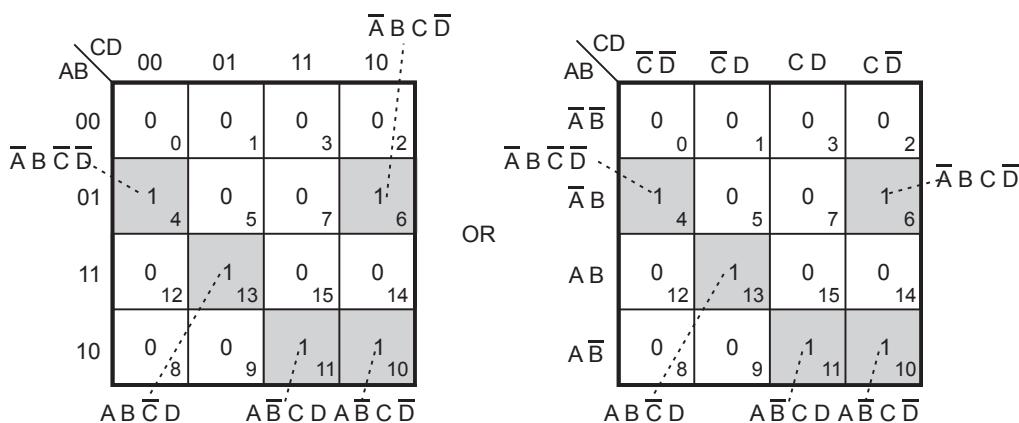


Fig. 1.2.8

2.2.2.3 Representing Standard POS on K-Map

- A Boolean expression in the product of sums can be plotted on the Karnaugh map by placing a 0 in each cell corresponding to a term (maxterm) in the expression. Remaining cells are filled with ones. This is illustrated in the following examples.

Illustrative Examples**Example 2.2.3** Plot Boolean expression $Y = (A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)(A + B + \bar{C})$ on the Karnaugh map.

Solution : The expression has 3-variables and hence it can be plotted using 3-variable map as shown in Fig. 2.2.9.

$$(A + \bar{B} + C) = M_2, (A + \bar{B} + \bar{C}) = M_3, (\bar{A} + \bar{B} + C) = M_6, (A + B + \bar{C}) = M_1$$

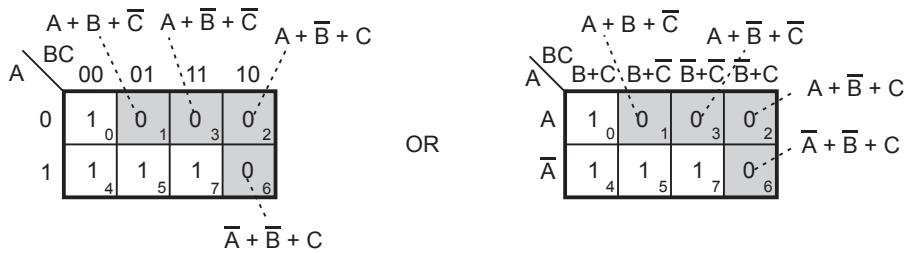


Fig. 2.2.9

Example 2.2.4 Plot Boolean expression.

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

Solution : The expression has 4-variables and hence it can be plotted using 4-variable map as shown in Fig. 2.2.10.

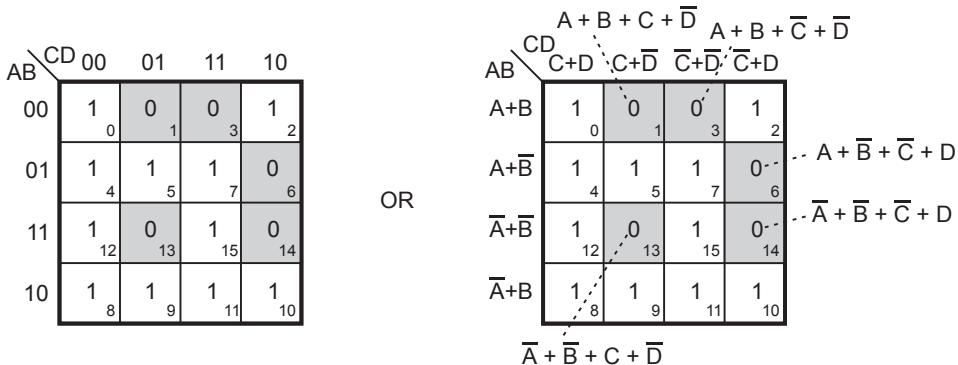


Fig. 2.2.10

$$(A + B + C + \bar{D}) = M_1, (A + \bar{B} + \bar{C} + D) = M_6, (A + B + \bar{C} + \bar{D}) = M_3,$$

$$(\bar{A} + \bar{B} + C + \bar{D}) = M_{13}, (\bar{A} + \bar{B} + \bar{C} + D) = M_{14}$$

2.2.3 Grouping Cells for Simplification

In the last section we have seen representation of Boolean function on the Karnaugh map. We have also seen that minterms are marked by 1s and maxterms are marked by 0s. Once the Boolean function is plotted on the Karnaugh map we have to use grouping technique to simplify the Boolean function. The grouping is nothing but combining terms in adjacent cells. Two cells are said to be adjacent if they conform the single change rule. i.e. there is only one variable difference between co-ordinates of two cells. For example, the cells for minterms ABC and $\bar{A}BC$ are adjacent. The Fig. 2.2.11 shows the adjacent cells. The simplification is achieved by grouping adjacent 1s or 0s in groups of 2^i , where $i = 1, 2, \dots, n$ and n is the number of variables. When adjacent 1s are grouped then we get result in the sum of products form; otherwise we get result in the product of sums form. Let us see the various grouping rules.

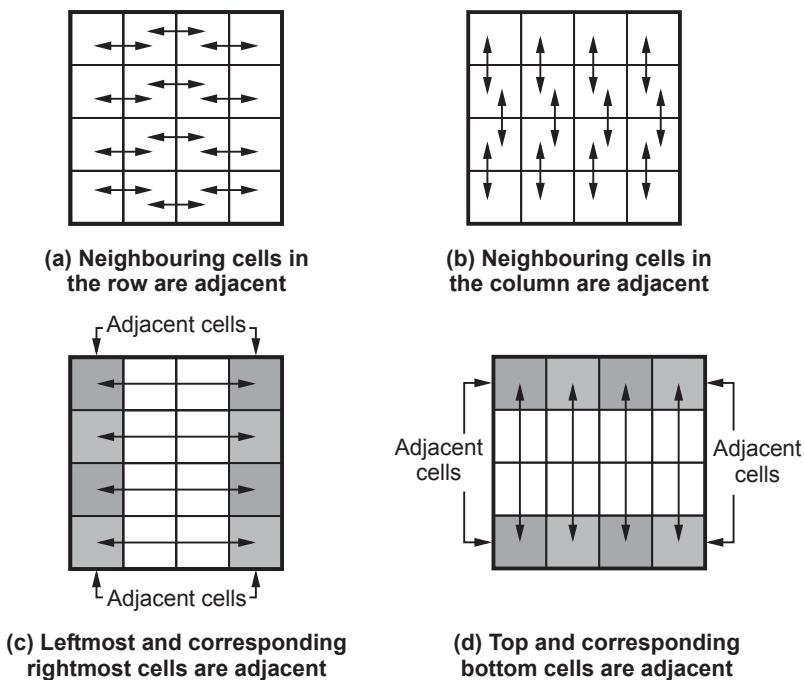


Fig. 2.2.11 Adjacent cells

2.2.3.1 Grouping Two Adjacent Ones (Pair)

Fig. 2.2.12 (a) shows the Karnaugh map for a particular three variable truth table. This K-map contains a pair of 1s that are horizontally adjacent to each other; the first represents $\bar{A} \bar{B} C$ and the second represents $\bar{A} B C$. Note that in these two terms only the B variable appears in both normal and complemented form (\bar{A} and C remain unchanged). Thus these two terms can be combined to give a resultant that eliminates the B variable since it appears in both uncomplemented and complemented form. This is easily proved as follows :

$$\begin{aligned}
 Y &= \bar{A} \bar{B} C + \bar{A} B C \\
 &= \bar{A} C (\bar{B} + B) \quad \text{Rule 6 : } [\bar{A} + A = 1] \\
 &= \bar{A} C
 \end{aligned}$$

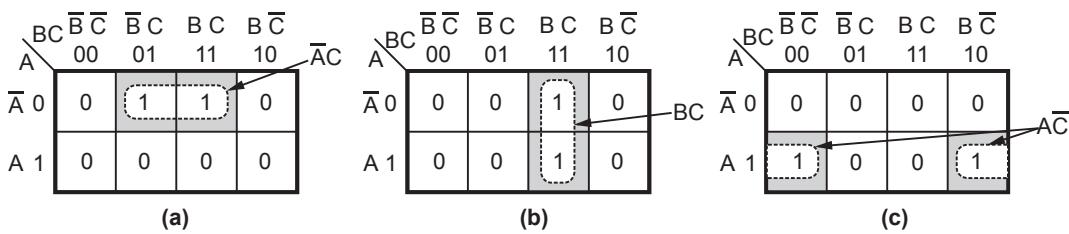


Fig. 2.2.12

This same principle holds true for any pair of vertically or horizontally adjacent 1s. Fig. 2.2.12 (b) shows an example of two vertically adjacent 1s. These two can be combined to eliminate A variable since it appears in both its uncomplemented and complemented forms. This gives result

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

In a Karnaugh map the corresponding cells in the leftmost column and rightmost column are considered to be adjacent. Thus, the two 1s in these columns with a common row can be combined to eliminate one variable. This is illustrated in Fig. 2.2.12 (c).

Here variable B has appeared in both its complemented and uncomplemented forms and hence eliminated as follows :

$$\begin{aligned} Y &= A \overline{B} \overline{C} + A B \overline{C} \\ &= A \overline{C} (\overline{B} + B) \\ &= A \overline{C} \end{aligned}$$

Rule 6 : $\overline{A} + A = 1$

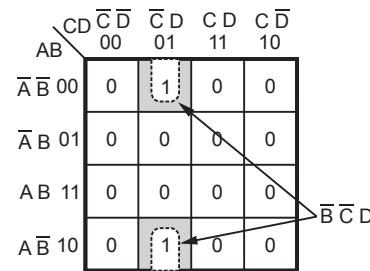


Fig. 2.2.12 (d)

$$\begin{aligned} Y &= \overline{A} \overline{B} \overline{C} D + A \overline{B} \overline{C} D \\ &= \overline{B} \overline{C} D (\overline{A} + A) \\ &= \overline{B} \overline{C} D \end{aligned}$$

Fig. 2.2.12 (e) shows a Karnaugh map that has two overlapping pairs of 1s. This shows that we can share one term between two pairs.

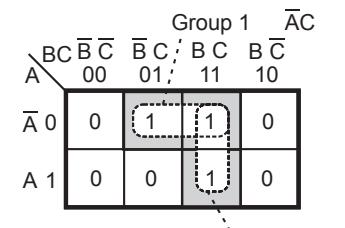


Fig. 2.2.12 (e)

$$\begin{aligned} Y &= \overline{A} \overline{B} C + \overline{A} B C + A B C \\ &= \overline{A} \overline{B} C + \overline{A} B C + \overline{A} B C + A B C \\ &= \overline{A} C (\overline{B} + B) + B C (\overline{A} + A) \\ &= \overline{A} C + B C \end{aligned}$$

Rule 5 : $[A + A = A]$

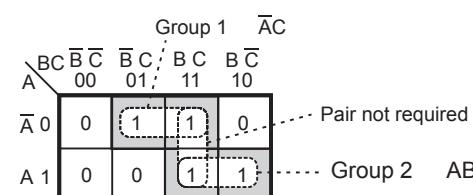


Fig. 2.2.12 (f) Examples of combining pairs of adjacent ones

$$\begin{aligned}
 Y &= \overline{A} \overline{B} C + \overline{A} B C + A B C + A B \overline{C} \\
 &= \overline{A} C (\overline{B} + B) + A B (C + \overline{C}) \\
 &= \overline{A} C + A B
 \end{aligned}$$

Rule 6 : $[\overline{A} + A = 1]$

A pair is a group of two adjacent cells in a Karnaugh map. It cancels one variable in a K-map simplification.

2.2.3.2 Grouping Four Adjacent Ones (Quad)

In a Karnaugh map we can group four adjacent 1s.

The resultant group is called Quad. Fig. 2.2.13 shows several examples of quads.

Fig. 2.2.13 (a) shows the four 1s are horizontally adjacent and Fig. 2.2.13 (b) shows they are vertically adjacent.

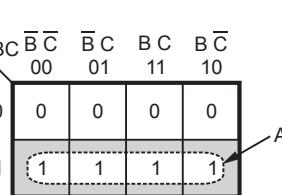
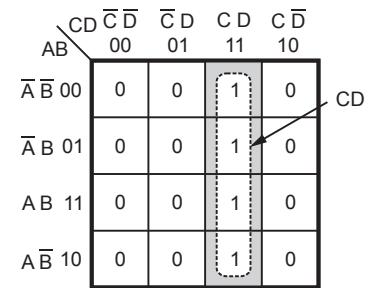
(a) $Y = A$ (b) $Y = CD$

Fig. 2.2.13

A K-map in Fig. 2.2.13 (c) contains four 1s in a square and they are considered adjacent to each other. The four 1s in Fig. 2.2.13 (d) are also adjacent, as are those in Fig. 2.2.13 (e) because, as mentioned earlier, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

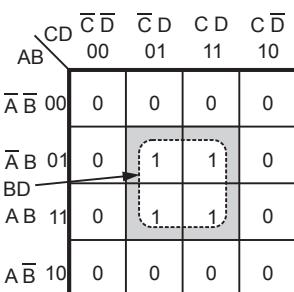
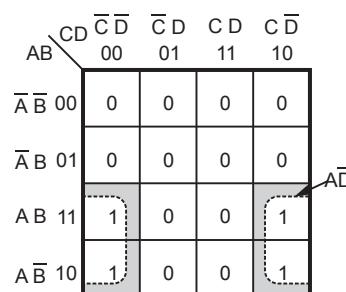
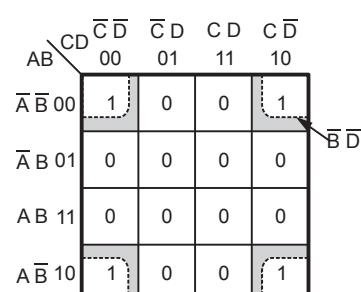
(c) $Y = BD$ (d) $Y = AD$ (e) $Y = \overline{BD}$

Fig. 2.2.13

From the above Karnaugh maps we can easily notice that when a quad is combined two variables are eliminated. For example, in Fig. 2.2.13 (c) we have following terms with 4 variables :

$$\begin{aligned}
 Y &= \overline{A} B \overline{C} D + \overline{A} B C D + A B \overline{C} D + A B C D \\
 &= \overline{A} B D (\overline{C} + C) + A B D (\overline{C} + C) = \overline{A} B D + A B D
 \end{aligned}$$

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

Fig. 2.2.13 (f) shows overlapping groups. As mentioned earlier one term can be shared between two or more groups.

Quad is a group of four adjacent cells in a Karnaugh map. It cancels two variables in a K-map simplification.

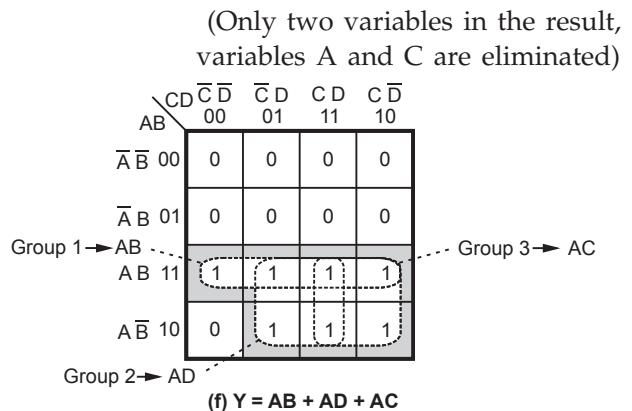
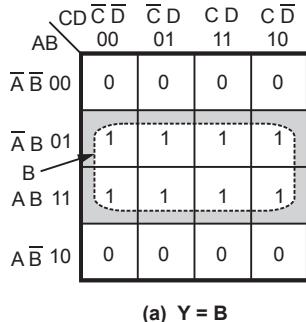


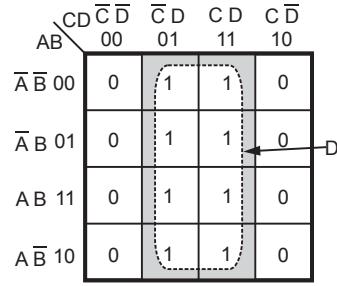
Fig. 2.2.13 Examples of combining quads of adjacent ones

2.2.3.3 Grouping Eight Adjacent Ones (Octet)

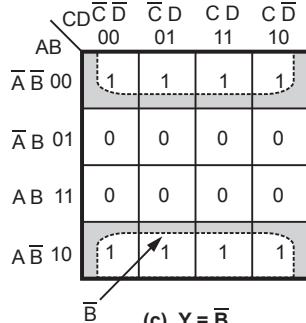
In a Karnaugh map we can group eight adjacent 1s. The resultant group is called as octet. Fig. 2.2.14 shows several examples of octets. Fig. 2.2.14 (a) shows the eight 1s are horizontally adjacent and Fig. 2.2.14 (b) shows they are vertically adjacent.



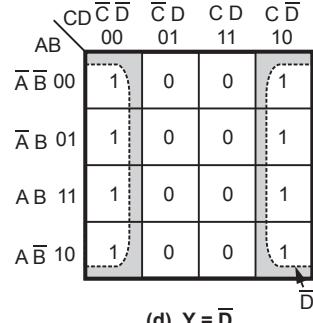
(a) $Y = B$



(b) $Y = D$



(c) $Y = \bar{B}$



(d) $Y = \bar{D}$

Fig. 2.2.14 Examples of combining octets of adjacent ones

From the Fig. 2.2.14 we can easily observe that when an octet is combined in a four variable map, three of the four variables are eliminated because only one variable

remains unchanged. For example, in K-map shown in Fig. 2.2.14 (a) we have following terms :

$$\begin{aligned}
 Y &= \overline{A} B \bar{C} \bar{D} + \overline{A} B \bar{C} D + \overline{A} B C D + \overline{A} B C \bar{D} \\
 &\quad + A B \bar{C} \bar{D} + A B \bar{C} D + A B C D + A B C \bar{D} \\
 &= \overline{A} B \bar{C} (\bar{D} + D) + \overline{A} B C (D + \bar{D}) + A B \bar{C} (\bar{D} + D) + A B C (D + \bar{D}) \\
 &= \overline{A} B \bar{C} + \overline{A} B C + A B \bar{C} + A B C = \overline{A} B (\bar{C} + C) + A B (\bar{C} + C) \\
 &= \overline{A} B + A B = B (\overline{A} + A) = B
 \end{aligned}$$

Octet is a group of eight adjacent cells in a Karnaugh map. It cancels three variables in a K-map simplifications.

2.2.4 Illegal Grouping

The Fig. 2.2.15 shows the examples of illegal grouping of cells.

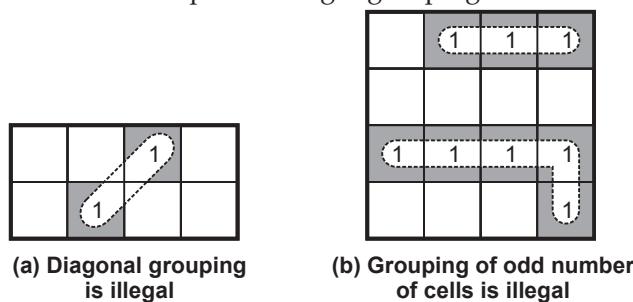


Fig. 2.2.15

2.3 Simplification of SOP Expression

SPPU : Dec.-09, May-13,19

We have seen how combination of pairs, quads and octets on a Karnaugh map can be used to obtain a simplified expression. A pair of 1s eliminates one variable, a quad of 1s eliminates two variables and an octet of 1s eliminates three variables. In general, when a variable appears in both complemented and uncomplemented form within a group, that variable is eliminated from the resultant expression. Variables that are same in all with the group must appear in the final expression. Each group gives us a product term and summation of all product term gives us a Boolean expression. Therefore, we can say that, each product term implies the function and, hence is an implicant of the function. All the implicants of a function determined using a Karnaugh map are the prime implicants.

From the above discussion we can outline generalized procedure to simplify Boolean expressions as follows :

1. Plot the K-map and place 1s in those cells corresponding to the 1s in the truth table or sum of product expression. Place 0s in other cells.

2. Check the K-map for adjacent 1s and encircle those 1s which are not adjacent to any other 1s. These are called isolated 1s.
3. Check for those 1s which are adjacent to only one other 1 and encircle such pairs.
4. Check for quads and octets of adjacent 1s even if it contains some 1s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 1s that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

Illustrative Examples

Example 2.3.1 Minimize the expression $Y = \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}$.

Solution : Step 1 : Fig 2.3.1 (a) shows the K-map for three variables and it is plotted according to the given expression.

Step 2 : There are no isolated 1s.

Step 3 : 1 in the cell 3 is adjacent only to 1 in the cell 1. This pair is combined and referred to as group 1.

Step 4 : There is no octet, but there is a quad. Cells 0, 1, 4 and 5 form a quad. This quad is combined and referred to as group 2.

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 B variable is eliminated and in group 2 variables A and C are eliminated and we get,

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

Example 2.3.2 Minimize the expression

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

Solution : Step 1 : Fig. 2.3.2 (a) shows the K-map for four variables and it is plotted according to the given expression.

	BC $\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
AC	00	01	11	10
$\bar{A}0$	1 0	1 1	1 3	0 2
A1	1 4	1 5	0 7	0 6

Fig. 2.3.1 (a)

	BC $\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
AC	00	01	11	10
$\bar{A}0$	1	1	1	0
A1	1	1	0	0

Fig. 2.3.1 (b)

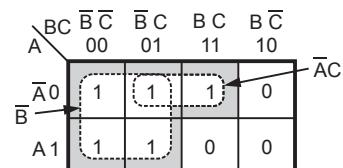


Fig. 2.3.1 (c)

	CD	$\bar{C}D$	$\bar{C}D$	CD	$C\bar{D}$
	00	01	11	11	10
$\bar{A}B$	00	0	0	0	1
$\bar{A}B$	01	1	1	0	0
AB	11	1	1	0	0
A \bar{B}	10	0	1	0	0

Fig. 2.3.2 (a)

	CD	$\bar{C}D$	$\bar{C}D$	CD	$C\bar{D}$
	00	01	11	11	10
$\bar{A}B$	00	0	0	0	1
$\bar{A}B$	01	1	1	0	0
AB	11	1	1	0	0
A \bar{B}	10	0	1	0	0

Fig. 2.3.2 (b)

	CD	$\bar{C}D$	$\bar{C}D$	CD	$C\bar{D}$
	00	01	11	11	10
$\bar{A}B$	00	0	0	0	1
$\bar{A}B$	01	1	1	0	0
AB	11	1	1	0	0
A \bar{B}	10	0	1	0	0

Fig. 2.3.2 (c)

	CD	$\bar{C}D$	$\bar{C}D$	CD	$C\bar{D}$
	00	01	11	11	10
$\bar{A}B$	00	0	0	0	1
$\bar{A}B$	01	1	1	0	0
AB	11	1	1	0	0
A \bar{B}	10	0	1	0	0

$\bar{A}\bar{B}C\bar{D}$

$\bar{A}\bar{C}D$

Fig. 2.3.2 (d)

Step 3 : 1 in the cell 9 is adjacent only to 1 in the cell 13. This pair is combined and referred to as group 2.

Step 4 : There is no octet, but there is quad cells 4, 5, 12 and 13 form a quad. This quad is combined and referred to as group 3.

Step 5 : All 1s have already grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 variable is not eliminated. In group 2 variable B is eliminated and in group 3 variables A and D are eliminated and we get,

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

Example 2.3.3 Reduce the following four variable function to its minimum sum of products form :

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

Solution : Step 1 : Fig. 2.3.3 (a) shows the K-map for four variables and it is plotted according to the given expression.

Step 2 : There are no isolated 1s.

Step 3 : There are no such 1s which are adjacent to only one other 1.

Step 4 : There are three quads formed by cells 0, 2, 8, 10, cells 8, 10, 12, 14 and cells 2, 3, 10, 11. These quads are combined and referred to as group 1, group 2 and group 3 respectively.

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 variables A and C are eliminated, in group 2 variables B and C are eliminated and in group 3 variables A and D are eliminated and we get,

$$Y = \overline{B} \overline{D} + A\overline{D} + \overline{B}C$$

Example 2.3.4 Reduce the following function to its minimum sum of products form :

$$Y = \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{ABC}\overline{D} + AB\overline{C}\overline{D} + A\overline{B}CD + ABCD + A\overline{B}CD$$

Solution : Step 1 : Fig. 2.3.4 (a) shows the K-map for four variables and it is plotted according to the given expression.

Step 2 : There are no isolated 1s.

Step 3 : The 1 in the cell 1 is adjacent only to 1 in the cell 5, the 1 in the cell 6 is adjacent only to the 1 in the cell 7, the 1 in the cell 12 is adjacent only to the 1 in the cell 13 and the 1 in the cell 11 is adjacent only to the 1 in the cell 15. These pairs are combined and referred to as group 1-4 respectively.

Step 4 : There is no octet, but there is a quad. However, all 1s in the quad have already been grouped. Therefore this quad is ignored.

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 variable B is eliminated. Similarly, in group

	CD	$\overline{C}D$	$\overline{C}D$	CD	$C\overline{D}$
AB	00	01	11	10	
$\overline{A}B$	00	01	11	10	
	1 ₀	0 ₁	1 ₃	1 ₂	
	0 ₄	0 ₅	0 ₇	0 ₆	
	1 ₁₂	0 ₁₃	0 ₁₅	1 ₁₄	
	1 ₈	0 ₉	1 ₁₁	1 ₁₀	

Fig. 2.3.3 (a)

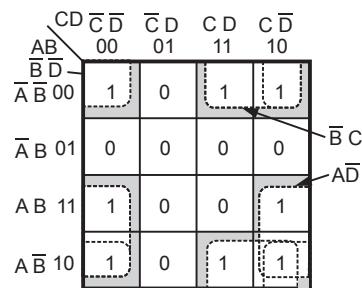


Fig. 2.3.3 (b)

	CD	$\overline{C}D$	$\overline{C}D$	CD	$C\overline{D}$
AB	00	01	11	10	
$\overline{A}B$	00	01	11	10	
	0 ₀	1 ₁	0 ₃	0 ₂	
	0 ₄	1 ₅	1 ₇	1 ₆	
	1 ₁₂	1 ₁₃	1 ₁₅	0 ₁₄	
	0 ₈	0 ₉	1 ₁₁	0 ₁₀	

Fig. 2.3.4 (a)

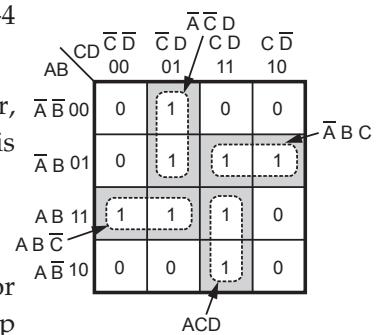


Fig. 2.3.4 (b)

2-4 variables D, D and B are eliminated one in each group. We finally get minimum sum of products form as,

$$Y = \overline{A} \overline{C} D + \overline{A} BC + ABC\bar{C} + ACD$$

Example 2.3.5 Simplify the logic function specified by the truth Table 2.3.1 using the Karnaugh map method. Y is the output variable, and A, B and C are the input variables.

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

Table 2.3.1

Solution : **Step 1 :** Fig. 2.3.5 (a) shows the K-map for three variables and it is plotted according to given truth table.

Step 2 : There are no isolated 1s.

Step 3 : The 1 in the cell 0 is adjacent only to 1 in the cell 4 and the 1 in the cell 3 is adjacent only to 1 in the cell 7. These two pairs are grouped and referred to as group 1 and group 2.

Step 4 : There is no octet and quad.

Step 5 : All 1s have already been grouped.

Step 6 : In group 1 and group 2 variable A is eliminated and we get,

$$Y = \overline{B} \overline{C} + BC$$



Fig. 2.3.5 (a)

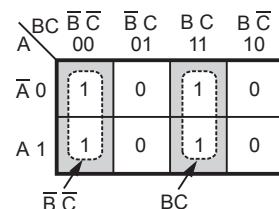


Fig. 2.3.5 (b)

Example 2.3.6 Reduce the following function using Karnaugh map technique and implement using basic gates

$$f(A, B, C, D) = \overline{AB}D + AB\overline{C}\overline{D} + \overline{A}BD + ABC\overline{D}$$

Solution : The given function is not in the standard sum of products form. It is converted into standard SOP form as given below.

$$\begin{aligned}
 f(A, B, C, D) &= \overline{ABD} + ABC\bar{D} + \overline{ABD} + ABC\bar{D} \\
 &= \overline{ABD}(C + \bar{C}) + ABC\bar{D} + \overline{ABD}(C + \bar{C}) + ABC\bar{D} \\
 &= \overline{ABCD} + \overline{ABC}\bar{D} + ABC\bar{D} + \overline{ABCD} + \overline{ABC}\bar{D} + ABC\bar{D}
 \end{aligned}$$

Step 1 : Fig. 2.3.6 (a) shows the K-map for four variables and it is plotted according to expression in standard SOP form.

	CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
AB	00	01	11	10	
\overline{AB} 00	0	1	1	0	
\overline{AB} 01	0	1	1	0	
AB 11	1	0	0	1	
$A\bar{B}$ 10	0	0	0	0	

Fig. 2.3.6 (a)

Step 2 : There are no isolated 1s.

	CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
AB	00	01	11	10	
\overline{AB} 00	0	1	1	0	
\overline{AB} 01	0	1	1	0	
AB 11	1	0	0	1	
$A\bar{B}$ 10	0	0	0	0	

Fig. 2.3.6 (b)

Step 3 : The 1 in the cell 12 is adjacent only to the 1 in the cell 14. This pair is combined and referred to as group 1.

Step 4 : There is a quad. Cells 1, 3, 5 and 7 form a quad. This quad is referred to as group 2.

Step 5 : All 1s have already been grouped.

Step 6 : In group 1 variable C is eliminated and in group 2 variables B and C are eliminated. We get simplified equation as,

$$Y = ABD + AD$$

Example 2.3.7 Reduce the following function using K-map technique.

$$f(A, B, C, D) = \sum m(0, 1, 4, 8, 9, 10).$$

Solution : **Step 1 :** Fig. 2.3.7 (a) shows the K-map for four variables and it is plotted according to given minterms.

Step 2 : There are no isolated 1s.

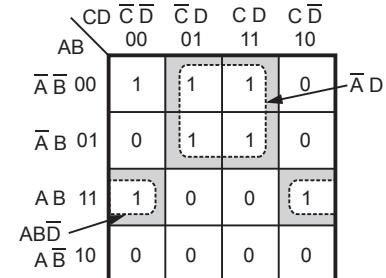


Fig. 2.3.6 (c)

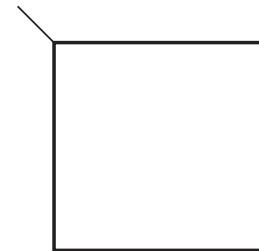


Fig. 2.3.7 (a)

Step 3 : Cell 4 is adjacent only to cell 0 and cell 10 is adjacent only to cell 8. These two pairs are combined and referred to as group 1 and group 2 respectively.

Step 4 : There is a quad. Cells 0, 1, 8 and 9 form a quad. This quad is referred to as group 3.

Step 5 : All 1s have already been grouped.

Step 6 : In group 1, B and in group 2, C are eliminated respectively. In group 3, A and D are eliminated and finally we get

$$f(A, B, C, D) = \overline{A}\overline{C}D + A\overline{B}\overline{D} + \overline{B}\overline{C}$$

	CD	$\overline{C}D$	$\overline{C}D$	C D	C \overline{D}
AB	00	01	11	10	
$\overline{A}\overline{B}$	00	1	1	0	0
$\overline{A}B$	01	1	0	0	0
A \overline{B}	11	0	0	0	0
A B	10	1	0	1	

Fig. 2.3.7 (b)

	CD	$\overline{C}D$	$\overline{C}D$	$\overline{B}\overline{C}$	C D	C \overline{D}
AB	00	01	11	10		
$\overline{A}\overline{B}$	00	1	1	0	0	
$\overline{A}C\overline{D}$	01	1	0	0	0	
$\overline{A}B$	11	0	0	0	0	
A \overline{B}	10	1	1	0	1	

Fig. 2.3.7 (c)

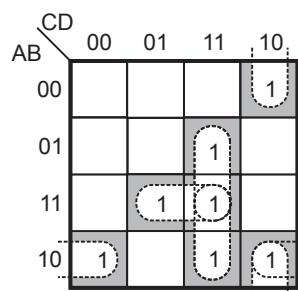
Example 2.3.8 Solve the following equations using corresponding minimization techniques :

- i) $Z = f(A, B, C, D) = \Sigma(2, 7, 8, 10, 11, 13, 15)$
- ii) $Z = f(A, B, C, D) = \Sigma(0, 3, 4, 9, 10, 12, 14)$

SPPU : May-19, Marks 12

Solution :

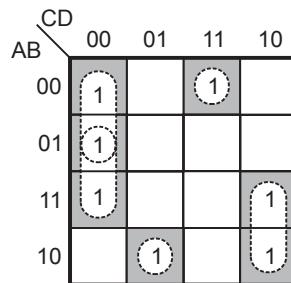
i)



$$F = ABD + A\overline{B}\overline{D} + BCD + A\overline{B}D + \overline{B}CD$$

Fig. 2.3.8

ii)



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

Fig. 2.3.9

Examples for Practice

Example 2.3.9 : Simplify following logical expression using Karnaugh maps

$$Y = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + A \overline{B} \overline{C} + \overline{A} \overline{B} C + A B \overline{C} \quad \text{Ans. : } Y = \overline{A} \overline{B} + \overline{C}$$

Example 2.3.10 : Simplify the following function

$$f_1(A, B, C, D) = \sum m(0, 3, 5, 6, 9, 10, 12, 15)$$

SPPU : May-13, Marks 4

$$\text{Ans. : } (A \oplus B) \odot (C \oplus D)$$

Example 2.3.11 : Simplify the following function

$$f_3(A, B, C, D) = \sum m(0, 1, 2, 3, 11, 12, 14)$$

SPPU : May-13, Marks 4

$$\text{Ans. : } \overline{A} \overline{B} + A B \overline{D} + \overline{B} C D$$

Example 2.3.12 : Simplify the following using K-map.

$$X = A'B + A'B'C + ABC' + ABC$$

$$\text{Ans. : } X = \overline{A}C + B$$

Example 2.3.13 : Solve the following using minimization technique

$$Z = f(A, B, C, D) = \sum (0, 2, 4, 7, 11, 13, 15) \quad \text{SPPU : Dec.-09, Marks 5}$$

$$\text{Ans. : } Z = \overline{A} \overline{B} \overline{D} + \overline{A} \overline{C} \overline{D} + B C D + A B D + A C D$$

2.3.1 Essential Prime Implicants

After grouping the cells, the sum terms which appear in the K-map are called prime implicants groups. It is observed that some cells may appear in only one prime implicants group; while other cells may appear in more than one prime implicants group. In Fig. 2.3.7 (c), cells 1, 4, 9 and 10 appear in only one prime implicants group. These cells are called **essential cells** and corresponding prime implicants are called **essential prime implicants**.

2.3.2 Incompletely Specified Functions (Don't Care Terms)

In some logic circuits, certain input conditions never occur, therefore the corresponding output never appears. In such cases the output level is not defined, it can be either HIGH or LOW. These output levels are indicated by 'X' or 'd' in the truth tables and are called **don't care outputs** or **don't care conditions** or **incompletely specified functions**. Let us see the output levels in the truth table as shown in the Table 2.3.2. Here outputs are defined for input conditions from 0 0 0 to 1 0 1. For remaining two conditions of input, output is not defined, hence these are called don't care conditions for this truth table.

A circuit designer is free to make the output for any "don't care" condition either a '0' or a '1' in order to produce the simplest output expression.

2.3.2.1 Describing Incomplete Boolean Function

We know that we describe the Boolean function using either a minterm canonical formula or a maxterm canonical formula. In order to obtain similar-type expressions for incomplete Boolean functions we use additional term to specify don't care conditions in the original expression. This is illustrated in the following examples.

In expression,

$$f(A, B, C) = \sum m(0, 2, 4) + d(1, 5)$$

minterms are 0, 2 and 4. The additional term $d(1, 5)$ is introduced to specify the don't care conditions. This term specifies that outputs for minterms 1 and 5 are not specified and hence these are don't care conditions. Letter d is used to indicate don't care conditions in the expression.

The above expression indicates how to represent don't care conditions in the minterm canonical formula. In the similar manner, we can specify the don't care conditions in the maxterm canonical formula. For example,

$$f(A, B, C) = \prod M(2, 5, 7) + d(1, 3)$$

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

Table 2.3.2

2.3.2.2 Don't Care Conditions in Logic Design

In this section, we see the example of incompletely specified Boolean function. Let us see the logic circuit for an even parity generator for 4-bit BCD number. The Table 2.3.3 shows the truth table for even-parity generator. The truth table shows that the output for last six input conditions cannot be specified, because such input conditions does not occur when input is in the BCD form.

The Boolean function for even parity generator with 4-bit BCD input can be expressed in minterm canonical formula as,

$$f(A, B, C, D) = \sum m(1, 2, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$$

2.3.2.3 Minimization of Incompletely Specified Functions

A circuit designer is free to make the output for any don't care condition either a '0' or '1' in order to produce the simplest output expression. Consider a truth table shown in Table 2.3.4. The K-map for this truth table is shown in

Fig. 2.3.5 with 'x' placed in the ABC and $A\bar{B}\bar{C}$ cells.

It is not always advisable to put don't cares as 1s. This is illustrated in Fig. 2.3.10 (b). Here, the don't care output for cell ABC is taken as 1 to form a quad and don't care output for cell $A\bar{B}\bar{C}$ is taken as 0, since it is not helping any way to reduce an expression. Using don't care conditions in this way we get the simplified Boolean expression as

$$Y = C$$

From the above discussion we can realize that it is important to decide which don't cares to change to 0 and which to 1 to produce the best K-map grouping (i.e. the simplest expression). Now, we will see more examples to provide practice in dealing with "don't care" conditions.

A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
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	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	-

Table 2.3.3 Truth table for even parity generator with 4-bit BCD input

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

Table 2.3.4

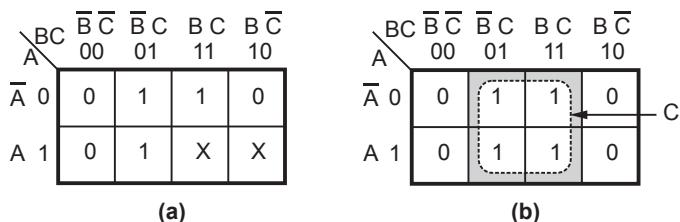


Fig. 2.3.10 Use of don't care conditions

Illustrative Examples

Example 2.3.14 Find the reduced SOP form of the following function.

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 4).$$

Solution :

		$\bar{C} \bar{D}$	$\bar{C} D$	$C \bar{D}$	$C D$
		00	01	11	10
$A \bar{B}$	00	X 0	1 1	1 3	X 2
	01	X 4	0 5	1 7	0 6
$A B$	11	0 12	0 13	1 15	0 14
	10	0 8	0 9	1 11	0 10

(a)

		$\bar{C} \bar{D}$	$\bar{C} D$	$C \bar{D}$	$C D$
		00	01	11	10
$A \bar{B}$	00	1	1	1	1
	01	0	0	1	0
$A B$	11	0	0	1	0
	10	0	0	1	0

(b)

Fig. 2.3.11

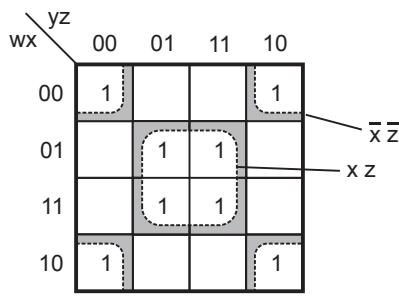
To form a quad of cells 0, 1, 2 and 3 the don't care conditions 0 and 2 are replaced by 1s. The remaining don't care condition is replaced by 0 since it is not required to form any group. With these replacements we get the simplified equation as

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

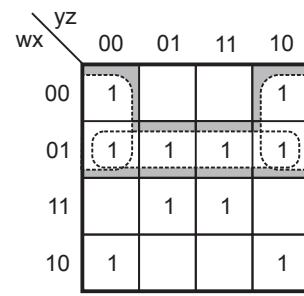
Group 1 Group 2

Example 2.3.15 Find the prime implicants for the following function and determine which are essential. $F(w, x, y, z) = \sum (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

Solution : The minterms of the given function are marked with 1's in the map of Fig. 2.3.12. The Fig 2.3.12 (a) shows two essential prime implicants. The term xz is essential because there is only one way to include minterms m_{13} and m_{15} within four adjacent squares. Similarly, there is only one way that minterms m_8 and m_{10} can be



(a) Essential prime implicants
 $\bar{x} \bar{z}$ and xz



(b) Prime implicants
 $\bar{w} x$ and $\bar{w} z$

Fig. 2.3.12

combined with four adjacent squares and this gives second term $\bar{x}\bar{z}$. The two essential prime implicants cover eight minterms. The remaining two minterms, m_4 and m_6 must be considered next.

The Fig. 2.3.12 shows the two possible ways of including remaining two minterms with prime implicants. Minterms m_4 and m_6 can be included with either $\bar{w}x$ or $\bar{w}\bar{z}$. The simplified expression is obtained from the logical sum of the two essential prime implicants and any one prime implicant that includes minterms m_4 and m_6 . Thus, there are two possible ways that the function can be expressed in the simplified form :

$$F = \bar{x}\bar{z} + xz + \bar{w}x = \bar{x}\bar{z} + xz + \bar{w}\bar{z}$$

Example 2.3.16 Express the following function as the minimal sum of products using a K-map. $f(a,b,c,d) = \Sigma(0,2,4,5,6,8,10,15) + \Sigma\phi(7,13,14)$

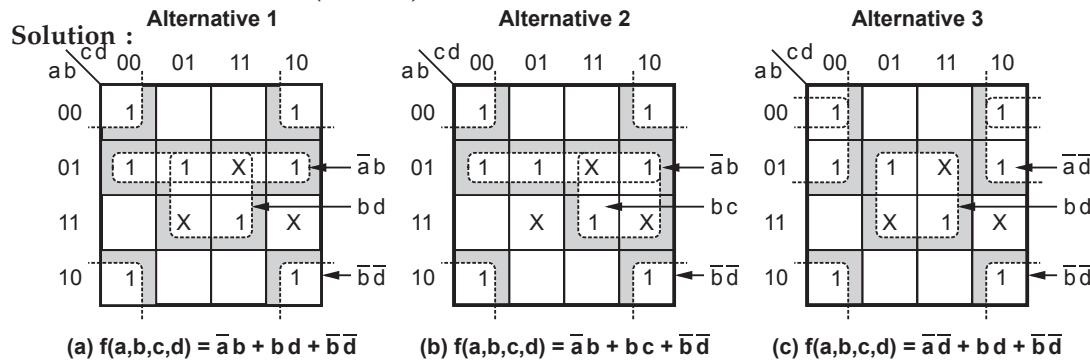


Fig. 2.3.13

As shown in Fig. 2.3.13 the given example has three solutions and all are correct. Students are expected to give any one solution.

Examples for Practice

Example 2.3.17 : Simplify the following switching function using Karnaugh map

$$F(A,B,C,D) = \Sigma(0,5,7,8,9,10,11,14,15) + \phi(1,4,13).$$

$$\text{Ans. : } \bar{B}\bar{C} + BD + AC$$

Example 2.3.18 : Determine the minimal sum of product form of

$$F(w,x,y,z) = \Sigma m(4,5,7,12,14,15) + d(3,8,10).$$

$$\text{Ans. : } \bar{w}x\bar{y} + xyz + wz$$

Review Questions

1. Give the steps for simplification of SOP expression.
2. What do you mean by essential prime implicants ?
3. What is don't care condition ?

2.4 Simplification of POS Expression

SPPU : Dec.-15

In the above discussion, we have considered the Boolean expression in sum of products form and grouped 2, 4, and 8 adjacent ones to get the simplified Boolean expression in the same form. In practice, the designer should examine both the sum of products and product of sums reductions to ascertain which is more simplified. We have already seen the representation of product of sums on the Karnaugh map. Once the expression is plotted on the K-map instead of making the groups of ones, we have to make groups of zeros. Each group of zero results a sum term and it is nothing but the **prime implicate**. The technique for using maps for POS reductions is a simple step by step process and it is similar to the one used earlier.

1. Plot the K-map and place 0s in those cells corresponding to the 0s in the truth table or maxterms in the product of sums expression.
2. Check the K-map for adjacent 0s and encircle those 0s which are not adjacent to any other 0s. These are called isolated 0s.
3. Check for those 0s which are adjacent to only one other 0 and encircle such pairs.
4. Check for quads and octets of adjacent 0s even if it contains some 0s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 0s that have not yet been grouped.
6. Form the simplified POS expression for F by taking product of sum terms of all the groups.

To get familiar with these steps we will solve some examples.

Illustrative Examples

Example 2.4.1 Minimize the expression.

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

Solution : $(A + B + \bar{C}) = M_1$, $(A + \bar{B} + \bar{C}) = M_3$,
 $(\bar{A} + \bar{B} + \bar{C}) = M_7$, $(\bar{A} + B + C) = M_4$,
 $(A + B + C) = M_0$

Step 1 : Fig. 2.4.1 (a) shows the K-map for three variable and it is plotted according to given maxterms.

Step 2 : There are no isolated 0s.

Step 3 : 0 in the cell 4 is adjacent only to 0 in the cell 0 and 0 in the cell 7 is adjacent only to 0 in the

		BC	B+C	B+ \bar{C}	$\bar{B}+\bar{C}$	$\bar{B}+C$
		00	01	11	10	11
A	0	0 0	0 1	0 3	2	
	1	0 4		0 7		6

Fig. 2.4.1 (a)

cell 3. These two pairs are combined and referred to as group 1 and group 2 respectively.

Step 4 : There are no quads and octets.

Step 5 : The 0 in the cell 1 can be combined with 0 in the cell 3 to form a pair. This pair is referred to as group 3.

Step 6 : In group 1 and in group 2, A is eliminated, whereas in group 3 variable B is eliminated and we get,

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

	BC	B+C	B+ \bar{C}	$\bar{B}+\bar{C}$	$\bar{B}+C$
A	00	01	11	11	10
\bar{A}	0	0	0	0	

Fig. 2.4.1 (b)

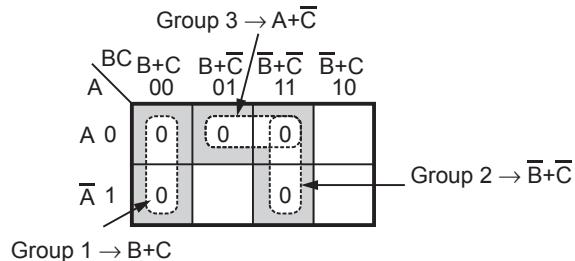


Fig. 2.4.1 (c)

Example 2.4.2 Minimize the following expression in the POS form

$$\begin{aligned} Y &= (\bar{A} + \bar{B} + C + D)(\bar{A} + \bar{B} + \bar{C} + D)(\bar{A} + \bar{B} + \bar{C} + \bar{D}) \\ &\quad (\bar{A} + B + C + D)(A + \bar{B} + \bar{C} + D)(A + \bar{B} + \bar{C} + \bar{D})(A + B + C + D) \\ &\quad (\bar{A} + \bar{B} + C + \bar{D}) \end{aligned}$$

Solution : $(\bar{A} + \bar{B} + C + D) = M_{12}$, $(\bar{A} + \bar{B} + \bar{C} + D) = M_{14}$, $(\bar{A} + \bar{B} + \bar{C} + \bar{D}) = M_{15}$
 $(\bar{A} + B + C + D) = M_8$, $(A + \bar{B} + \bar{C} + D) = M_6$, $(A + \bar{B} + \bar{C} + \bar{D}) = M_7$
 $(A + B + C + D) = M_0$ and $(\bar{A} + \bar{B} + C + \bar{D}) = M_{13}$

Step 1 : Fig. 2.4.2 (a) shows the K-map for four variable and it is plotted according to given maxterms.

Step 2 : There are no isolated 0s.

	CD	C+D	C+ \bar{D}	$\bar{C}+D$	$\bar{C}+D$
AB	00	01	11	10	
$A+B$	00	0	1	3	2
$A+\bar{B}$	01	4	5	0	6
$\bar{A}+\bar{B}$	11	0	0	0	0
$\bar{A}+B$	10	12	13	15	14
	0	8	9	11	10

Fig. 2.4.2 (a)

	CD	C+D	C+ \bar{D}	$\bar{C}+D$	$\bar{C}+D$
AB	00	01	11	10	
$A+B$	00	0	1	3	2
$A+\bar{B}$	01	4	5	0	6
$\bar{A}+\bar{B}$	11	0	0	0	0
$\bar{A}+B$	10	12	13	15	14
	0	8	9	11	10

Fig. 2.4.2 (b)

Step 4 : There are two quads. Cells 12, 13, 14 and 15 forms a quad 1 and cells 6, 7, 14, 15 forms a quad 2. These two quads are referred to as group 2 and group 3, respectively.

Step 5 : All 0s have already been grouped.

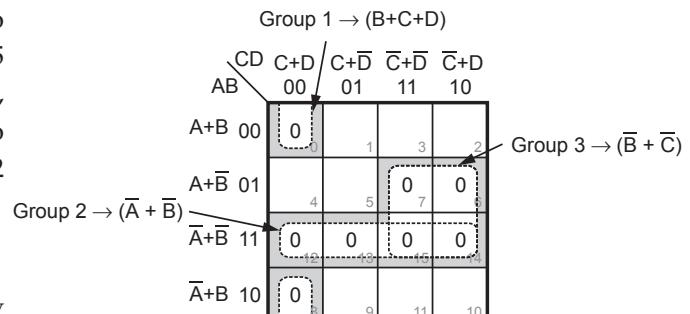


Fig. 2.4.2 (c)

Step 6 : In group 1, variable A is eliminated. In group 2, variable C and D are eliminated and in group 3 variables A and D are eliminated. Therefore we get simplified POS expression as,

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

Example 2.4.3 Reduce the following function using K-map technique

$$f(A, B, C, D) = \prod M(0, 2, 3, 8, 9, 12, 13, 15)$$

Solution : Step 1 : Fig. 2.4.3 (a) shows the K-map for four variables and it is plotted according to given maxterms.

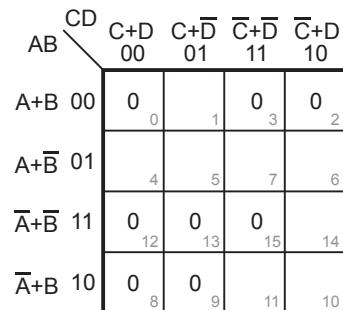


Fig. 2.4.3 (a)

Step 2 : There are no isolated 0s.

Step 3 : The 0 in the cell 15 is adjacent only to 0 in the cell 13 and 0 in the cell 3 is adjacent only to 0 in the cell 2. These two pairs are combined and referred to as group 1 and group 2, respectively.

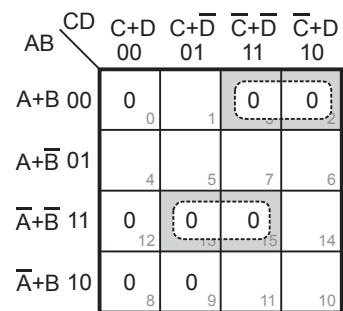


Fig. 2.4.3 (b)

Step 4 : The cells 8, 9, 12 and 13 form a quad which is referred to as group 3.

Step 5 : The remaining 0 in the cell 0 is combined with the 0 in the cell 2 to form a pair, which is referred to as group 4.

	CD	C+D	C+D̄	C̄+D	C̄+D̄
AB	00	01	11	10	10
A+B	00	0	1	0	0
A+̄B	01	4	5	7	6
̄A+̄B	11	0	0	0	14
̄A+B	10	0	0	11	10

Fig. 2.4.3 (c)

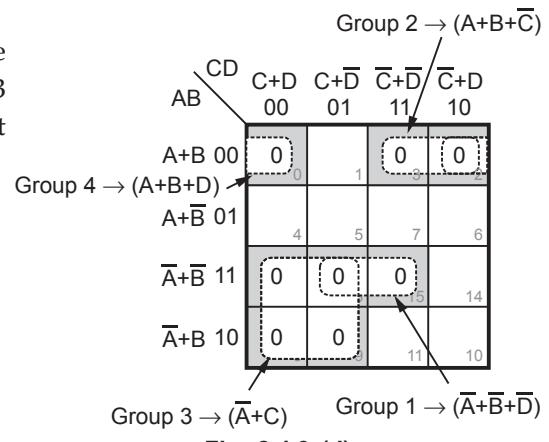


Fig. 2.4.3 (d)

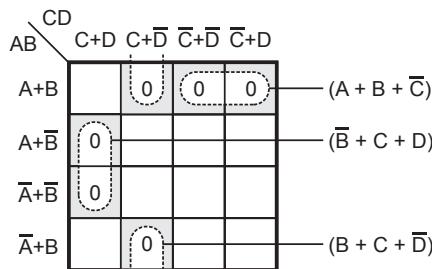
Example 2.4.4 Using K-map convert the following standard POS expression into a minimum POS expression, a standard SOP expression and minimum SOP expression

$$(A' + B' + C + D) (A + B' + C + D) (A + B + C + D') \\ (A + B + C' + D') (A' + B + C + D') (A + B + C' + D).$$

SPPU : Dec.-15, Marks 6

Solution : $(\bar{A} + \bar{B} + C + D) (A + \bar{B} + C + D) (A + B + C + \bar{D}) (A + B + \bar{C} + \bar{D}) (\bar{A} + B + C + \bar{D})$
 $(A + B + \bar{C} + D) = \pi M(12, 4, 1, 3, 9, 2) = \pi M(1, 2, 3, 4, 9, 12)$

Minimum DOS Expression using K-map

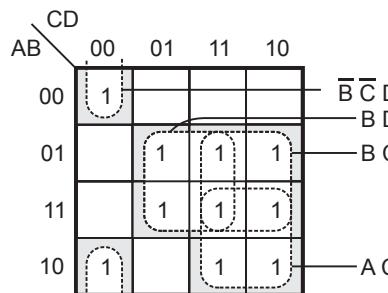


$$\therefore F(A, B, C, D) = (A + B + \bar{C}) (\bar{B} + C + D) (B + C + \bar{D})$$

Standard SOP Expression

$$\begin{aligned}\pi M(1, 2, 3, 4, 9, 12) = \Sigma m(0, 5, 6, 7, 8, 10, 11, 13, 14, 15) \\ = \overline{ABCD} + \overline{ABC}\bar{D} + \overline{A}\overline{BCD} + \overline{AB}\overline{CD} + \overline{ABC}\overline{D} \\ + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}\overline{C}D + ABC\overline{D} + ABCD\end{aligned}$$

Minimum SOP Expression



$$\therefore F(A, B, C, D) = \overline{B}\overline{C}D + BD + BC + AC$$

Examples for Practice

Example 2.4.5 Simplify the following Boolean function for minimal POS form

$$F(w, x, y, z) = \pi M(4, 5, 6, 7, 8, 12) + d(1, 2, 3, 9, 11, 14)$$

$$\text{Ans. : } (w + \overline{x})(\overline{w} + y + z)$$

Example 2.4.6 Reduce the following function using K-map

$$F(A, B, C) = \pi M(0, 1, 2, 3, 4, 7)$$

$$\text{Ans. : } F = A(\overline{B} + \overline{C}) \cdot (B + C)$$

Example 2.4.7 Solve the following using minimization technique

$$Z = f(A, B, C, D) = \pi(1, 2, 3, 6, 8, 11, 14, 15)$$

$$\text{Ans. : } Z = (A + B + \overline{D})(A + B + \overline{C})(\overline{B} + \overline{C} + D)(\overline{A} + \overline{C} + \overline{D})(\overline{A} + B + C + D)$$

Review Question

- Give the steps for simplification of POS expression.

2.5 Summary of Rules for K-Map Simplification

Rules for Simplifying logic function using K-map are :

- Group should not include any cell containing a zero.
- The number of cells in a group must be a power of 2, such as 1, 2, 4, 8 or 16.
- Group may be horizontal, vertical but not diagonal.
- Cell containing 1 must be included in at least one group.

5. Groups may overlap.
6. Each group should be as large as possible to get maximum simplification.
7. Groups may be wrapped around the map. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
8. A cell may be grouped more than once. The only condition is that every group must have at least one cell that does not belong to any other group. Otherwise, redundant terms will result.
9. We need not group all don't care cells, only those that actually contribute to a maximum simplification.
10. All above rules are stated considering the SOP simplification. In case of POS simplification all rules are same except 0 (zero) takes place of 1 (one).

Review Question

1. State the rules for K-map simplification.

2.6 Limitations of Karnaugh Map

The map method of simplification is convenient as long as the number of variables does not exceed five or six. As the number of variables increases it is difficult to make judgements about which combinations form the minimum expression. In case of complex problem with 7, 8 or even 10 variables it is almost an impossible task to simplify expression by the mapping method. Another important point is that the K-map simplification is manual technique and simplification process is heavily depends on the human abilities. To meet this need, W. V. Quine and E. J. McCluskey developed an exact tabular method to simplify the Boolean expression. This method is called the **Quine Mccluskey or tabular method**.

Review Question

1. Explain the limitations of Karnaugh map.

2.7 Quine-McCluskey or Tabular Method SPPU : Dec.-09,18, May-10,14,18

In simplification of Boolean expression we observed that the adjacent minterms can be reduced. These minterms are reduced because they differ by only one literal. For example, $A\bar{B}\bar{C}$ and $A\bar{B}C$ can be reduced because only the B literal differs. The binary numbers equivalent to these minterms are 110 and 100. Note that the 2's place indicates precisely the same information that the literal represented, namely, that the B is the only

literal that differs. Thus we can say that the minterms whose binary equivalent differ only in one place can be combined to reduce the minterms. This is the fundamental principle of the Quine McCluskey method.

Like in other methods, in this method minterms are listed to specify the given function. However, the minterms are written in their binary equivalents. These minterms are grouped according to number of 1s contained and separated by a horizontal line between each number of 1s category, as shown in the Table 2.7.1 (column (b)). This separation of minterms helps in searching the binary minterms that differ only in one place. Once the separation is over, each binary number is compared with every term in the next higher category and if they differ by only one position, a check mark is placed beside each of the two terms and then the term is copied in the second column with a '-' in the position that they differed.

For example, if the terms are 0000 and 0010 then the resultant term will be 0 0 – 0. This term, 00–0 is called an **implicant**, it implies the $\overline{A}\overline{B}\overline{D}$ could be a term in the final expression, covering minterms $\overline{A}\overline{B}\overline{C}\overline{D}$ and $\overline{A}\overline{B}C\overline{D}$. This process of comparison is repeated for every minterm. Once this process is completed the same process is applied to the new resultant terms which are placed in the Table 2.7.2 column (c). These cycles are continued until a single pass through a cycle yields no further elimination of literals. The remaining terms and all the terms that did not match during the process are called the **prime implicants**. Summing one or more prime implicant gives the simplified Boolean expression.

2.7.1 Algorithm for Generating Prime Implicants

1. List all minterms in the binary form.
2. Arrange the minterms according to number of 1s.
3. Compare each binary number with every term in the adjacent next higher category and if they **differ only by one position**, put a check mark and copy the term in the next column with '-' in the position that they differed.
4. Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.
5. List all prime implicants.
6. Select the minimum number of prime implicants which must cover all the minterms.

See the following examples to illustrate the algorithm.

Illustrative Examples

Example 2.7.1 Simplify the following Boolean function by using a Quine McCluskey method. $F(A, B, C, D) = \sum m(0, 2, 3, 6, 7, 8, 10, 12, 13)$.

Solution : Step 1 : List all minterms in the binary form as shown in Table 2.7.1 (column (a)).

Step 2 : Arrange the minterms according to number of 1s, as shown in the Table 2.7.1 (column (b)).

Step 3 : Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark and copy the term in the next column with '-' in the position that they differed.

Minterm	Binary representation	Minterm	Binary representation
m_0	0 0 0 0	m_0	0 0 0 0 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_3	0 0 1 1	m_8	1 0 0 0 ✓
m_6	0 1 1 0	m_3	0 0 1 1 ✓
m_7	0 1 1 1	m_6	0 1 1 0 ✓
m_8	1 0 0 0	m_{10}	1 0 1 0 ✓
m_{10}	1 0 1 0	m_{12}	1 1 0 0 ✓
m_{12}	1 1 0 0	m_7	0 1 1 1 ✓
m_{13}	1 1 0 1	m_{13}	1 1 0 1 ✓

Column (a)

Column (b)

Table 2.7.1

Step 4 : Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary representation	Minterms	Binary representation
0, 2	0 0 - 0 ✓	0, 2, 8, 10	- 0 - 0
0, 8	- 0 0 0 ✓	2, 3, 6, 7	0 - 1 -
2, 3	0 0 1 - ✓		
2, 6	0 - 1 0 ✓		
2, 10	- 0 1 0 ✓		
8, 10	1 0 - 0 ✓		
8, 12	1 - 0 0		
3, 7	0 - 1 1 ✓		
6, 7	0 1 1 - ✓		
12, 13	1 1 0 -		

Column (c)

Column (d)

Table 2.7.2

Step 5 : List the prime implicants.

Step 6 : Select the minimum number of prime implicants which must cover all the minterms.

Table 2.7.4 shows prime implicant selection chart. Each prime implicant is represented in a row and each minterm in a column. Dots are placed in each row to show the composition of minterms that make the prime implicants. From this chart we have to select the minimum number of prime implicants which must cover all the minterms. The selection procedure is as follows.

Prime implicants		Binary representation
A \bar{C} \bar{D}	8, 12	1 - 0 0
A B \bar{C}	12, 13	1 1 0 -
\bar{B} \bar{D}	0, 2, 8, 10	- 0 - 0
\bar{A} C	2, 3, 6, 7	0 - 1 -

Table 2.7.3

Prime implicants	m_0 (Col 1)	m_2 (Col 2)	m_3 (Col 3)	m_6 (Col 4)	m_7 (Col 5)	m_8 (Col 6)	m_{10} (Col 7)	m_{12} (Col 8)	m_{13} (Col 9)
A \bar{C} \bar{D}	8, 12					.		.	
A B \bar{C}	12, 13 ✓							○	○
\bar{B} \bar{D}	0, 2, 8, 10 ✓	○	○			○	○		
\bar{A} C	2, 3, 6, 7 ✓	○	○	○	○				

Table 2.7.4 Prime implicant selection chart

Search for single dot columns and select the prime implicants corresponding to that dot by putting the check mark in front of it.

Search for multiple dot columns one by one. If the corresponding minterm is already included in the final expression ignore the minterm and go to next multi-dot column ; otherwise include the corresponding prime implicant in the final expression.

In our case, column 1 (m_0) has single dot, so include corresponding prime implicant (0, 2, 8, 10). Column 3 (m_3) has single dot so include corresponding prime implicant (2, 3, 6, 7). Columns 4, 5 and 7 have single dots but the corresponding minterms are already included in the final expression.

Column 9 has single dot so include corresponding prime implicant (12, 13) in the final expression. Now search for multi-dot columns. Columns 2, 6 and 8 have multi-dots but their minterms are already included in the final expression. Therefore, the final expression is

$$F(A, B, C, D) = (1 \ 1 \ 0 \ -) + (- \ 0 \ - \ 0) + (0 \ - \ 1 \ -) = AB\bar{C} + \bar{B}\bar{D} + \bar{A}C$$

Example 2.7.2 Minimize the expression using Quine McCluskey method.

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

Solution : Step 1 : List all minterms in the binary form, as shown in Table 2.7.5 (column (a)).

Step 2 : Arrange minterms according to categories of 1s, as shown in the Table 2.7.5 (column (b)) .

Minterm	Binary representation	Minterm	Binary representation
m_4	0 1 0 0	m_2	0 0 1 0
m_5	0 1 0 1	m_4	0 1 0 0 ✓
m_{12}	1 1 0 0	m_5	0 1 0 1 ✓
m_{13}	1 1 0 1	m_9	1 0 0 1 ✓
m_9	1 0 0 1	m_{12}	1 1 0 0 ✓
m_2	0 0 1 0	m_{13}	1 1 0 1 ✓
Column (a)		Column (b)	

Table 2.7.5

Step 3 : Compare each binary number with every term in the next higher category and if they differ by only one position put a check mark and copy the term in the next column with '-' in the position that they differed.

Step 4 : Apply the same process described in step 3 for the resultant column and continue this cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary representation				Minterms	Binary representation				
4, 5	0	1	0	-	✓	4, 5, 12, 13	-	1	0	-
4, 12	-	1	0	0	✓					
5, 13	-	1	0	1	✓					
9, 13	1	-	0	1						
12, 13	1	1	0	-	✓					

Table 2.7.6

Step 5 : List the prime implicants.

Prime implicants	Binary representation
$\bar{A} \bar{B} C \bar{D}$	0 0 1 0
$A \bar{C} D$	1 - 0 1
$B \bar{C}$	- 1 0 -

Table 2.7.7

Step 6 : Select the minimum number of prime implicants which must cover all the minterms.

Prime implicants	m_2 (Col 1)	m_4 (Col 2)	m_5 (Col 3)	m_9 (Col 4)	m_{10} (Col 7)	m_{12} (Col 5)	m_{13} (Col 9)
$\bar{A} \bar{B} C \bar{D}$	2 ✓	⊕					
$A \bar{C} D$	9, 13 ✓			⊕		⊕	⊕
$B \bar{C}$	4, 5, 12, 13 ✓		⊕	⊕	⊕	⊕	⊕

Table 2.7.8 Prime implicant selection chart

The final expression is $Y = (0\ 0\ 1\ 0) + (1\ -\ 0\ 1) + (-\ 1\ 0\ -) = \bar{A} \bar{B} C \bar{D} + A \bar{C} D + B \bar{C}$

2.7.2 Quine McCluskey using Don't Care Terms

The same rules that applied to using don't care terms with Karnaugh map are appropriate for Quine McCluskey. Here, don't care terms are never included as prime implicants by **themselves**. Let us consider the following example.

Illustrative Example

Example 2.7.3 Simplify the following using tabulation methods.

$$Y(w, x, y, z) = \sum m(1, 2, 3, 5, 9, 12, 14, 15) + \sum d(4, 8, 11).$$

Solution : It is important to note that don't care conditions are used to find the prime implicant but it is not compulsory to include don't care terms in the final expression.

Step 1 : List all minterms in the binary form, as shown in the Table 2.7.9 (Column (a)).

Minterm	Binary representation	Minterm	Binary representation
m_1	0 0 0 1	m_1	0 0 0 1 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_3	0 0 1 1	m_4	0 1 0 0 ✓
m_5	0 1 0 1	m_8	1 0 0 0 ✓
m_9	1 0 0 1	m_3	0 0 1 1 ✓
m_{12}	1 1 0 0	m_5	0 1 0 1 ✓
m_{14}	1 1 1 0	m_9	1 0 0 1 ✓
m_{15}	1 1 1 1	m_{12}	1 1 0 0 ✓

dm ₄	0 1 0 0	m ₁₁	1 0 1 1	✓
dm ₈	1 0 0 0	m ₁₄	1 1 1 0	✓
dm ₁₁	1 0 1 1	m ₁₅	1 1 1 1	✓
Column (a)			Column (b)	

Table 2.7.9

Step 2 : Arrange the minterms according to categories of 1s as shown in the Table 2.7.9 (Column (b)).

Step 3 : Compare each binary number with every term in the adjacent next higher category and if they differ by only one position put a check mark and copy the term in the next column with '-' in the position that they differed.

Step 4 : Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Step 5 : List the prime implicants.

Step 6 : Select the minimum number of prime implicants which must cover all the minterms, except don't care minterms.

Only column 2 has single dot i.e. it is essential prime implicant and hence the prime implicant corresponding to it ($m_{2,3}$) is included in the final expression.

Minterms	Binary representation	Minterms	Binary representation
1, 3	0 0 - 1 ✓	1, 3, 9, 11	- 0 - 1
1, 5	0 - 0 1		
1, 9	- 0 0 1 ✓		
2, 3	0 0 1 -		
4, 5	0 1 0 -		
4, 12	- 1 0 0		
8, 9	1 0 0 -		
8, 12	1 - 0 0		
3, 11	- 0 1 1 ✓		
9, 11	1 0 - 1 ✓		
12, 14	1 1 - 0		
11, 15	1 - 1 1		
14, 15	1 1 1 -		

Table 2.7.10

Prime implicants	Binary representation
$\bar{A} \bar{C} D$	1, 5
$\bar{A} \bar{B} C$	2, 3
$\bar{A} B \bar{C}$	4, 5
$B \bar{C} \bar{D}$	4, 12
$A \bar{B} \bar{C}$	8, 9
$A \bar{C} \bar{D}$	8, 12
$A B \bar{D}$	12, 14
$A C D$	11, 15
$A B C$	14, 15
$\bar{B} D$	1, 3, 9, 11

Table 2.7.11

Now search for multi-dot columns. Column 1 has multi-dot and the corresponding terms are not included in the final expression, so we can include either $m_{1,5}$ or $m_{1,3,9,11}$. Let us include prime implicant which has more minterms. Thus minterm 1, 3, 9, 11 is included. Now minterm for column 3 is already included and minterm for column 4 is don't care. Column 5 has a minterm which is not included yet, therefore prime implicant 1, 5 is included in the final expression. The minterms from column 6 and 8 are don't care and the minterm of column 7 (m_9) is already included in the final expression. Column 9 has minterm 12 which is not included yet.

Prime Implicants		m_1 (Col 1)	m_2 (Col 2)	m_3 (Col 3)	dm_4 (Col 4)	m_5 (Col 5)	dm_8 (Col 6)	m_9 (Col 7)	dm_{11} (Col 8)	m_{12} (Col 9)	m_{14} (Col 10)	m_{15} (Col 11)
$\bar{A} \bar{C} D$	1, 5 ✓	◎				◎						
$\bar{A} \bar{B} C$	2, 3 ✓		◎	◎								
$\bar{A} B \bar{C}$	4, 5				•	•						
$B \bar{C} \bar{D}$	4, 12				•					•		
$A \bar{B} \bar{C}$	8, 9				Essential prime implicant		•	•				
$A \bar{C} \bar{D}$	8, 12				•					•		
$A B \bar{D}$	12, 14 ✓									◎	◎	
$A C D$	11, 15								•			•
$A B C$	14, 15 ✓									◎	◎	
$\bar{B} D$	1, 3, 9, 11 ✓	◎		◎				◎	◎			

Table 2.7.12 Prime implicant selection table

To include this term we have 3 options. We include prime implicant 12, 14 because with this inclusion we can also cover minterm 14 in the final expression. The minterm from the remaining column 11 (m_{15}) can be included in the final expression by including prime implicant 14, 15. Therefore, the final expression is

$$\begin{aligned}
 Y &= (0 - 0 1) + (0 0 1 -) + (1 1 - 0) + (1 1 1 -) + (- 0 - 1) \\
 &= \bar{A} \bar{C} D + \bar{A} \bar{B} C + A B \bar{D} + A B C + \bar{B} D
 \end{aligned}$$

Example 2.7.4 Simplify the following function using Quine-McCuskey minimization technique :

$$Y(A, B, C, D) = \Sigma m (0, 1, 2, 3, 5, 7, 8, 9, 11, 14)$$

SPPU : May-18, Dec.-18, Marks 4

Solution :

	Binary Representation			
m_0	0	0	0	0
m_1	0	0	0	1
m_2	0	0	1	0
m_3	0	0	1	1
m_5	0	1	0	1
m_7	0	1	1	1
m_8	1	0	0	0
m_9	1	0	0	1
m_{11}	1	0	1	1
m_{14}	1	1	1	0

	Binary Representations			
$m_0 \checkmark$	0	0	0	0
$m_1 \checkmark$	0	0	0	1
$m_2 \checkmark$	0	0	1	0
$m_8 \checkmark$	1	0	0	0
$m_3 \checkmark$	0	0	1	1
$m_5 \checkmark$	0	1	0	1
$m_9 \checkmark$	1	0	0	1
$m_7 \checkmark$	0	1	1	1
$m_{11} \checkmark$	1	0	1	1
$m_{14} \checkmark$	1	1	1	0

Min terms	Binary Representation			
$0, 1 \checkmark$	0	0	0	-
$0, 2 \checkmark$	0	0	-	0
$0, 8 \checkmark$	-	0	0	0
$1, 3 \checkmark$	0	0	1	-
$1, 5 \checkmark$	0	-	0	1
$1, 9 \checkmark$	-	0	0	1
$2, 3 \checkmark$	0	0	1	-
$8, 9 \checkmark$	-	0	0	1
$3, 7 \checkmark$	0	-	1	1
$3, 11 \checkmark$	-	0	1	1
$5, 7 \checkmark$	0	1	-	1
$9, 11 \checkmark$	1	0	-	1

	Binary Representation			
$0, 1, 2, 3$	0	0	-	-
$0, 1, 8, 9$	-	0	0	-
$1, 3, 5, 7$	0	-	-	1
$1, 3, 9, 11$	-	0	-	1

Prime Implicants			m_0	m_1	m_2	m_3	m_5	m_7	m_8	m_9	m_{11}	m_{14}
✓	0,1,2,3	0 0 - -	$\bar{A}\bar{B}$	◎	◎	◎	◎					
✓	0,1,8,9	- 0 0 -	$\bar{B}\bar{C}$	◎	◎				◎	◎		
✓	1,3,5,7	0 - - 1	$\bar{A}D$		◎		◎	◎				
✓	1,3,9,11	- 0 - 1	$\bar{B}D$		◎		◎			◎	◎	
✓	14	1 1 1 0	$ABC\bar{D}$									◎

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

2.7.3 Prime Implicant Table and Redundant Prime Implicants

We have seen that the prime implicant selection table displays pictorially the covering relationships between the prime implicants and the minterms of the function. It consists of an array of u columns and v rows, where u and v designate the number of minterms for which the function takes on the value 1 and the number of prime implicants, respectively. The entries of the i^{th} row in the chart consists of dots placed at its intersections with the columns, corresponding to minterms covered by the i^{th} prime implicant. For example, the prime implicant chart of $f(ABC) = \sum m(0, 1, 2, 5, 6, 7)$ is shown in Table 2.7.13 (b). It consists of 6 columns corresponding to the minterms of f and six rows which correspond to the prime implicant generated in Table 2.7.13 (a).

Minterm	Binary representation	Minterms	Binary representation
0	0 0 0	0, 1	0 0 -
1	0 0 1	0, 2	0 - 0
2	0 1 0	1, 5	- 0 1
5	1 0 1	2, 6	- 1 0
6	1 1 0	5, 7	1 - 1
7	1 1 1	6, 7	1 1 -

Table 2.7.13 (a)

Prime implicants		m_0	m_1	m_2	m_5	m_6	m_7
$\bar{A}\bar{B}$	0, 1✓	•	•				
$\bar{A}\bar{C}$	0, 2	•		•			
$\bar{B}C$	1, 5		•		•		
$B\bar{C}$	2, 6✓			•		•	
AC	5, 7✓				•		•
AB	6, 7					•	•

Table 2.7.13 (b) Prime implicant selection table

The problem now is to select a minimal subset of prime implicants such that column contains at least one dot in the rows corresponding to the selected subset and the total number of literals in the prime implicants selected as small as possible. In our example, each column has two dots. It is a special case. When each column in the prime implicant selection chart has two or more dots, the chart is known as **cyclic prime implicant table**.

Since all columns have two dots, we have to proceed by trial and error. Both (0, 1) and (0, 2) cover column 0, so we will try (0, 1). After selecting row (0, 1) and columns 0, and 1 we examine column 2, which is covered by (0, 2) and (2, 6). The best choice is

(2, 6) because it covers two of the remaining columns while (0, 2) covers only one of the remaining columns. After selecting row (2, 6) and columns 2 and 6, we see that (5, 7) covers the remaining columns and completes the solution. Therefore, one of the solution is $f = \overline{A}\overline{B} + B\overline{C} + A C$. The prime implicants which are not selected to get the solution [(0, 2), (1, 5), (6, 7)] are called **redundant prime implicants**.

2.7.4 Advantages and Disadvantages of Quine McCluskey Method

The Quine McCluskey method of simplification can be applied to any number of variables and is algorithmic in nature. The simplification process of Quine-McCluskey becomes lengthy and time consuming as number of variables increase; however the algorithm can be easily programmed to run on a computer to identify prime implicants and implicants of any function.

Examples for Practice

Example 2.7.5 : Find prime implicants for the Boolean expression by using Quine McCluskey method. $f(A,B,C,D) = \sum (1, 3, 6, 7, 8, 9, 10, 12, 14, 15) + d(11, 13)$

$$\text{Ans. : } F(A, B, C, D) = \overline{BD} + A + BC$$

Example 2.7.6 : Give simplified logic equation using Quine-McCluskey method for the following Boolean function $f(A, B, C, D) = \sum m (0, 1, 2, 3, 10, 11, 12, 13, 14, 15)$

$$\text{Ans. : } F(A, B, C, D) = \overline{A}\overline{B} + AC + AB$$

Example 2.7.7 : With the help of Quine-McClusky technique determine the PI for the following equation :

$$z = f(A, B, C, D) = \sum (0, 1, 3, 4, 6, 8, 10, 12, 14)$$

SPPU : Dec.-09, Marks 10

$$\text{Ans. : } f(A, B, C, D) = (\overline{CD} + B\overline{D} + A\overline{D} + \overline{AB}D)$$

Example 2.7.8 : Minimize the given terms

$\pi M (0, 1, 4, 11, 13, 15) + \pi d (5, 7, 8)$ using Quine-McCluskey methods and verify the results using K-map methods.

$$\text{Ans. : } f(A, B, C, D) = \overline{A}C + A\overline{D} + A\overline{B}\overline{C}$$

Example 2.7.9 : With the help of Quine-McClusky technique determine the PI, EPI for the following equation :

$$Z = f(A, B, C, D) = \sum (0, 3, 8, 9, 10, 12, 15) \quad \text{SPPU : May-10, Marks 10}$$

$$\begin{aligned} \text{Ans. : PI} &= \overline{A}\overline{B}CD + ABCD + \overline{B}CD\overline{D} + A\overline{C}\overline{D} + A\overline{B}\overline{C} + A\overline{B}\overline{D} \\ \text{EPI} &= \overline{A}\overline{B}CD + ABCD + \overline{B}CD\overline{D} + A\overline{C}\overline{D} + A\overline{B}\overline{C} + A\overline{B}\overline{D} \end{aligned}$$

Example 2.7.10 : Minimize the following expression using Quine-McClusky :

$$F(A, B, C, D) = \Sigma m(1, 5, 6, 12, 13, 14) + d(2, 4)$$

SPPU : May-14, Marks 6

Ans. : $B\bar{C} + B\bar{D} + \bar{A}\bar{C}\bar{D}$

Review Questions

1. Explain the Quine McCluskey method of simplification of Boolean function with the help of example.
2. State the algorithm for generating prime implicants.
3. State the advantages and disadvantages of Quine McCluskey method of simplification.

2.8 Implementation of Boolean Function using Logic Gates

SPPU : Dec.-13, May-15,16,18

The Boolean algebra is used to express the output of any combinational network. Such a network can be implemented using logic gates. Let us see the implementation of SOP and POS Boolean expressions.

2.8.1 Implementation of SOP Boolean Expression

Consider the Boolean expression

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

In this expression, we have three product terms with 2 literals in each product term. Thus, we can implement these product terms by using three 2-input AND gates, as shown in the Fig. 2.8.1. Expression tells that these product terms should be ORed to get the output F. We have three product terms so we have to use 3-input OR gate to obtain the sum of products. It is important to note that literals are complemented using NOT gates.

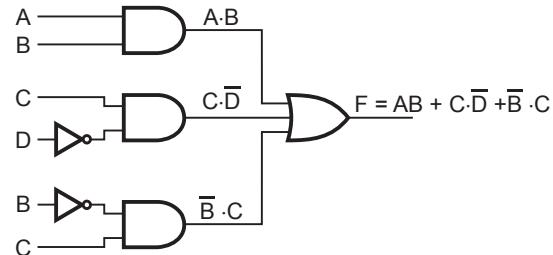


Fig. 2.8.1 Implementation of SOP Boolean expression

2.8.2 Implementation of POS Boolean Expression

Consider the Boolean expression

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

In this expression, we have three sum terms with 2 literals in two terms and 3 literals in one term. We can implement these sum terms by using two 2-input OR gates and one 3-input OR gate, as shown in the Fig. 2.8.2. Expression tells that these product terms should be ANDed to get the output F. We have three sum terms so we have to use

3-input AND gate to obtain the product of sums. It is important to note that literals are complemented using NOT gates.

In the previous examples we have seen that simplified SOP Boolean expression can be implemented using AND-OR gates. The AND-OR implementation is a two level implementation. In the first level we implement all product terms using AND gates and in the second level all product terms are logically ORed using OR gate. In case of POS expression we use OR-AND implementation. Here, we implement all sum terms using OR gates in the first level and all sum terms are logically ANDed using AND gate, to get product of sum, in the second level. We know that, the logic gates are available in the integrated circuit (IC) packages. When we implement logic circuit using basic gates, we require ICs for AND, OR and NOT gates.

Many times it may happen that all gates from the IC packages are not required to build the circuit and thus remaining gates are unused. Consider a combinational circuit which requires two 2-input AND gates and one 2-input OR gate as shown in Fig. 2.8.3. To implement such a circuit we require IC 7408 (Four 2-input AND gates) and IC 7432 (Four 2-input OR gate). When we use these two ICs we find that two 2-input AND gates are unused and three 2-input OR gates are unused. Thus, the utility factor is very poor. This utility factor can be increased by using universal gates to implement logic functions.

Example 2.8.1 Minimize the following function using K-map and realize using logic gates.

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

SPPU : Dec.-13,19, Marks 4

Solution :

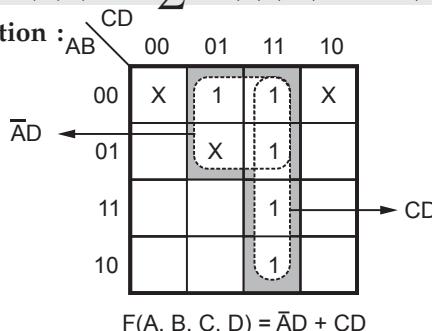


Fig. 2.8.4

Logic diagram

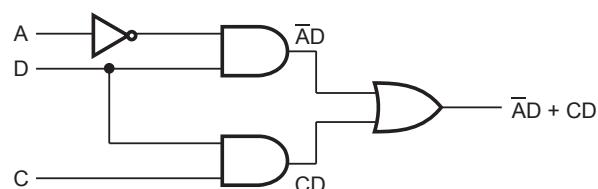


Fig. 2.8.5

Example 2.8.2 Minimize the following function using K-map and realize using logic gates :

$$F(A, B, C, D) = \sum m (1, 5, 7, 13, 15) + d(0, 6, 12, 14)$$

SPPU : May-15,18, Marks 4

Solution :

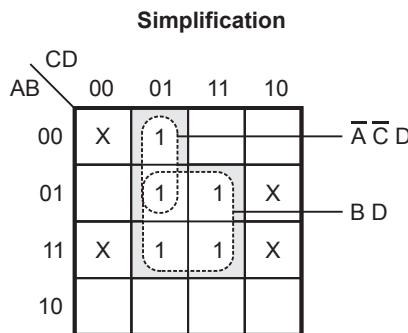


Fig. 2.8.6

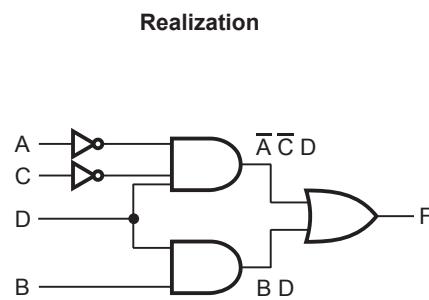


Fig. 2.8.7

Example 2.8.3 Minimize the following function using K-map and realize using logic gates :

$$F(A, B, C, D) = \Sigma m (0, 2, 5, 8, 11, 15) + d(1, 7, 14)$$

SPPU : May-16, Marks 4

Solution :

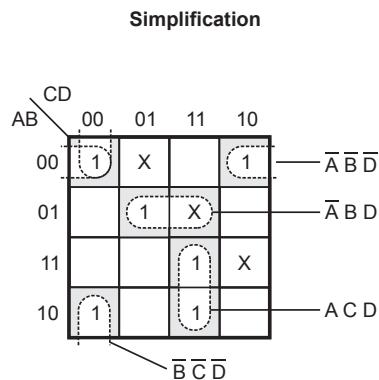


Fig. 2.8.8

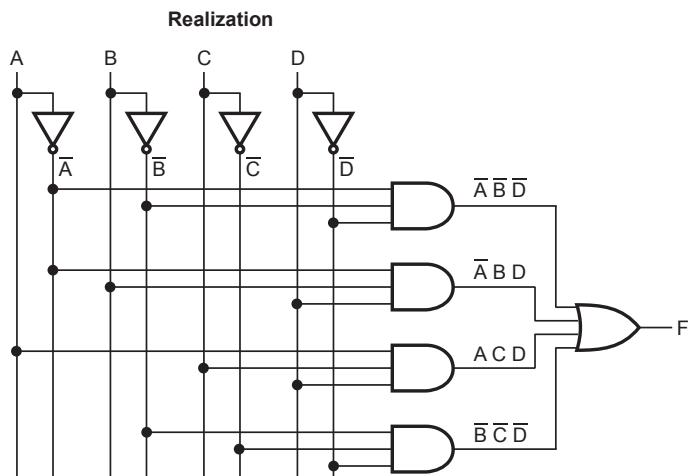


Fig. 2.8.9

2.9 Universal Gates

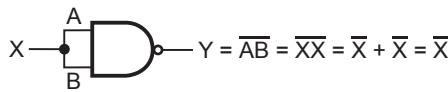
The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates.

2.9.1 NAND Gate

The NAND gate can be used to generate the NOT function, the AND function, the OR function, and the NOR function.

NOT Function :

An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Fig. 2.9.1, for a two-input gate.



A	B	\overline{AB}
0	0	1
0	1	1
1	0	1
1	1	0

$X = 0 \quad Y = 1$
 $X = 1 \quad Y = 0$

Fig. 2.9.1 NOT function using NAND gate

AND Function :

An AND function can be generated using only NAND gates. It is generated by simply inverting output of NAND gate; i.e. $\overline{AB} = AB$. Fig. 2.9.2 shows the two input AND gate using NAND gates.

A	B	\overline{AB}	$\overline{\overline{AB}}$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Table 2.9.1 Truth table

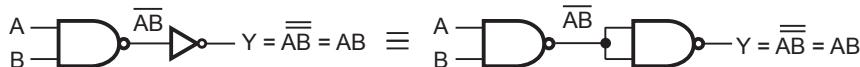


Fig. 2.9.2 AND function using NAND gates

OR Function :

OR function is generated using only NAND gates as follows : We know that Boolean expression for OR gate is

$$\begin{aligned} Y &= A + B = \overline{\overline{A}} + \overline{\overline{B}} && [\overline{\overline{A}} = A] \\ &= \overline{\overline{A} \cdot \overline{B}} && \text{DeMorgan's Theorem 1} \end{aligned}$$

The above equation is implemented using only NAND gates as shown in the Fig. 2.9.3.

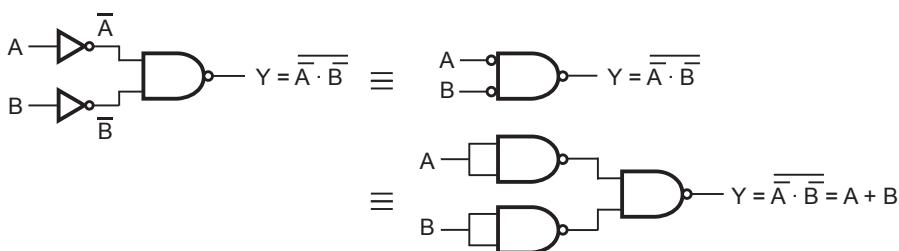


Fig. 2.9.3 OR function using only NAND gates

Note Bubble at the input of NAND gate indicates inverted input.

A	B	$A + B$		A	B	$\bar{A} \cdot \bar{B}$	$\bar{\bar{A}} \cdot \bar{\bar{B}}$
0	0	0	≡	0	0	1	0
0	1	1		0	1	0	1
1	0	1		1	0	0	1
1	1	1		1	1	0	1

Table 2.9.2 Truth table

NOR Function :

NOR function is generated using only NAND gates as follows : We know that Boolean expression for NOR gate is

$$\begin{aligned} Y &= \overline{A+B} = \overline{A} \cdot \overline{B} && \text{DeMorgan's Theorem 2} \\ &= \overline{\overline{A} \cdot \overline{B}} \\ &= \overline{\overline{A}} = A \end{aligned}$$

The above equation is implemented using only NAND gates, as shown in the Fig. 2.9.4.

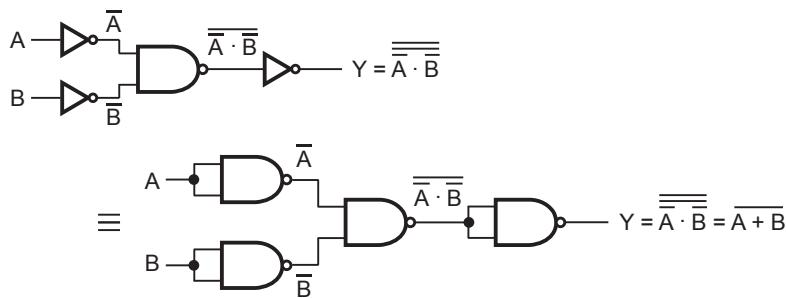


Fig. 2.9.4 NOR function using only NAND gates

A	B	$\overline{A+B}$		A	B	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A}} \cdot \overline{\overline{B}}$	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	≡	0	0	1	0	1
0	1	0		0	1	0	1	0
1	0	0		1	0	0	1	0
1	1	0		1	1	0	1	0

2.9.2 NOR Gate

Similar to NAND gate, the NOR gate is also a universal gate, since it can be used to generate the NOT, AND, OR and NAND functions.

NOT Function :

An inverter can be made from a NOR gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Fig. 2.9.5.

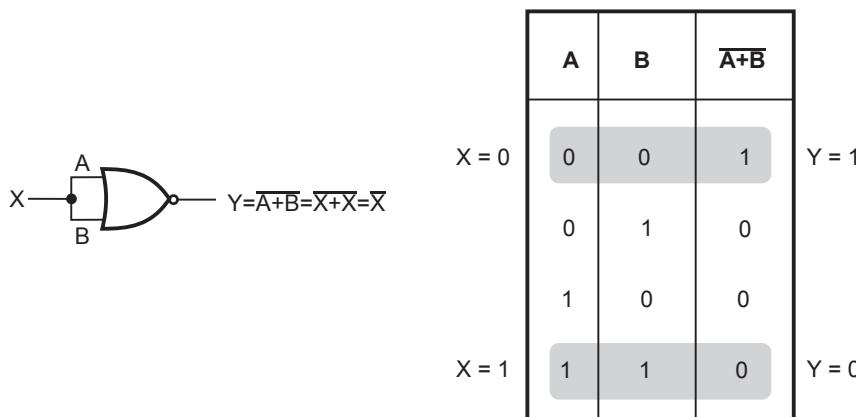


Fig. 2.9.5 NOT function using NOR gate

OR Function :

An OR function can be generated using only NOR gates. It can be generated by simply inverting output of NOR gate; i.e. $\overline{\overline{A+B}} = A + B$. Fig. 2.9.6 shows the two input OR gate using NOR gates.

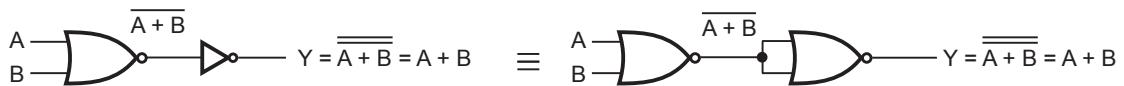


Fig. 2.9.6 OR function using NOR gates

A	B	$A + B$	$\overline{A+B}$	$\overline{\overline{A+B}}$
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1

Table 2.9.3 Truth table

AND Function :

AND function is generated using only NOR gates as follows : We know that Boolean expression for AND gate is

$$\begin{aligned}
 Y &= A \cdot B = \overline{\overline{A}} \cdot \overline{\overline{B}} \\
 &= \overline{\overline{A+B}}
 \end{aligned}
 \quad [\overline{\overline{A}} = A] \quad \text{De-Morgan's Theorem 2}$$

The above equation is implemented using only NOR gates as shown in the Fig. 2.9.7.

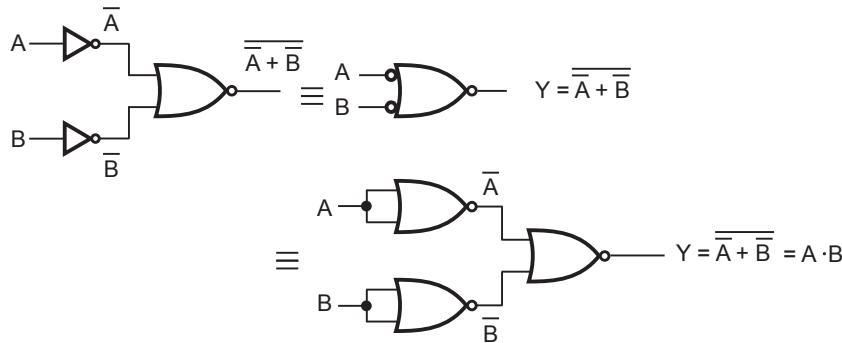


Fig. 2.9.7 AND function using NOR gates

Note Bubble at the input of NOR gate indicates inverted input.

A	B	$A \cdot B$	A	B	$\bar{A} + \bar{B}$	$\bar{\bar{A}} + \bar{\bar{B}}$
0	0	0	0	0	1	0
0	1	0	0	1	1	0
1	0	0	1	0	1	0
1	1	1	1	1	0	1

Table 2.9.4 Truth table

NAND Function :

NAND function is generated using only NOR gates as follows : We know that Boolean expression for NAND gate is

$$\begin{aligned} Y &= \overline{A \cdot B} = \overline{A} + \overline{B} && \text{DeMorgan's Theorem 1} \\ &= \overline{\overline{A} + \overline{B}} && [\overline{\overline{A}} = A] \end{aligned}$$

The above equation is implemented using only NOR gates, as shown in the Fig. 2.9.8.

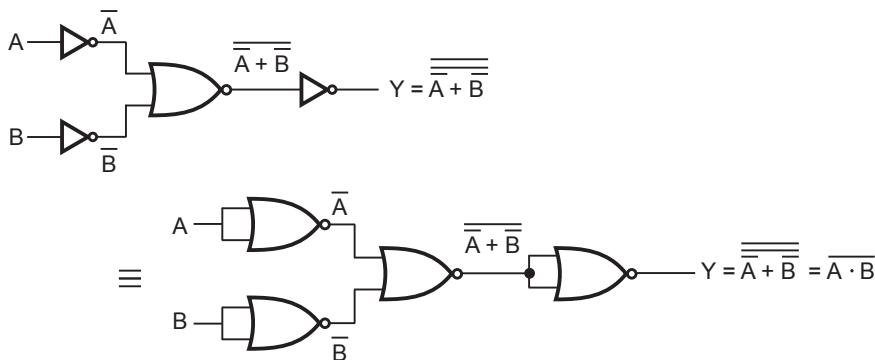


Fig. 2.9.8 NAND function using only NOR gates

A	B	$A \cdot B$	A	B	$\bar{A} + \bar{B}$	$\bar{A} + \bar{B}$	$\bar{\bar{A}} + \bar{\bar{B}}$
0	0	1	0	0	1	0	1
0	1	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	1	0	1	1	0	1	0

Table 2.9.5 Truth table

Review Questions

1. What are universal gates ? Give examples.
2. Why NAND and NOR gates are called universal gates ?
3. Why digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates ?
4. Realize i) AND gate ii) NOR gate using only NAND gates.
5. Realize i) OR gate ii) EX-OR gate using NAND gates.
6. Realize i) OR gate ii) AND gate using only NOR gates.

2.10 NAND-NAND Implementation SPPU : May-06,07,08, Dec.-05,06,11,15,16

The implementation of a Boolean function with NAND-NAND logic requires that the function be simplified in the sum of product form. The relationship between AND-OR logic and NAND-NAND logic is explained using following example.

Consider the Boolean function : $Y = A B C + D E + F$.

This Boolean function can be implemented using AND-OR logic, as shown in Fig. 2.10.1 (a).

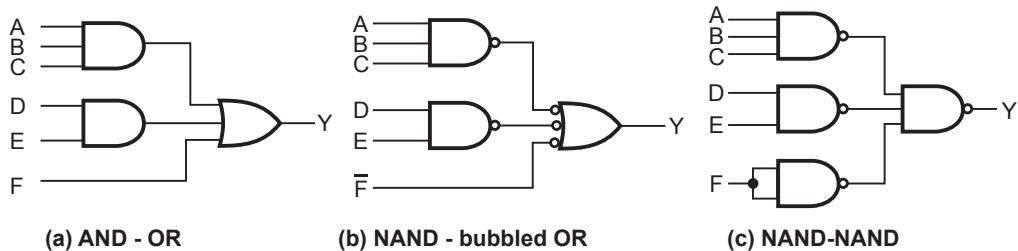


Fig. 2.10.1 NAND-NAND implementation

Fig. 2.10.1 (b) shows the AND gates are replaced by NAND gates and the OR gate is replaced by a bubbled OR gate. The implementation shown in Fig. 2.10.1 (b) is equivalent to implementation shown in Fig. 2.10.1 (a), because two bubbled on the same line represent double inversion (complementation) which is equivalent to having no

bubble on the line. In case of single variable, F, the complemented variable is again complemented by bubble to produce the normal value of F.

In Fig. 2.10.1 (c), the output NAND gate is redrawn with the conventional symbol. The NAND gate with same inputs gives complemented result, therefore \bar{F} is replaced by NAND gate with F input to its both inputs. Thus all the three implementations of Boolean function are equivalent.

From the above example we can summarize the rules for obtaining the NAND-NAND logic diagram from a Boolean function as follows :

1. Simplify the given Boolean function and express it in sum of product form (SOP form).
2. Draw a NAND gate for each product term of the function that has two or more literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first level gates.
3. If Boolean function includes any single literal or literals draw NAND gate for each single literal and connect corresponding literal as an input to the NAND gate.
4. Draw a single NAND gate in the second level, with inputs coming from outputs of first level gates.

Illustrative Examples

Example 2.10.1 Implement the following Boolean function with NAND-NAND logic

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

Solution : Step 1 : Simplify the given Boolean function.

$$\begin{aligned} Y &= A C + A B C + \bar{A} B C + A B + D \\ &= A C + B C (A + \bar{A}) + A B + D = A C + B C + A B + D \end{aligned}$$

Step 2 : Implement using AND-OR logic.

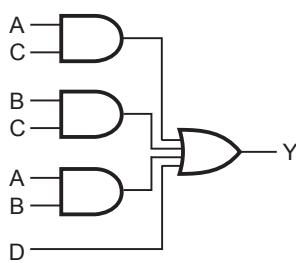


Fig. 2.10.2 (a)

Step 3 : Convert AND-OR logic to NAND-NAND logic.

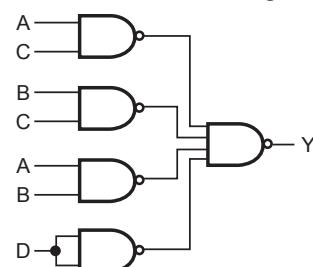


Fig. 2.10.2 (b)

Example 2.10.2 Implement the following Boolean function with NAND-NAND logic

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

Solution :

Step 1 : Implement Boolean function with AND-OR logic.

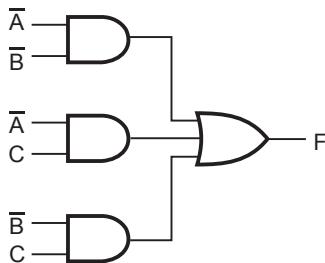


Fig. 2.10.3 (a)

Step 2 : Convert AND-OR logic to NAND-NAND logic.

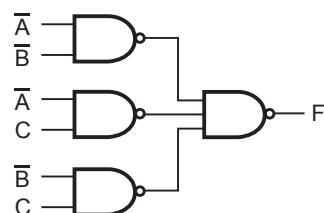


Fig. 2.10.3 (b)

Note It is possible to directly go to step 2 skipping step 1. Here, step 1 is included for clear understanding.

Example 2.10.3 Implement the following Boolean function with NAND-NAND logic.

$$F = (A, B, C) = \sum m (0, 1, 3, 5)$$

Solution : **Step 1 :** Simplify the given Boolean function.

A	BC	00	01	11	10
0		1	1	1	0
1		0	1	0	0

Fig. 2.10.4

$$\therefore F = \overline{A} \overline{B} + \overline{A} C + \overline{B} C$$

Step 2 : Implement Boolean function with AND-OR logic.

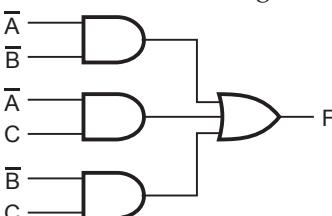


Fig. 2.10.5 (a)

Step 3 : Convert AND-OR logic to NAND-NAND logic.

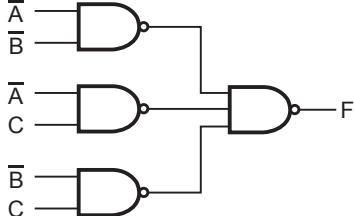


Fig. 2.10.5 (b)

Note It is possible to directly go to step 3 skipping step 2. Here, step 2 is included for clear understanding.

Example 2.10.4 Implement EX-OR gate using only NAND gates.

Solution : The Boolean expression for EX-OR gate is : $Y = A\bar{B} + \bar{A}B$

We can implement AND-OR logic by using NAND-NAND logic as shown in Fig. 2.10.6 (b).

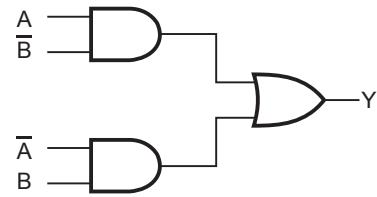


Fig. 2.10.6 (a)

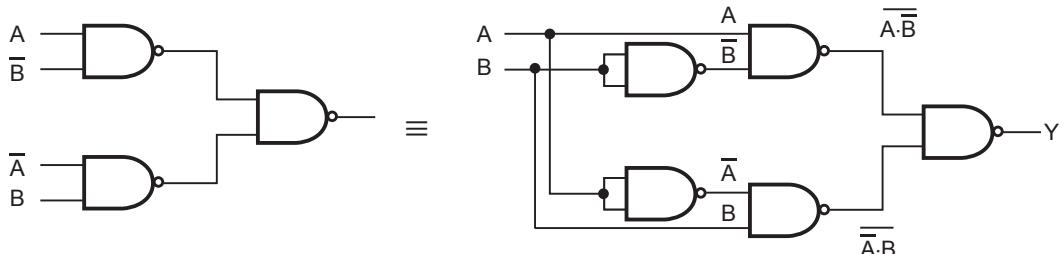


Fig. 2.10.6 (b)

Example 2.10.5 Implement EX-NOR gate using only NAND gates.

Solution : The Boolean expression for EX-NOR gate is $y = AB + \bar{A}\bar{B}$. We can implement AND-OR logic by using NAND-NAND logic as shown in Fig. 2.10.7.

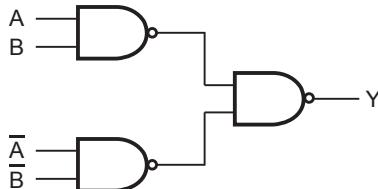


Fig. 2.10.7

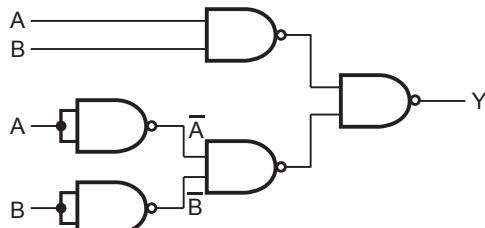


Fig. 2.10.7 (a)

Example 2.10.6 Minimize $f(A, B, C, D) = \sum m(0, 2, 5, 6, 7, 13) + d(8, 10, 15)$

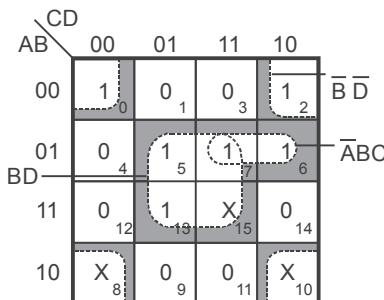
Implement using NAND gates.

$$f(A, B, C, D) = \sum m(0, 2, 5, 6, 7, 13) + d(8, 10, 15).$$

SPPU : May-07, Marks 8

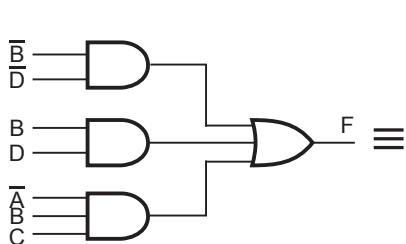
Solution : if $(A, B, C, D) = \sum m(0, 2, 5, 6, 7, 13) + d(8, 10, 15)$

K-map simplification :



$$f(A, B, C, D) = \overline{B} \overline{D} + B D + \overline{A} B C$$

Implementation :



Implementation using NAND gates :

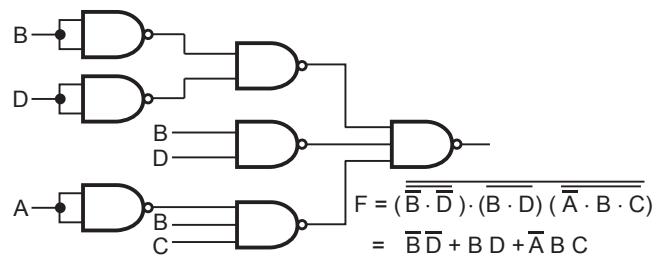


Fig. 2.10.8

Example 2.10.7 Simplify the following 4 variable function using K-map and represent using NAND gate only :

$$\begin{aligned} F(A, B, C, D) = & (\overline{ABC} \overline{D} + \overline{AB} \overline{CD} + \overline{ABC} \overline{D} + \overline{AB} \overline{C} \overline{D} \\ & + \overline{AB} \overline{C} \overline{D} + ABC \overline{D}) + d(\overline{A} \overline{B} \overline{C} D + \overline{A} \overline{B} C \overline{D} \\ & + \overline{A} \overline{B} C \overline{D} + A \overline{B} \overline{C} D + A \overline{B} C \overline{D} + A B \overline{C} \overline{D}) \end{aligned}$$

SPPU : May-08, Marks 8

Solution :

$$\begin{aligned} F(A, B, C, D) = & (\overline{ABC} \overline{D} + \overline{AB} \overline{CD} \\ & + \overline{ABC} \overline{D} + \overline{AB} \overline{C} \overline{D} \\ & + A \overline{B} \overline{C} \overline{D} + A B \overline{C} \overline{D}) \\ & + d(\overline{A} \overline{B} \overline{C} D + \overline{A} \overline{B} C \overline{D}) \\ & + \overline{A} \overline{B} C \overline{D} + A \overline{B} \overline{C} D \\ & + A \overline{B} C \overline{D} + A B \overline{C} \overline{D}) \end{aligned}$$

K-map simplification :

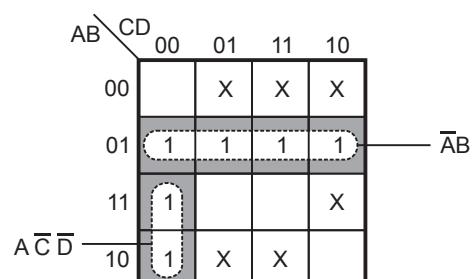
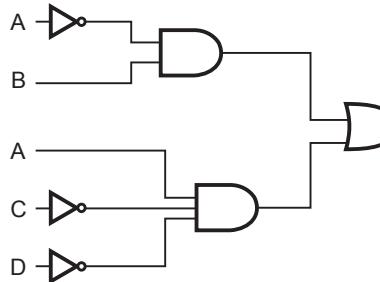


Fig. 2.10.9

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

$$\therefore f = \overline{A}B + A\overline{C}\overline{D}$$

Implementation using basic gates



Implementation using only NAND gates

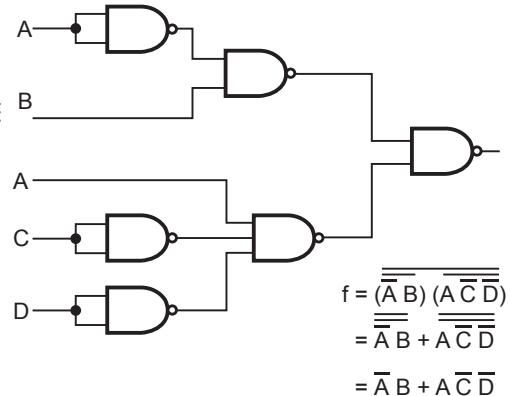


Fig. 2.10.9 (a)

Note AND-OR logic can be implemented by NAND-NAND logic.

Example 2.10.8 Minimize the following equation using K-map and realize it using NAND gates only.

$$Y = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 11, 14)$$

SPPU : Dec.-11, Marks 10

Solution : K-map simplification

Implementation

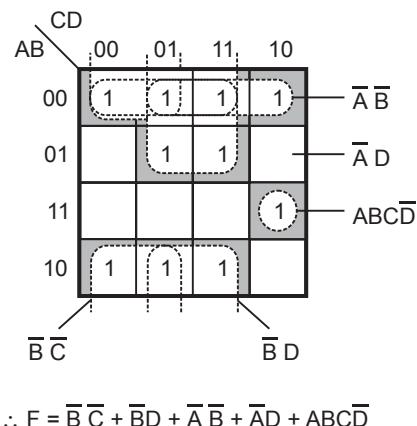


Fig. 2.10.10

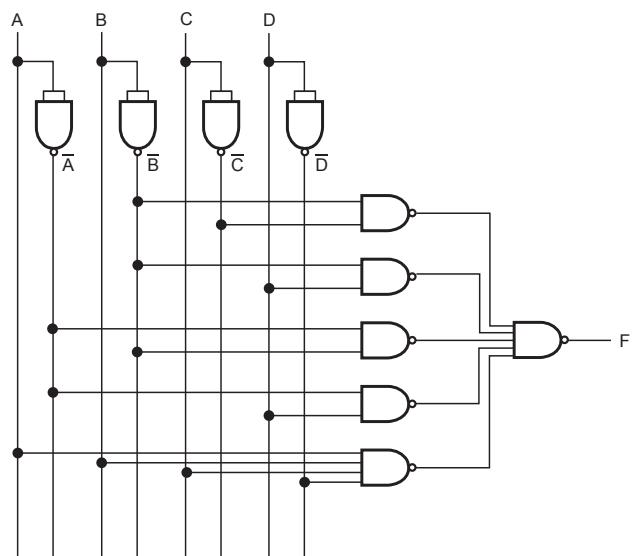


Fig. 2.10.11

Example 2.10.9 Implement each expression with NAND logic 1 :

$$i) ABC + DE \quad ii) ABC + D' + E'.$$

SPPU : Dec.-15, Marks 4

Solution : i) $ABC + DE$

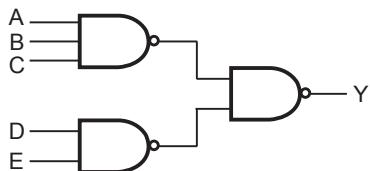


Fig. 2.10.12

ii) $ABC + \bar{D} + \bar{E}$

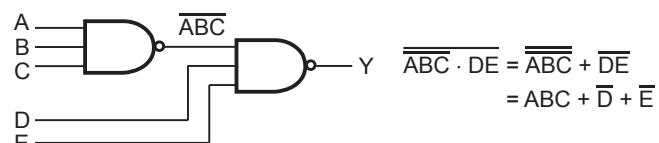


Fig. 2.10.13

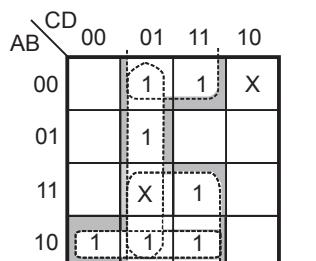
Example 2.10.10 Minimize the following logic function and realize using NAND gates :

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

SPPU : Dec.-16, Marks 4

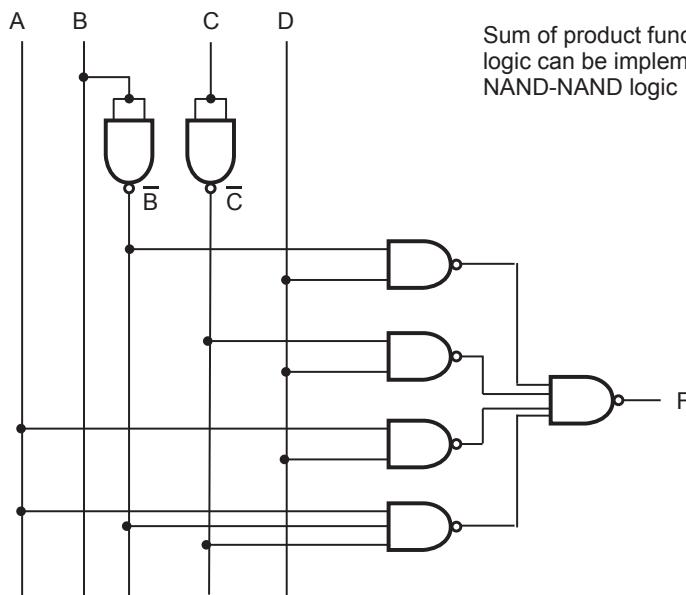
Solution :

K-map simplification



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

Realization using NAND gate



Sum of product function, i.e. AND-OR logic can be implemented using NAND-NAND logic

Examples for Practice

Example 2.10.11 : Simplify the following Boolean function using K-map and implement the same using only NAND gates :

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

[Ans. : $Y = \overline{D} \cdot (\overline{B} \cdot \overline{C}) \cdot (\overline{A} \cdot \overline{C})$] **SPPU : Dec.-05, Marks 6**

Example 2.10.12 : Minimize the following equation using K-map and realise it using NAND gates only :

$$Y = \sum m(4, 5, 8, 9, 11, 12, 13, 15)$$

[Ans. : $y = \overline{\overline{BC}} + \overline{\overline{AB}} + \overline{AD}$] **SPPU : May-06, Marks 8**

Example 2.10.13 : Minimize the function using K-map and implement using only NAND gates :

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

[Ans. : $F = \overline{\overline{AB}} \cdot \overline{\overline{(AC \cdot D)}}$] **SPPU : Dec.-06, Marks 6**

2.11 NOR-NOR Implementation

SPPU : Dec.-05,06,07

The NOR function is a dual of the NAND function. For this reason, the implementation procedures and rules for NOR-NOR logic are the duals of the corresponding procedures and rules developed for NAND-NAND logic.

The implementation of a Boolean function with NOR-NOR logic requires that the function be simplified in the product of sum form. In product of sum form, we implement all sum terms using OR gates. This constitutes the first level. In the second level all sum terms are logically ANDed using AND gate. The relationship between OR-AND logic and NOR-NOR is explained using following example.

Consider the Boolean function : $Y = (A + B + C)(D + E) F$

This Boolean function can be implemented using OR-AND logic, as shown in the Fig. 2.11.1 (a). Fig. 2.11.1 (b) shows the OR gates are replaced by NOR gates and the AND gate is replaced by a bubbled AND gate. The implementation shown in Fig. 2.11.1 (b) is equivalent to implementation shown in Fig. 2.11.1 (a), because two bubbles on the same line represent double inversion (complementation) which is

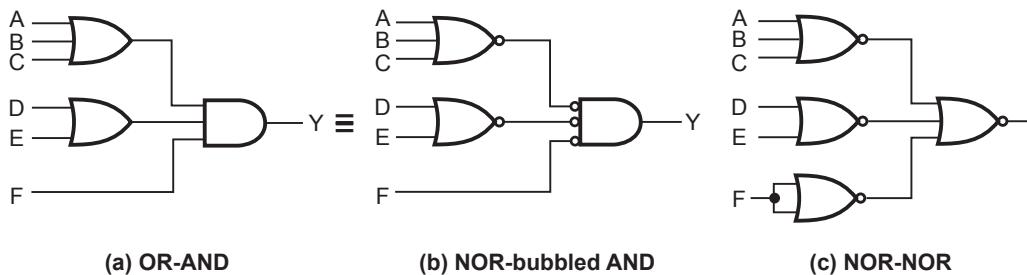


Fig. 2.11.1

equivalent to having no bubble on the line. In case of single variable, F, the complemented variable again complemented by bubble to produce the normal value of F.

In Fig. 2.11.1 (c), the output NOR gate is redrawn with the conventional symbol. The NOR gate with same inputs gives complemented result, therefore, \bar{F} is replaced by NOR gate with F input to its both inputs. Thus all the three implementations of Boolean function are equivalent.

From the above example we can summarize the rules for obtaining the NOR-NOR logic diagram from a Boolean function as follows :

1. Simplify the given Boolean function and express it in product of sum form (POS form).
2. Draw a NOR gate for each sum term of the function that has two or more literals. The inputs to each NOR gate are the literals of the term. This constitute a group of first level gates.
3. If Boolean function includes any single literal or literals, draw NOR gate for each single literal and connect corresponding literal as an input to the NOR gate.
4. Draw a single NOR gate in the second level, with inputs coming from outputs of first level gates.

Illustrative Examples

Example 2.11.1 Implement the following Boolean function with NOR-NOR logic.

$$F = (A, B, C) = \prod M (0, 2, 4, 5, 6)$$

Solution : Step 1 : Simplify the given Boolean function.

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

		BC	00	01	11	10
		A	0			
0	1	0	0	0		0
		1	0	0		0

Fig. 2.11.2

Step 2 : Implement Boolean function with OR-AND logic.

Step 3 : Convert OR-AND logic to NOR-NOR logic.

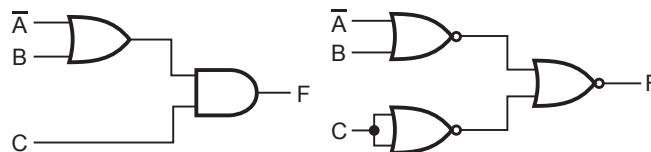


Fig. 2.11.3 (a)

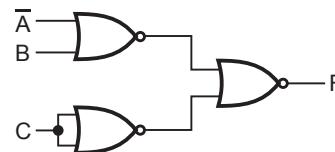


Fig. 2.11.4

Note It is possible to directly go to step 3 skipping step 2. Here, step 2 is included for clear understanding.

Example 2.11.2 Implement EX-NOR gate using only NOR gates.

Solution : The Boolean expression for EX-NOR gate is :

$$\begin{aligned} Y &= AB + \bar{A}\bar{B} = \overline{\bar{A}\bar{B}} + \overline{AB} \\ &= \overline{\bar{A}\bar{B}} \cdot \overline{AB} = (\bar{A} + B) \cdot (A + \bar{B}) \end{aligned}$$

We can implement OR-AND logic by using NOR-NOR logic, as shown in Fig. 2.11.5 (b).

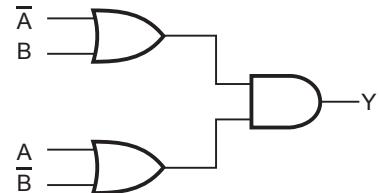


Fig. 2.11.5 (a)

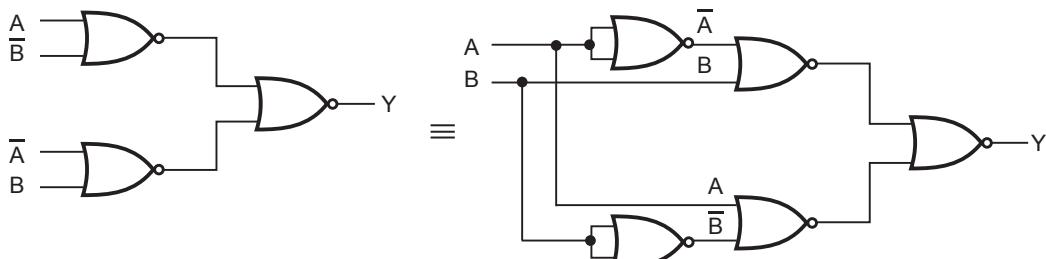


Fig. 2.11.5 (b)

Example 2.11.3 Implement EX-OR gate using only NOR gates.

Solution : Boolean expression of EX-OR gate $Y = A\bar{B} + \bar{A}B = \overline{A\bar{B}} + \overline{\bar{A}B}$

$$\begin{aligned} &= \overline{A\bar{B}} \cdot \overline{\bar{A}B} \\ &= (\bar{A} + \bar{B})(A + B) \end{aligned}$$

Note : We can implement OR-AND logic by NOR-NOR logic.

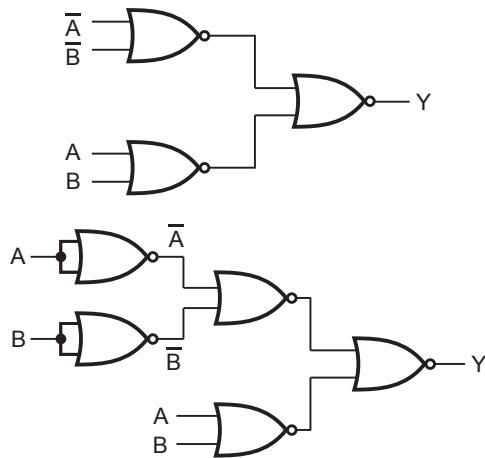


Fig. 2.11.6

Example 2.11.4 Minimize the function using K-map and implement using only NOR gates :

$$f(w, x, y, z) = \pi M(1, 3, 5, 7, 13, 15)$$

SPPU : Dec.-05, Marks 6

Solution : K-map simplification :

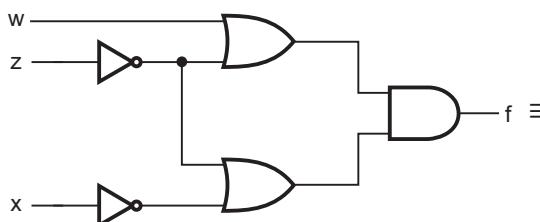
$$f(w, x, y, z) = (w + \bar{z})(\bar{x} + \bar{z})$$

Implementation

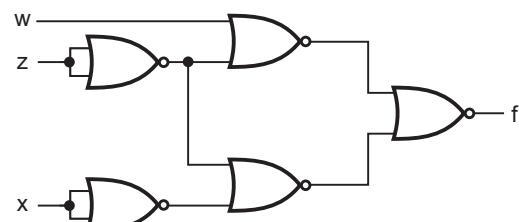
The OR-AND logic can be directly implemented using NOR-NOR logic as below,

wx	yz	y+z	y+̄z	̄y+z	̄y+̄z	w+̄z
w+x	00	1	0	0	1	
w+̄x	01	1	0	0	1	
̄w+x	11	1	0	0	1	
̄w+x	10	1	1	1	1	

Fig. 2.11.7



(a) Implementation using OR-AND logic



(b) Implementation using only NOR gates

Fig. 2.11.7

Example 2.11.5 Minimize the function using K-map and implement using only one type of gate : $f(A, B, C, D) = \pi M(0, 3, 5, 6, 10) \cdot d(1, 2, 11, 12)$.

SPPU : Dec.-06, Marks 6

Solution : K-map simplification :

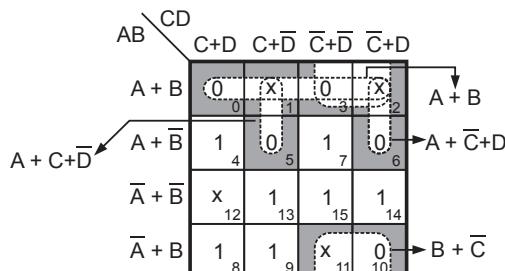


Fig. 2.11.8

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

OR-AND logic can be replaced by NOR-NOR logic

Implementation :

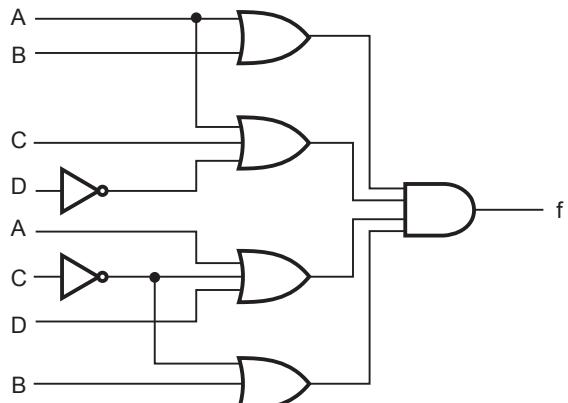


Fig. 2.11.8 (a)

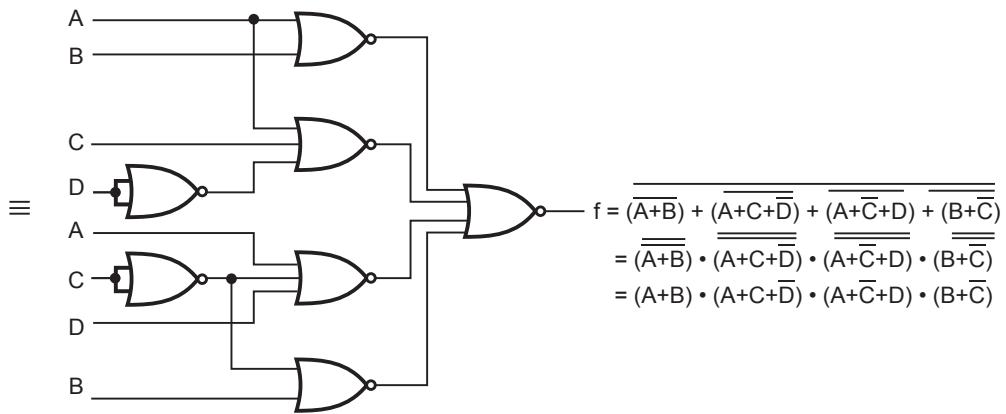


Fig. 2.11.8 (b)

Example 2.11.6 Minimize function using K-map and implement using NOR gate :

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

SPPU : Dec.-07, Marks 6

Solution : K-map simplification :

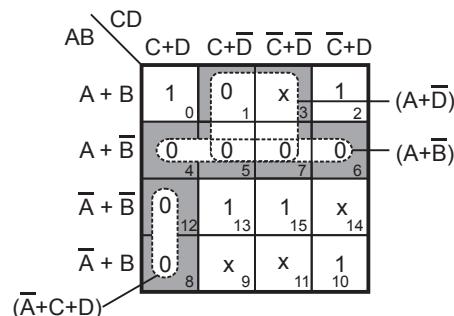
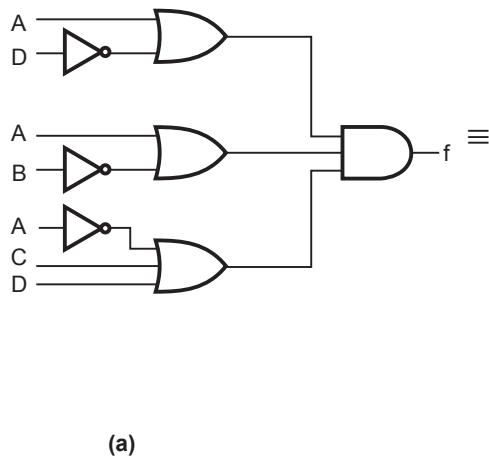
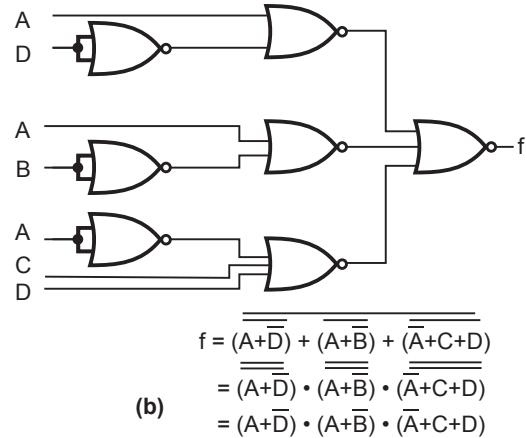


Fig. 2.11.9

$$\therefore f(ABCD) = (A + \overline{D})(A + \overline{B})(\overline{A} + C + D)$$

Implementation using basic gates**Implementation using only NOR gates****Fig. 2.11.9**

Note OR-AND logic can be converted by NOR-NOR logic.



Notes

UNIT - II

3

Combinational Logic Design

Contents

3.1	<i>Introduction</i>	
3.2	<i>Code Converter</i>	<i>Dec.-10,12,13,17, May-12,13,15,16</i> Marks 8
3.3	<i>Adders</i>	<i>May-07,14, Dec.-15,18</i> Marks 6
3.4	<i>Subtractors</i>	<i>Dec.-08,</i> Marks 8
3.5	<i>Parallel Adder</i>	<i>May-06, Dec.-06,</i> Marks 4
3.6	<i>Parallel Subtractor</i>	
3.7	<i>Parallel Adder / Subtractor</i>	
3.8	<i>Four-bit Parallel Adder (7483/74283)</i>	
3.9	<i>BCD Adder</i>	<i>Dec.-05,06,10,12,14,16, May-05,07,08,10,12</i> Marks 10
3.10	<i>Look-Ahead Carry Adder</i>	<i>May-06,14,17, Dec.-14,17</i> Marks 6
3.11	<i>Multiplexers (MUX)</i>	<i>May-05,10,11,12,13,17, Dec.-10,12,16,17,18,19</i> Marks 8
3.12	<i>Demultiplexers (DEMUX)</i>	<i>Dec.-11,</i> Marks 8
3.13	<i>Decoder</i>	<i>May-10,11, Dec.-10,13</i> Marks 8
3.14	<i>Magnitude Comparator using 7485</i>	<i>May-10,12,18, Dec.-10,12,</i> Marks 8
3.15	<i>Parity Generator and Checker using 74180..</i>	<i>Dec.-11,</i> Marks 8

3.1 Introduction

When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called **combinational logic circuit**. In combinational logic circuit, the output variables are at all times dependent on the combination of input variables.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the input variables and generate output signals. This process transforms binary information from the given input data to the required output data. Fig. 3.1.1 shows the block diagram of a combinational circuit. As shown in Fig. 3.1.1, the combinational circuit accepts n -input binary variables and generates output variables depending on the logical combination of gates.

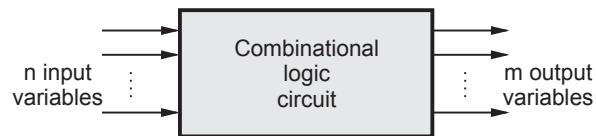


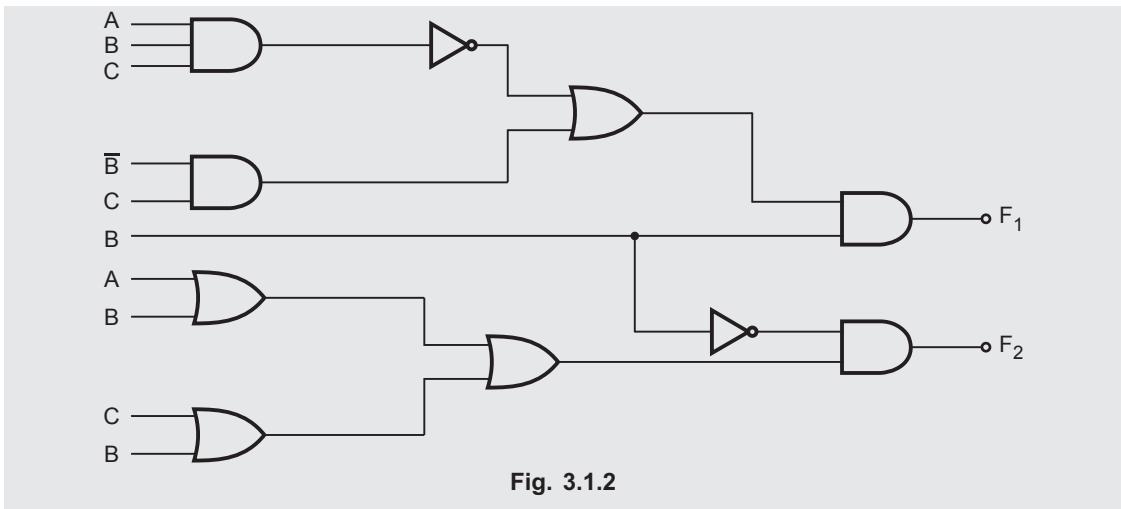
Fig. 3.1.1 Block diagram of a combinational circuit

3.1.1 Analysis Procedure

The analysis of a combinational circuit is the procedure by which we can determine the function that the circuit implements. In this procedure from the given circuit diagram we have to obtain a set of Boolean functions for outputs of circuit, a truth table, or a possible explanation of the circuit operation. Let us see the procedure to determine the Boolean functions for outputs of circuits from the given circuit.

1. First make sure that the given circuit is combinational circuit and not the sequential circuit. The combinational circuit has logic gates with no feedback path or memory elements.
2. Label all gate outputs that are a function of input variables with arbitrary symbols, and determine the Boolean functions for each gate output.
3. Label the gates that are a function of input variables and previously labeled gates and determine the Boolean function for them.
4. Repeat the step 3 until the Boolean function for outputs of the circuit are obtained.
5. Finally, substituting previously defined Boolean functions, obtain the output Boolean functions in terms of input variables.

Example 3.1.1 Obtain the Boolean functions for outputs of the given circuit.



Solution : After labeling the gate outputs, the Boolean functions at the output of each gate are as shown in the Fig. 3.1.3

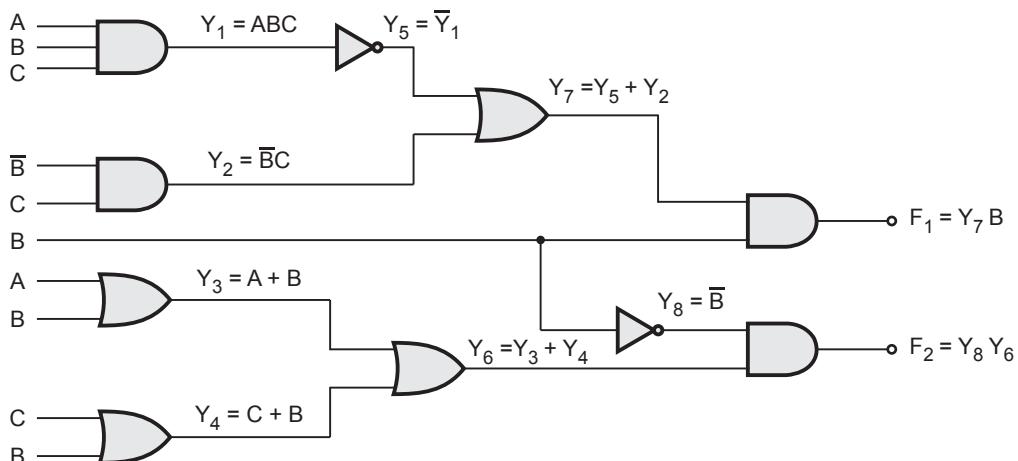


Fig. 3.1.3

$$F_1 = Y_7 B = (\bar{Y}_1 + Y_2) B = (\bar{A}\bar{B}\bar{C} + \bar{B}C) B = [(\bar{A} + \bar{B} + \bar{C}) + \bar{B}C] B = (\bar{A} + \bar{C}) B$$

$$F_2 = Y_8 Y_6 = \bar{B}(Y_3 + Y_4) = \bar{B}[(A + B) + (C + B)] = \bar{B}(A + C)$$

Once the Boolean functions of outputs of circuit are known we can easily determine the truth table using following steps :

1. Determine the number of input variables in the circuit. For n inputs, list the binary numbers from 0 to $2^n - 1$ in a table.
2. Determine the outputs of selected gates for all input combinations.
3. Obtain the truth table for the outputs of those gates that are a function of previously defined values.

4. Repeat step 3 until we get truth table for final outputs.

The following table illustrates process of determining truth table for circuit given in the Fig. 3.1.3.

Note : It is important to note that, from Boolean functions we can directly determine the truth table for outputs of the circuit. However, during testing of the circuit we may require the truth table for output of each gate.

A	B	C	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇	Y ₈	F ₁	F ₂
0	0	0	0	0	0	0	1	0	1	1	0	0
0	0	1	0	1	0	1	1	1	1	1	0	1
0	1	0	0	0	1	1	1	1	1	0	1	0
0	1	1	0	0	1	1	1	1	1	0	1	0
1	0	0	0	0	1	0	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	1	1	0	1
1	1	0	0	0	1	1	1	1	1	0	1	0
1	1	1	1	0	1	1	0	1	0	0	0	0

Truth table for given circuit diagram

Example 3.1.2 Consider the combinational circuit shown in Fig. 3.1.4.

- Derive the Boolean expressions for T_1 through T_4 . Evaluate the outputs F_1 and F_2 as a function of the four inputs.
- List the truth table with 16 binary combinations of the four input variables. Then list the binary values for T_1 through T_4 and outputs F_1 and F_2 in the table.
- Plot the output Boolean function obtained in part (ii) on maps and show that simplified Boolean expressions are equivalent to the ones obtained in part (i)

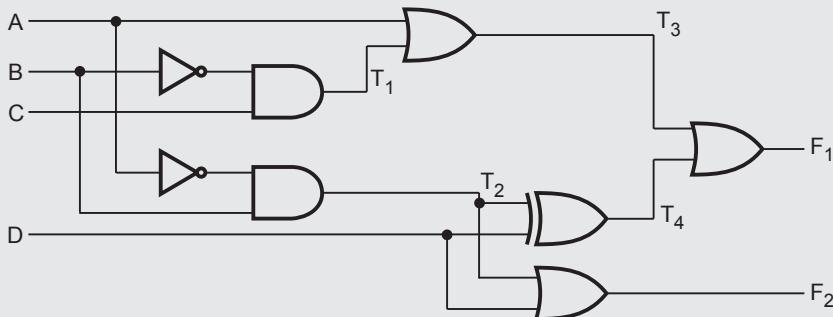


Fig. 3.1.4

$$\text{Solution : i) } T_1 = \overline{BC} \quad T_2 = \overline{AB} \quad T_3 = A + \overline{BC}$$

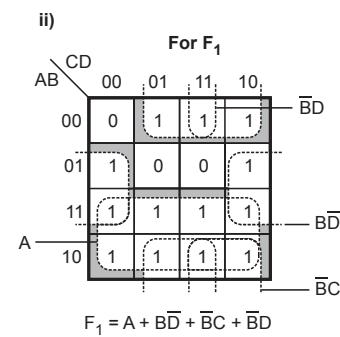
$$T_4 = \overline{AB} \oplus D = \overline{AB}D + \overline{AB}\overline{D} = (A + \overline{B})D + \overline{A}B\overline{D} = AD + \overline{BD} + \overline{A}B\overline{D}$$

$$\begin{aligned} F_1 &= A + \overline{BC} + AD + \overline{BD} + \overline{A}B\overline{D} = A(1+D) + \overline{BC} + \overline{BD} + \overline{A}B\overline{D} \\ &= A + \overline{BC} + \overline{BD} + \overline{A}B\overline{D} = A + \overline{BD} + \overline{BC} + \overline{BD} \end{aligned}$$

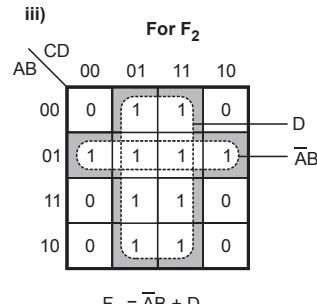
$$F_2 = \overline{AB} + D$$

The simplified Boolean expressions are equivalent to the ones obtained in part (i).

A	B	C	D	T ₁	T ₂	T ₃	T ₄	F ₁	F ₂
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	1
0	0	1	0	1	0	1	0	1	0
0	0	1	1	1	0	1	1	1	1
0	1	0	0	0	1	0	1	1	1
0	1	0	1	0	1	0	0	0	1
0	1	1	0	0	1	0	1	1	1
0	1	1	1	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	1	1	1	1
1	0	1	0	1	0	1	0	1	0
1	0	1	1	1	0	1	1	1	1
1	1	0	0	0	0	1	0	1	0
1	1	0	1	0	0	1	1	1	1
1	1	1	0	0	0	1	0	1	0
1	1	1	1	0	0	1	1	1	1



(a)



(b)

Fig 3.1.5

3.1.2 Design Procedure

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

1. The problem definition.
2. The determination of number of available input variables and required output variables.
3. Assigning letter symbols to input and output variables.
4. The derivation of truth table indicating the relationships between input and output variables.
5. Obtain simplified Boolean expression for each output.
6. Obtain the logic diagram.

Illustrative Example

Example 3.1.3 Design a combination logic circuit with three input variables that will produce a logic 1 output when more than one input variables are logic 1.

Solution :

Step 1 : Derive the truth table for given statement.

Given problem specifies that there are three input variables and one output variable. We assign A, B and C letter symbols to three input variables and assign Y letter symbol to one output variable. The relationship between input variables and output variable can be tabulated as shown in truth Table 3.1.1.

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

Table 3.1.1 Truth table

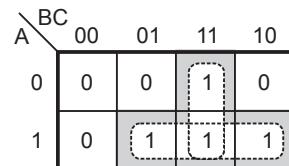


Fig. 3.1.6 K-map simplification

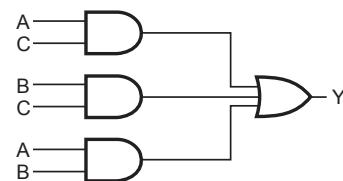


Fig. 3.1.7 Logic diagram

Example for Practice

Example 3.1.4 The inputs to a circuit are the 4 bits of the binary number $D_3D_2D_1D_0$. The circuit produces a 1 if and only if all of the following conditions hold.
 1) MSB is '1' or any of the other bits are a '0'.
 2) D_2 is a 1 or any of the other bits are a '0'. 3) Any of the 4 bits are a 0.
 Obtain a minimal expression for the output. [Ans. : $Y = \overline{D}_1 + \overline{D}_0 + \overline{D}_3\overline{D}_2$]

Review Questions

- What is a combinational logic ?
- Explain the analysis procedure for combinational circuits.
- Explain the design procedure for combinational circuits.

3.2 Code Converter

SPPU : Dec.-10,12,13,17, May-12,13,15,16

There is a wide variety of binary codes used in digital systems. Some of these codes are binary-coded-decimal (BCD), Excess-3, Gray and so on. Many times it is required to convert one code to another.

Let us see the procedure to design code converters :

Step 1 : Write the truth table showing the relationship between input code and output code.

Step 2 : For each output code bit determine the simplified Boolean expression using K-map.

Step 3 : Realize the code converter using logic gates.

Illustrative Examples

Example 3.2.1 Design a 4-bit binary to BCD converter.

Solution :

Step 1 : Form the truth table relating binary and BCD code.

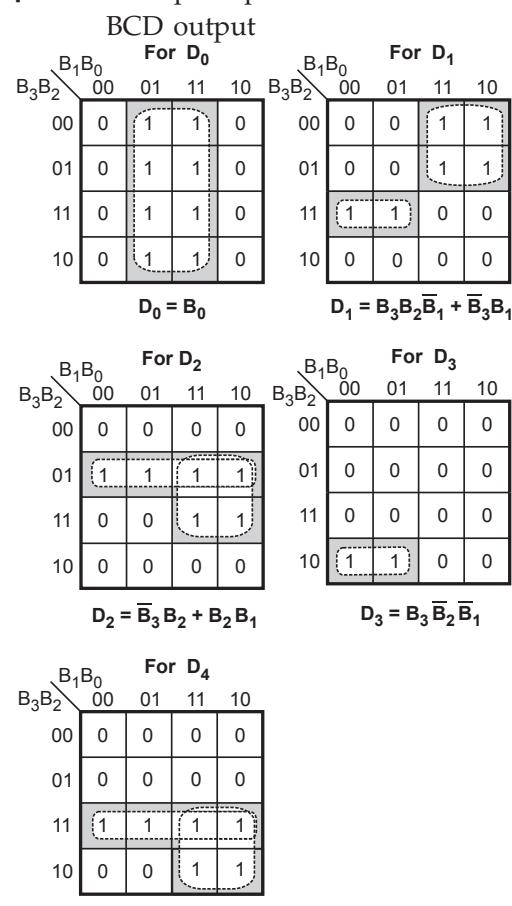
Input code : Binary code : $B_3\ B_2\ B_1\ B_0$ (B_0 LSB)

Output code : BCD (Decimal) code : $D_4\ D_3\ D_2\ D_1\ D_0$ (D_0 LSB)

Binary code				BCD code				
B_3	B_2	B_1	B_0	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

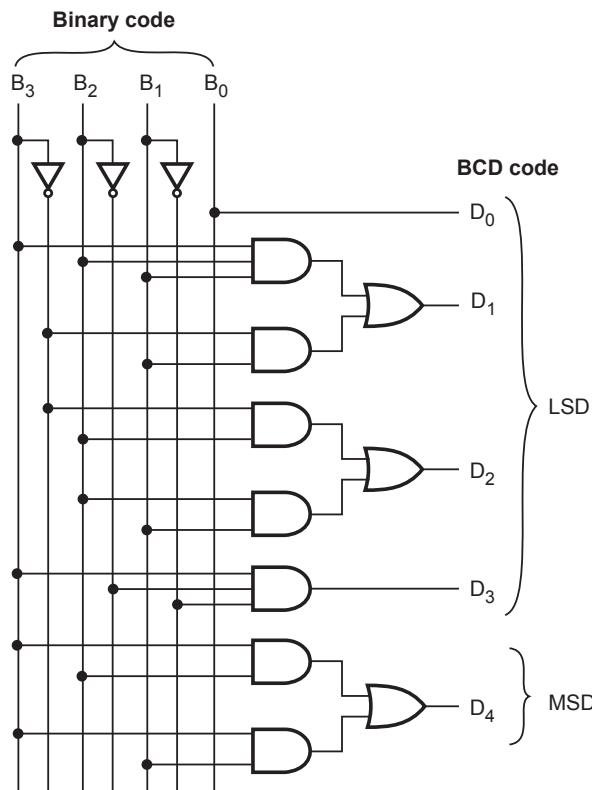
Table 3.2.1 Truth table for binary to BCD conversion

Step 2 : K-map simplification for each BCD output



$$D_4 = B_3B_2 + B_3B_1$$

Fig. 3.2.1

Step 3 : Realization of code converter**Fig. 3.2.2 Binary to BCD converter**

Example 3.2.2 Design a logic circuit to convert the 8421 BCD to Excess-3 code.

SPPU : Dec.-12,13, May-13,16, Marks 8

Solution : Step 1 : Form the truth table relating BCD and Excess-3 code

Excess-3 code is a modified form of a BCD number. The Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as shown in Table 3.2.2.

Decimal	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
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

Table 3.2.2

Input code : BCD code : D₃ D₂ D₁ D₀ (D₀ LSB)

Output code : Excess-3 code : $E_3\ E_2\ E_1\ E_0$ (E_0 LSB)

Step 2 : K-map simplification for each Excess-3 code output.

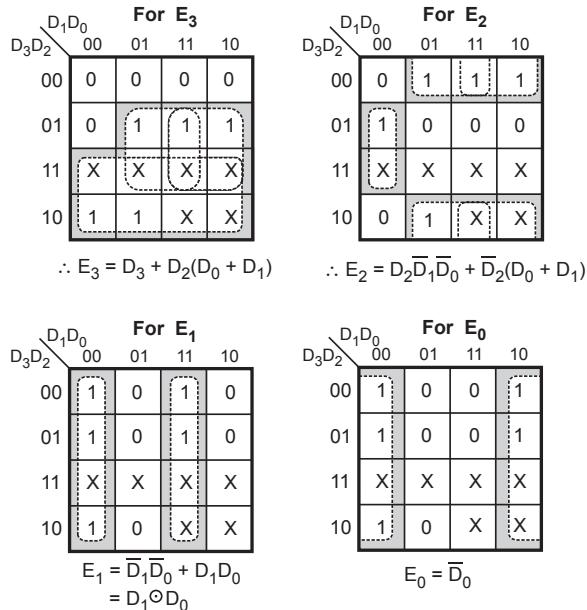


Fig. 3.2.3

Step 3 : Realization of code converter.

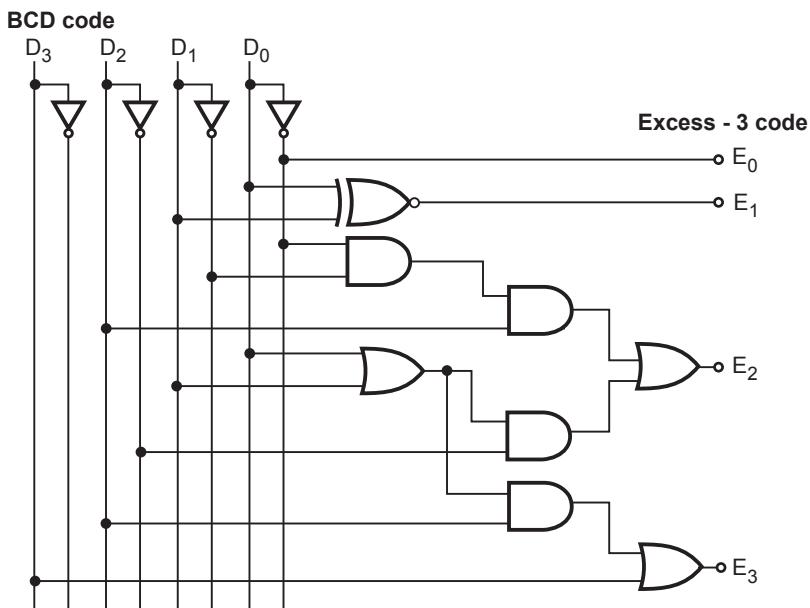


Fig. 3.2.4 BCD to excess-3 code converter

Example 3.2.3 Design a logic circuit to convert BCD to gray code.

Solution : Step 1 : Form the truth table relating BCD and gray code.

Input code : BCD code : $D_3 D_2 D_1 D_0$ (D_0 LSB)

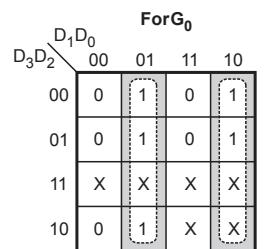
Output code : Gray code : $G_3 G_2 G_1 G_0$ (G_0 LSB)

Table 3.2.3 shows truth table for BCD to gray code converter.

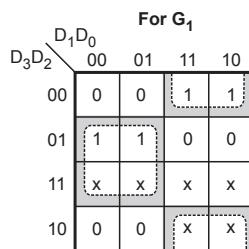
BCD code				Gray code			
D_3	D_2	D_1	D_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

Table 3.2.3 Truth table for BCD to gray code converter

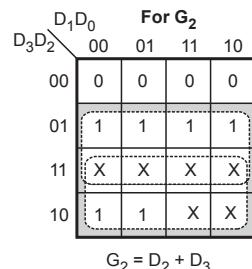
Step 2 : K-map simplification



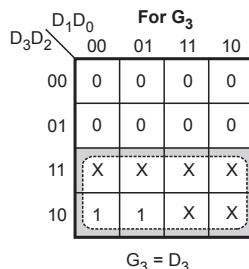
$$G_0 = \overline{D}_1 D_0 + D_1 \overline{D}_0 \\ = D_1 \oplus D_0$$



$$G_1 = D_2 \overline{D}_1 + \overline{D}_2 D_1 \\ = D_2 \oplus D_1$$



$$G_2 = D_2 + D_3$$



$$G_3 = D_3$$

Step 3 : Realization of code converter

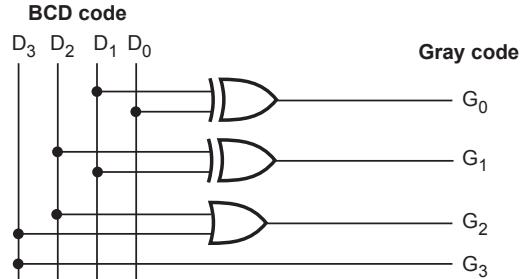


Fig. 3.2.5

Fig. 3.2.6 BCD to gray code converter

Example 3.2.4 Design and implement a 8421 to Gray code converter. Realize the converter using only NAND gates.

SPPU : May-15, Dec.-17, Marks 4

Solution : Step 1 : Form the Truth table relating 8421 binary code and Gray code

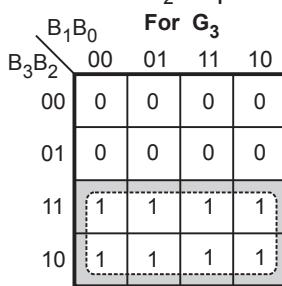
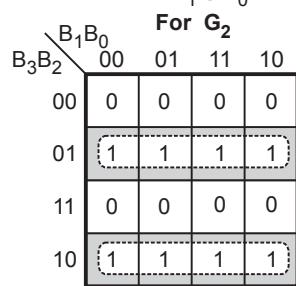
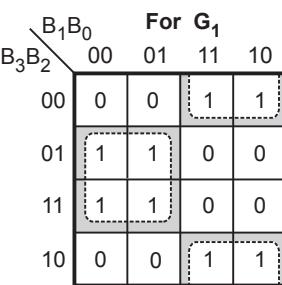
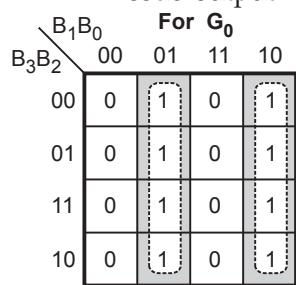
Input code : Binary code : $B_3 B_2 B_1 B_0$

Output code : Gray code : $G_3 G_2 G_1 G_0$

Decimal	Binary code				Gray code			
	B ₃	B ₂	B ₁	B ₀	G ₃	G ₂	G ₁	G ₀
0	0	0	0	0	0	0	0	0
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

Table 3.2.4

Step 2 : K-map simplification for each gray code output



$$G_2 = \overline{B}_3B_2 + B_3\overline{B}_2 = B_3 \oplus B_2$$

Fig. 3.2.7

Step 3 : Realization of code converter using XOR-gates

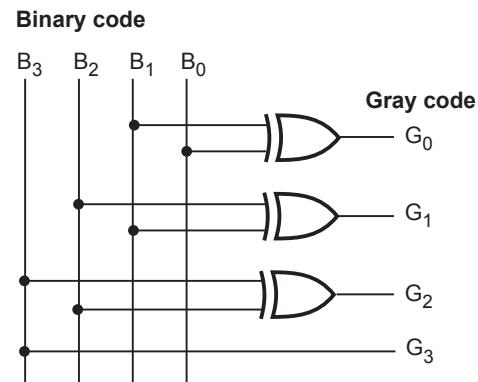
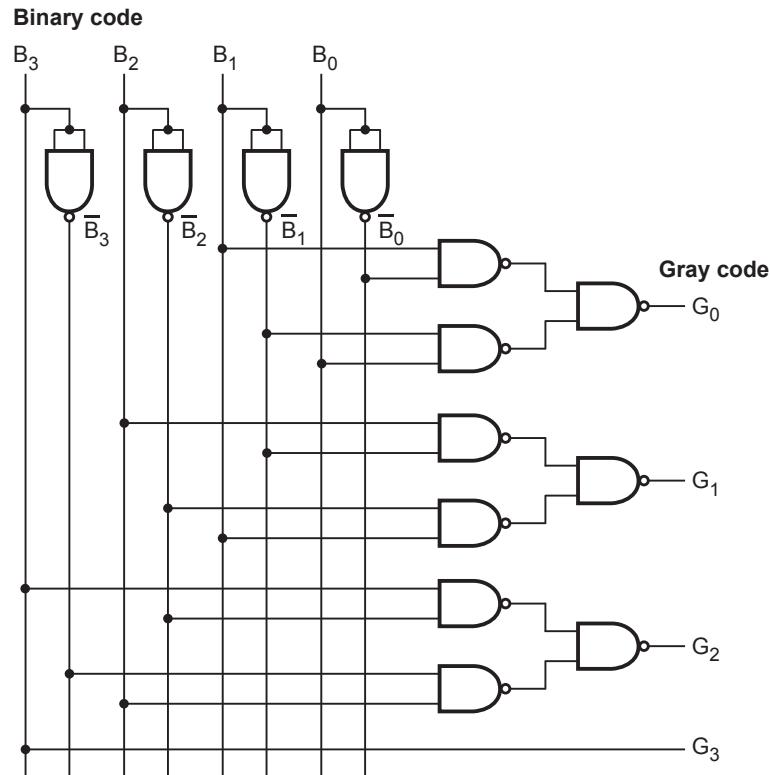


Fig. 3.2.8 Binary to gray code converter

Step 4 : Realization of code converter using NAND gates

For this converter we have derived the Boolean expressions for each gray code output in the sum of product (SOP) form. We can implement SOP expression using AND-OR logic or NAND-NAND logic. Let us see the implementation of code converter using NAND-NAND logic.

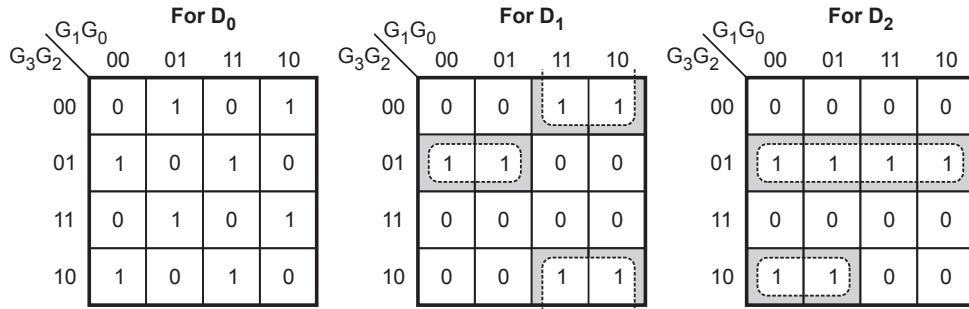

Fig. 3.2.9 Binary to Gray code converter
Example 3.2.5 Design a Gray to BCD code converter.

Solution : The Table 3.2.5 shows truth table for gray to BCD code converter.

Gray code				BCD code				
G ₃	G ₂	G ₁	G ₀	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	1	1	0	0	0	1	0	0
0	1	1	1	0	0	1	0	1

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

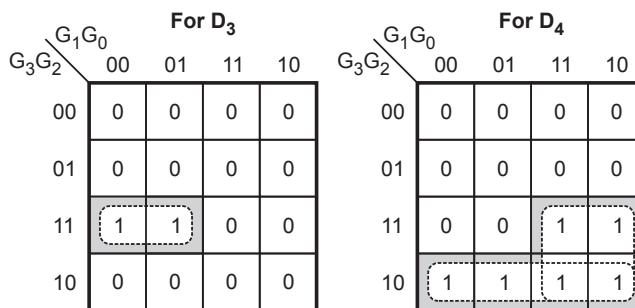
Table 3.2.5 Gray to BCD code converter

K-map Simplification

$$\begin{aligned}
 D_0 &= \overline{G}_3\overline{G}_2\overline{G}_1G_0 + \overline{G}_3\overline{G}_2G_1\overline{G}_0 + \\
 &\quad \overline{G}_3G_2\overline{G}_1\overline{G}_0 + \overline{G}_3G_2G_1\overline{G}_0 + \\
 &\quad G_3\overline{G}_2\overline{G}_1G_0 + G_3G_2G_1\overline{G}_0 + \\
 &\quad G_3\overline{G}_2\overline{G}_1\overline{G}_0 + G_3\overline{G}_2G_1G_0 \\
 &= \overline{G}_3\overline{G}_2(G_1 \oplus G_0) + \overline{G}_3G_2(\overline{G}_1 \oplus G_0) + \\
 &\quad G_3G_2(G_1 \oplus G_0) + G_3\overline{G}_2(\overline{G}_1 \oplus G_0) \\
 &= (\overline{G}_3 \oplus G_2)(G_1 \oplus G_0) + (G_3 \oplus G_2)(\overline{G}_1 \oplus G_0) \\
 &= G_3 \oplus G_2 \oplus G_1 \oplus G_0
 \end{aligned}$$

$$D_1 = \overline{G}_3G_2\overline{G}_1 + \overline{G}_2G_1$$

$$D_2 = \overline{G}_3G_2 + G_3\overline{G}_2\overline{G}_1$$



$$D_3 = G_3G_2\overline{G}_1$$

$$D_4 = G_3\overline{G}_2 + G_3G_1$$

Fig. 3.2.10

Logic Diagram

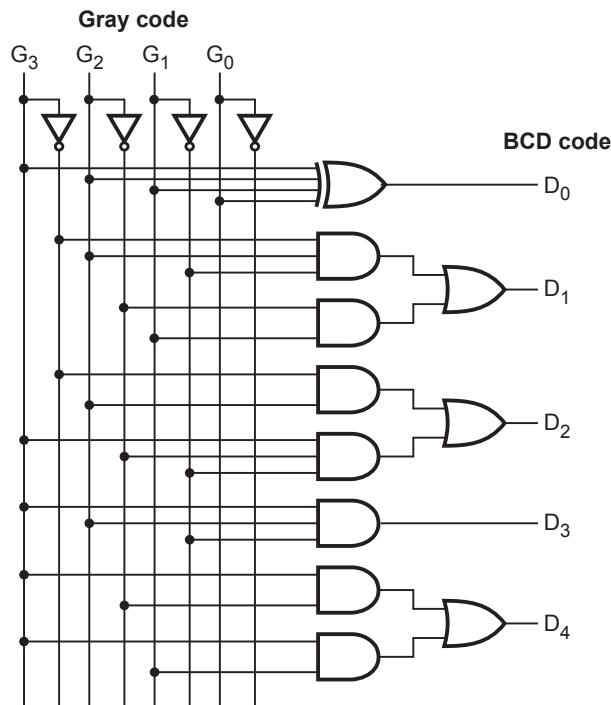


Fig. 3.2.11

Example 3.2.6 Design a 3-bit excess 3 to 3-bit BCD code converter using logic gates.

SPPU : May-15, Marks 8

Solution :

E ₂	E ₁	E ₀	B ₂	B ₁	B ₀
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	0	0

K-map simplification

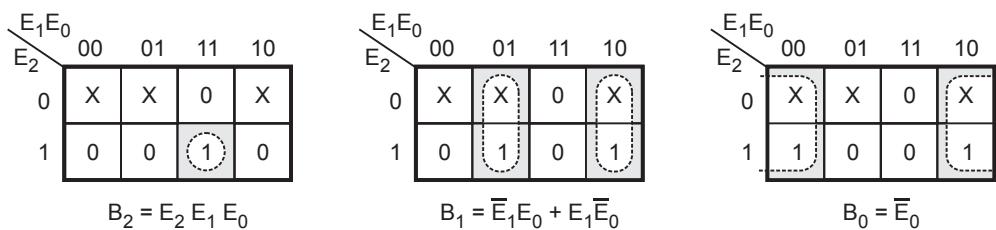


Fig. 3.2.12

Logic diagram

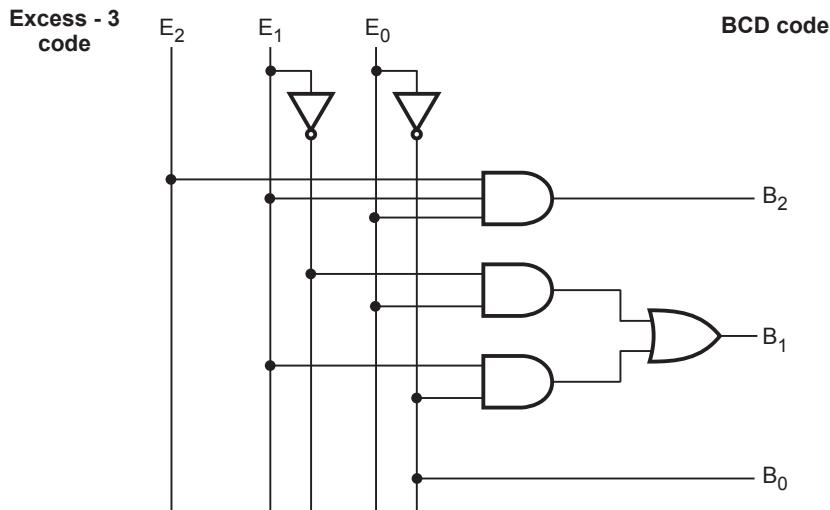


Fig. 3.2.13

Examples for Practice

Example 3.2.7 Design a logic circuit to convert excess-3 code to BCD code.

$$\begin{array}{ll}
 \text{(Ans. : } D_0 = \bar{E}_0, & D_1 = E_1 \oplus E_0 \\
 D_2 = \bar{E}_2 \bar{E}_1 + E_2 E_1 E_0 + \bar{E}_2 \bar{E}_0 & D_3 = E_3 E_2 + E_3 E_1 E_0
 \end{array}$$

Example 3.2.8 Design a logic circuit to convert gray code to binary code.

$$\begin{array}{ll}
 \text{(Ans. : } B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0 & B_1 = G_3 \oplus G_2 \oplus G_1 \\
 B_2 = G_3 \oplus G_2 & B_3 = G_3
 \end{array}$$

Review Questions

1. Explain the design procedure of code converter with the help of example.
2. Enlist various code conversion methods. SPPU : Dec.-10, Marks 2
3. Design 4-bit binary to gray code converter. State the applications of gray code.
(Refer example 3.2.4) SPPU : May-12, Marks 8
4. Design a 4-bit BCD to excess-3 code converter circuit using minimum number of logic gates.
(Refer example 3.2.2) SPPU : Dec.-13, Marks 6

3.3 Adders

SPPU : May-07,14, Dec.-15,18

Digital computers perform various arithmetic operations. The most basic operation, no doubt, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely,

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10_2$$

The first three operations produce a sum whose length is one digit, but when the last operation is performed sum is two digits. The higher significant bit of this result is called a **carry**, and lower significant bit is called **sum**. The logic circuit which performs this operation is called a **half-adder**. The circuit which performs addition of three bits (two significant bits and a previous carry) is a **full-adder**.

3.3.1 Half Adder

The half-adder operation needs two binary inputs : augend and addend bits; and two binary outputs : sum and carry. The truth table shown in Table 3.3.1 gives the relation between input and output variables for half-adder operation.

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

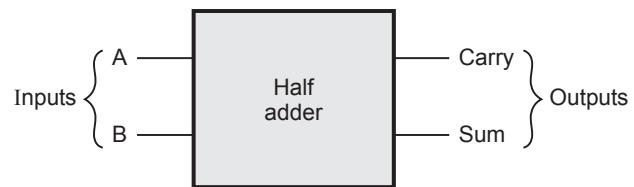
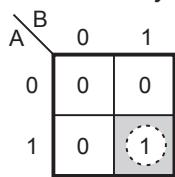


Fig. 3.3.1 Block schematic of half-adder

Table 3.3.1 Truth table for half-adder

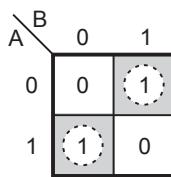
K-map simplification for carry and sum

For carry



$$\text{Carry} = AB$$

For sum



$$\begin{aligned} \text{Sum} &= \bar{A}\bar{B} + \bar{A}B \\ &= A \oplus B \end{aligned}$$

Fig. 3.3.2 Maps for half-adder

Logic diagram

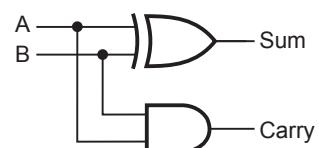


Fig. 3.3.3 Logic diagram for half-adder

Limitations of Half-Adder :

In multidigit addition we have to add two bits along with the carry of previous digit addition. Effectively such addition requires addition of three bits. This is not possible with half-adder. Hence half-adders are not used in practice.

Example 3.3.1 Draw half adder using NAND gates.

Solution : For half adder :

$$\begin{aligned}\text{Sum} &= A\bar{B} + \bar{A}B \quad C_{\text{out}} = AB \\ &= \overline{\overline{A}\overline{B}} + \overline{A}\overline{B} \quad = \overline{AB} \\ &= \overline{A}\overline{B} \cdot \overline{A}\overline{B}\end{aligned}$$

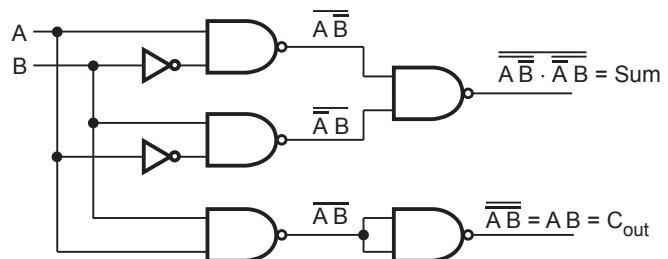


Fig. 3.3.4

3.3.2 Full Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by A and B, represent the two significant bits to be added. The third input C_{in} , represents the carry from the previous lower significant position. The truth table for full-adder is shown in Table 3.3.2.

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 3.3.2 Truth table for full-adder

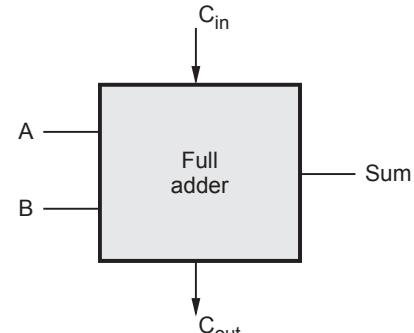
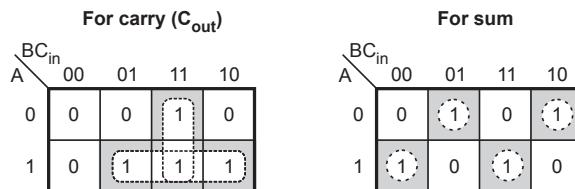


Fig. 3.3.5 Block schematic of full-adder

K-map simplification for carry and sum



$$C_{\text{out}} = AB + A C_{\text{in}} + B C_{\text{in}}$$

$$\text{Sum} = \overline{A} \overline{B} C_{\text{in}} + \overline{A} B \overline{C}_{\text{in}} + A \overline{B} \overline{C}_{\text{in}} + A B C_{\text{in}}$$

Fig. 3.3.6 Maps for full-adder

Logic diagram

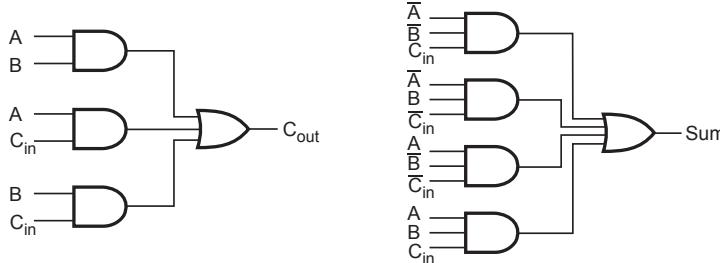


Fig. 3.3.7 Sum of product implementation of full-adder

The Boolean function for sum can be further simplified as follows :

$$\begin{aligned} \text{Sum} &= \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} \bar{C}_{in} + A B C_{in} \\ &= C_{in} (\bar{A} \bar{B} + AB) + \bar{C}_{in} (\bar{A} B + A \bar{B}) \\ &= C_{in} (A \odot B) + \bar{C}_{in} (A \oplus B) \\ &= C_{in} (A \oplus B) + \bar{C}_{in} (A \oplus B) = C_{in} \oplus (A \oplus B) \end{aligned}$$

With this simplified Boolean function circuit for full-adder can be implemented as shown in the Fig. 3.3.8.

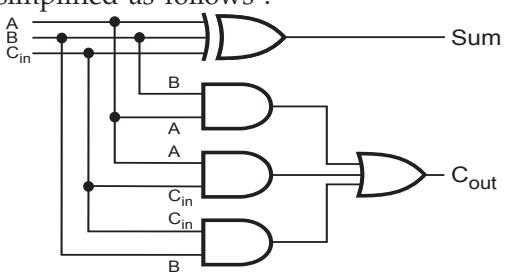


Fig. 3.3.8 Implementation of full-adder

Full adder using two half adders

A full-adder can also be implemented with two half-adders and one OR gate, as shown in the Fig. 3.3.9. The sum output from the second half-adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving

$$\begin{aligned} C_{out} &= AB + A C_{in} + B C_{in} = AB + A C_{in} (B + \bar{B}) + B C_{in} (A + \bar{A}) \\ &= AB + ABC_{in} + A \bar{B} C_{in} + ABC_{in} + A \bar{B} C_{in} = AB (1 + C_{in} + C_{in}) + A \bar{B} C_{in} + \bar{A} BC_{in} \\ &= AB + \bar{A} BC_{in} + \bar{A} BC_{in} = AB + C_{in} (\bar{A} B + AB) = AB + C_{in} (A \oplus B) \end{aligned}$$

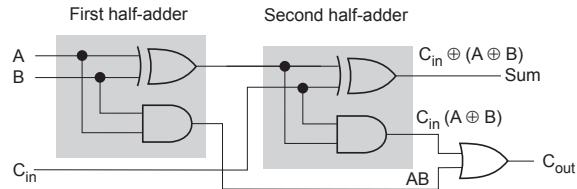


Fig. 3.3.9 Implementation of a full-adder with two half-adders and an OR gate

Review Questions

1. What do you mean by half-adder and full-adder ? How will you implement full-adder using half-adder ? Explain with circuit diagram. **SPPU : May-07, Dec.-18, Marks 6**
2. Design half-adder using only NAND gates.
3. Design full-adder using only NOR gates.
4. What do you mean by half adder and full adder ? How will you implement full adder using half adder ? Draw the circuit diagram. [Refer section 3.3] **SPPU : May-14, Marks 6**
5. What are the full adder's inputs that will produce each of the following outputs ?
 - i) $\Sigma = 0, C_{out} = 0$
 - ii) $\Sigma = 1, C_{out} = 0$
 - iii) $\Sigma = 1, C_{out} = 1$
 - iv) $\Sigma = 0, C_{out} = 1$**SPPU : Dec.-15, Marks 2**

3.4 Subtractors

SPPU : Dec.-08

The subtraction consists of four possible elementary operations, namely,

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ with 1 borrow}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

In all operations, each subtrahend bit is subtracted from the minuend bit. In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed. Just as there are half and full-adders, there are **half** and **full-subtractors**.

3.4.1 Half Subtractor

A half-subtractor is a combinational circuit that subtracts two-bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Let us designate minuend bit as A and the subtrahend bit as B. The result of operation A – B for all possible values of A and B is tabulated in Table 3.4.1.

As shown in the Table 3.4.1, half-subtractor has two input variables and two output variables. The Boolean expression for the outputs of half-subtractor can be determined as follows.

K-map simplification for half-subtractor

For difference

	B	0	1
A	0	0	1
	1	1	0

$$\text{Difference} = \overline{AB} + \overline{AB} \\ = A \oplus B$$

For borrow

	B	0	1
A	0	0	1
	1	0	0

$$\text{Borrow} = \overline{AB}$$

Fig. 3.4.1

		Inputs		Outputs	
		A	B	Difference	Borrow
0	0			0	0
0	1			1	1
1	0			1	0
1	1			0	0

Table 3.4.1 Truth table for half-subtractor

Logic diagram

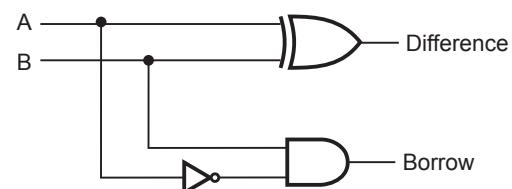


Fig. 3.4.2 Implementation of half-subtractor

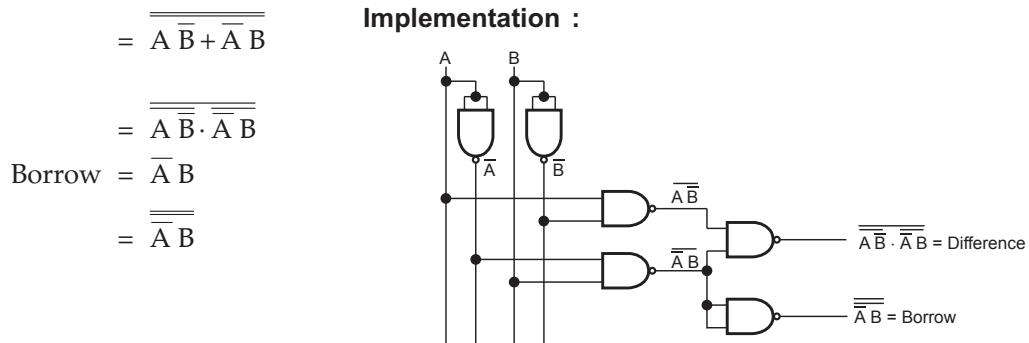
Limitations of half-subtractor :

In multidigit subtraction, we have to subtract two bits along with the borrow of the previous digit subtraction. Effectively such subtraction requires subtraction of three bits. This is not possible with half-subtractor.

Example 3.4.1 Draw half subtractor using NAND gates.

Solution : For half subtractor :

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



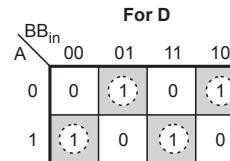
3.4.2 Full-Subtractor

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account borrow of the lower significant stage. This circuit has three inputs and two outputs. The three inputs are A , B and B_{in} denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and B_{out} , represent the difference and output borrow, respectively. The Table 3.4.2 shows the truth table for full-subtractor.

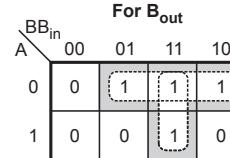
K-map simplification of D and B_{out}

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

Table 3.4.2 Truth table for full-subtractor



$$D = \overline{A}\overline{B}B_{in} + \overline{A}B\overline{B}_{in} + A\overline{B}\overline{B}_{in} + AB\overline{B}_{in}$$



$$B_{out} = \overline{A}B_{in} + \overline{A}B + BB_{in}$$

Fig. 3.4.4 Maps for full-subtractor

Logic diagram

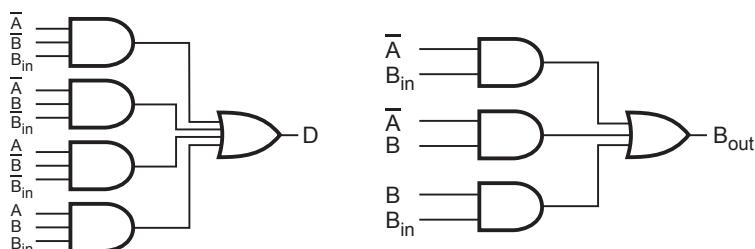


Fig. 3.4.5 Sum of product implementation of full-subtractor

The Boolean function for D (difference) can be further simplified as follows :

$$\begin{aligned} D &= \overline{A} \overline{B} B_{in} + A \overline{B} \overline{B}_{in} + A \overline{B} \overline{B}_{in} + A B B_{in} \\ &= B_{in} (\overline{A} B + A B) + \overline{B}_{in} (\overline{A} B + A \overline{B}) = B_{in} (A \odot B) + \overline{B}_{in} (A \oplus B) \\ &= B_{in} (A \oplus B) + \overline{B}_{in} (A \oplus B) = B_{in} \oplus (A \oplus B) \end{aligned}$$

With this simplified Boolean function circuit for full-subtractor can be implemented as shown in the Fig. 3.4.6

A full subtractor can also be implemented with two half-subtractors and one OR gate, as shown in the Fig. 3.4.7. The difference output from the second half-subtractor is the exclusive-OR of B_{in} and the output of the first half-subtractor, which is same as difference output of full-subtractor.

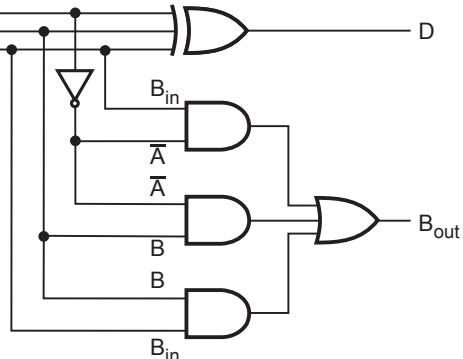


Fig. 3.4.6 Implementation of full-subtractor

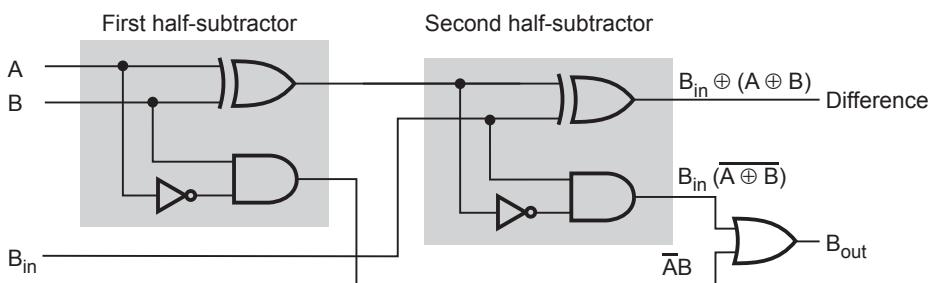


Fig. 3.4.7 Implementation of a full-subtractor with two half-subtractors and an OR gate

The borrow output for circuit shown in Fig. 3.4.7 can be given as,

$$\begin{aligned} B_{out} &= \overline{A} B_{in} + \overline{A} B + B B_{in} \\ &= \overline{A} B_{in} (B + \overline{B}) + \overline{A} B + B B_{in} (A + \overline{A}) \\ &= \overline{A} B B_{in} + \overline{A} \overline{B} B_{in} + \overline{A} B + A B B_{in} + \overline{A} B B_{in} \\ &= \overline{A} B (B_{in} + 1 + B_{in}) + \overline{A} \overline{B} B_{in} + A B B_{in} = \overline{A} B + \overline{A} \overline{B} B_{in} + A B B_{in} \\ &= \overline{A} B + B_{in} (\overline{A} \overline{B} + A B) = \overline{A} B + B_{in} (\overline{A} \oplus B) \end{aligned}$$

This Boolean function is same as borrow out of the full-subtractor. Therefore, we can implement full-subtractor using two half-subtractors and OR gate.

Review Questions

1. Define half-subtractor and full-subtractor.
2. With the help of a circuit diagram explain the half-subtractor.
3. Explain the circuit diagram of full-subtractor.

SPPU : Dec.-08, Marks 8

3.5 Parallel Adder

SPPU : May-06, Dec.-06

We have seen, a single full-adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers with more than one bit, additional full-adders must be employed.

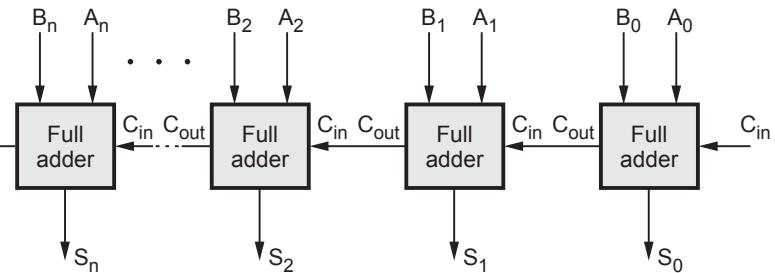


Fig. 3.5.1 Block diagram of n-bit parallel adder

A n-bit, parallel adder can be constructed using number of full adder circuits connected in parallel. Fig. 3.5.1 shows the block diagram of n-bit parallel adder using number of full-adder circuits connected in cascade, i.e. the carry output of each adder is connected to the carry input of the next higher-order adder.

It should be noted that either a half-adder can be used for the least significant position or the carry input of a full-adder is made 0 because there is no carry into the least significant bit position.

Example 3.5.1 Design a 4-bit parallel adder using full-adders.

Solution : Fig. 3.5.2 shows the block diagram for 4-bit adder. Here, for least significant position, carry input of full-adder is made 0.

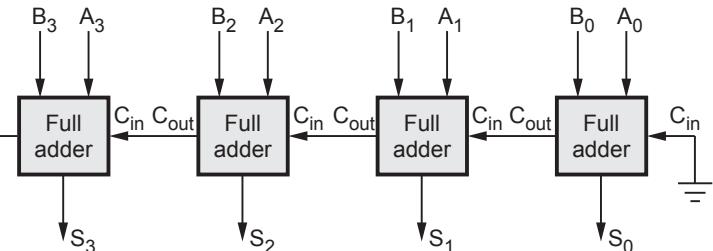


Fig. 3.5.2 Block diagram of 4-bit full-adder

Review Question

- Draw and explain the block diagram of 4-bit parallel adder.

SPPU : May-06, Dec.-06, Marks 4

3.6 Parallel Subtractor

The subtraction of binary numbers can be done most conveniently by means of complements. Remember that the subtraction A-B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry, as shown in the Fig. 3.6.1

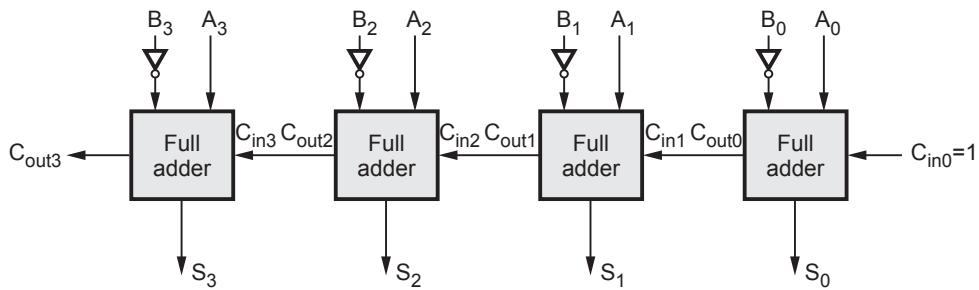


Fig. 3.6.1 4-bit parallel subtractor

Review Question

1. Draw and explain the block diagram of 4-bit parallel adder/subtractor.

3.7 Parallel Adder / Subtractor

The addition and subtraction operations can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full adder,

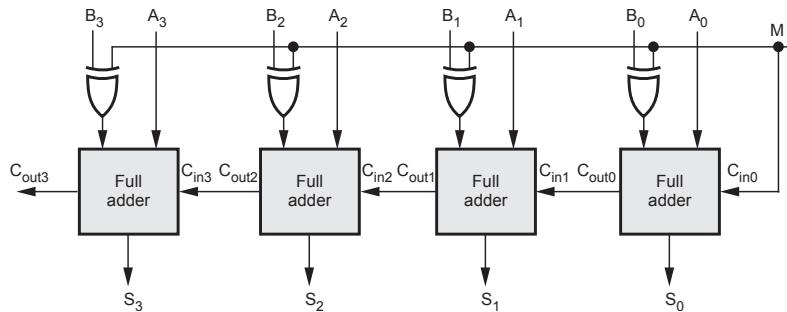


Fig. 3.7.1 4-bit adder-subtractor

as shown in Fig. 3.7.1. The mode input M controls the operation of the circuit. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B . When $M = 0$, we have $B \oplus 0 = B$. The full-adders receive the value of B , the input carry is 0, and the circuit performs A plus B . When $M = 1$, we have $B \oplus 1 = \bar{B}$ and $C_{in0} = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B , i.e. $A - B$.

The parallel adder is ripple carry adder in which the carry output of each full-adder stage is connected to the carry input of the next higher-order stage. Therefore, the sum and carry outputs of any stage cannot be produced until the input carry occurs; this leads to a time delay in the addition process. This delay is known as **carry propagation delay**.

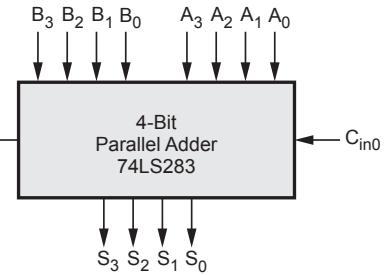
One method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition**. This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated.

Review Question

1. Draw and explain the block diagram of 4-bit parallel adder/subtractor.

3.8 Four-bit Parallel Adder (7483/74283)

Many high-speed adders available in integrated-circuit form utilize the look-ahead carry or a similar technique for reducing overall propagation delays. The most common is a 4-bit parallel adder IC (74LS83/74S283) that contains four interconnected full-adders and the look-ahead carry circuitry needed for high-speed operation. The 7483 and 74283 are a TTL medium scale integrated (MSI) circuit with same pin configuration. Fig. 3.8.1 shows the functional symbol for the 74LS283 4-bit parallel adder. The inputs to this IC are two 4-bit numbers, $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$, and the carry, C_{in0} , into the LSB position. The outputs are the sum bits $S_3 S_2 S_1 S_0$, and the carry, C_{out3} , output of the MSB position.



Two or more parallel adder blocks can be connected (cascaded) to accommodate the addition of larger binary numbers. This is illustrated in example 3.8.1

Fig. 3.8.1 Functional symbol for the 74LS283

Example 3.8.1 Design an 8-bit adder using two 74LS283s.

Solution : Fig. 3.8.2 shows how two 74LS283 adders can be connected to add two 8-bit numbers. The adder on the right adds the four least significant bits of the numbers. The C_{out3} output of this adder is connected as the input carry to the first position of the second adder, which adds the four most significant bits of the numbers. The eight sum outputs represent the resultant sum of the two 8-bit numbers. C_{out7} is the carry out of the last position (MSB) of the second adder. It can be used as an overflow bit or as a carry into another adder stage if large binary numbers are to be handled.

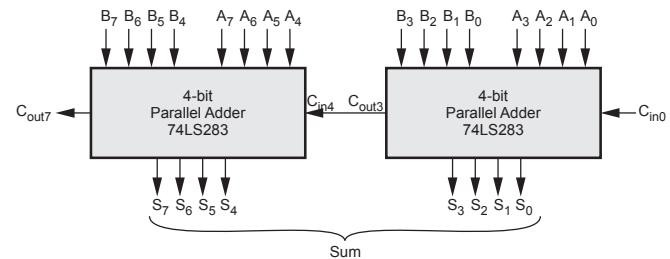


Fig. 3.8.2

Example 3.8.2 Design a 24-bit group ripple adder using 74X283 ICs.

Solution :

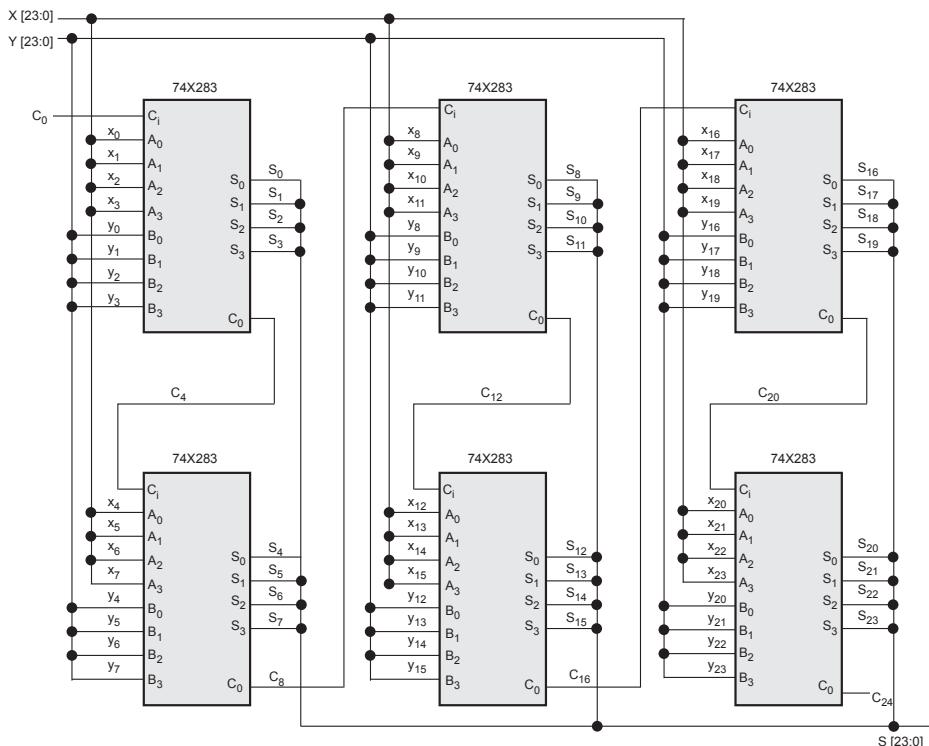


Fig. 3.8.3 A 24-bit group ripple adder using 74X283 ICs

3.9 BCD Adder

SPPU : Dec.-05,06,10,12,14,16, May-05,07,08,10,12

The digital systems handle the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. Here, we will see the implementation of addition of BCD numbers.

To implement BCD adder we require :

- 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add 0110_2 in the sum if sum is greater than 9 or carry is 1.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given truth table.

$Y = 1$ indicates sum is greater than 9. We can put one more term, C_{out} in the above expression to check

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

Table 3.9.1

whether carry is one. If any one condition is satisfied we add 6(0110) in the sum.

With this design information we can draw the block diagram of BCD adder, as shown in the Fig. 3.9.1 (b).

As shown in the Fig. 3.9.1 (b), the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum.

When the output carry is equal to zero (i.e. when sum ≤ 9 and $C_{out} = 0$) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum > 9 or $C_{out} = 1$), binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

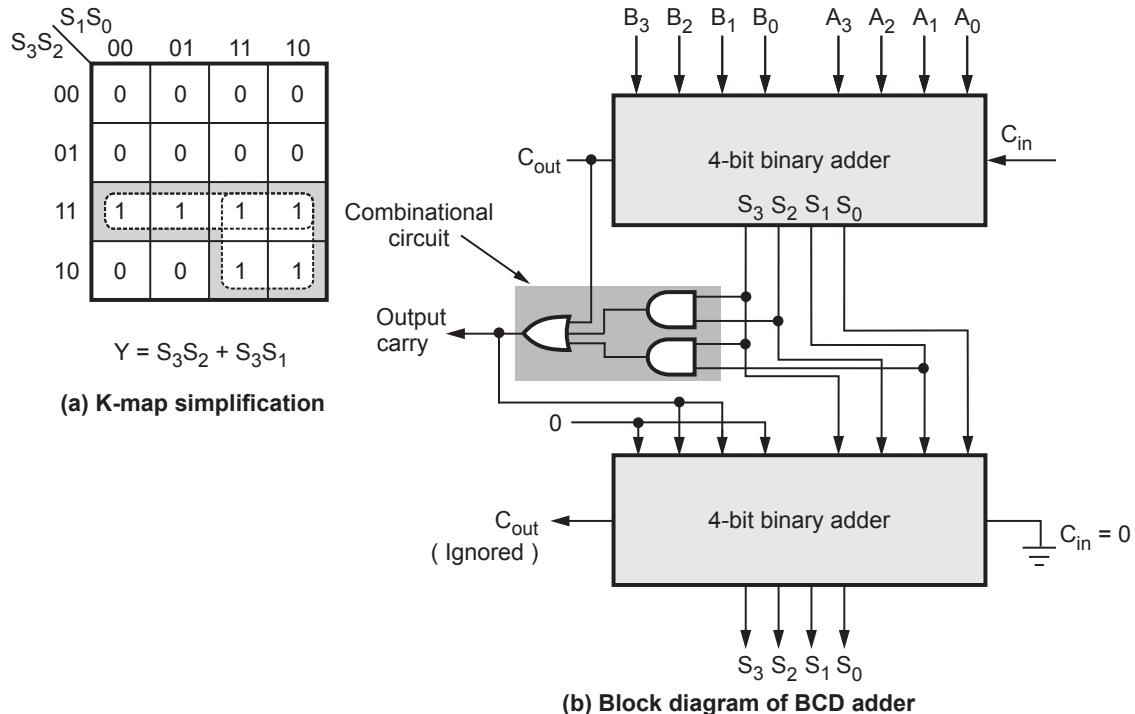


Fig. 3.9.1

The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

Example 3.9.1 Design an 8-bit BCD adder using 4-bit binary adder.

Solution : To implement 8-bit BCD adder we have to cascade two 4-bit BCD adders. In cascade connection carry output of the lower position (digit) is connected as a carry input of the higher position (digit). Fig. 3.9.2 shows the block diagram of 8-bit BCD adder.

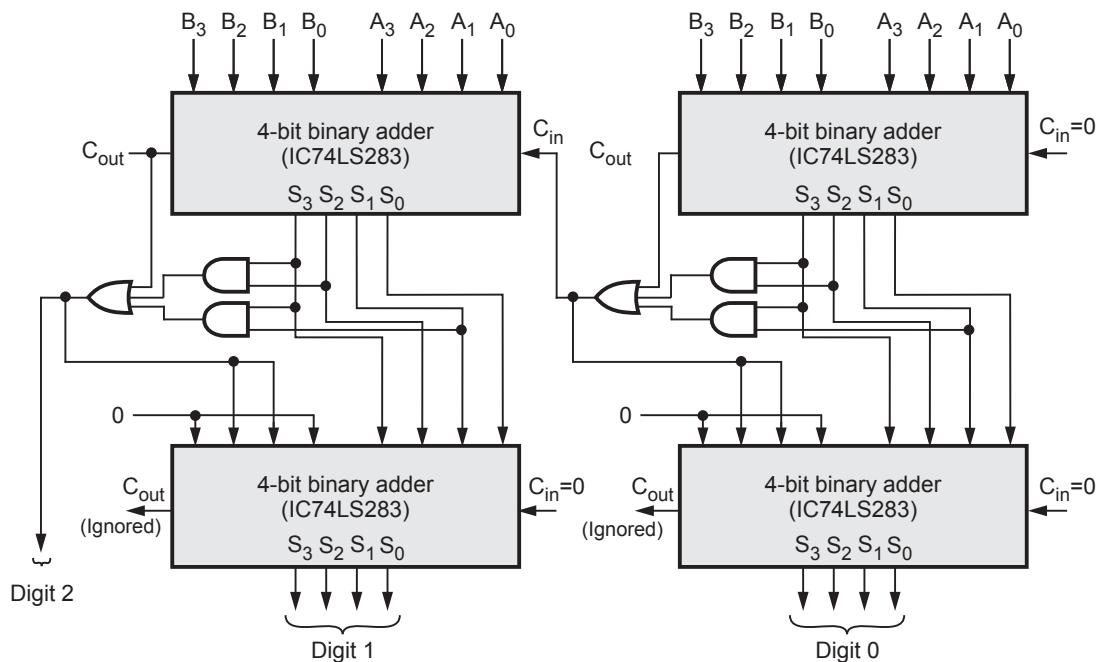


Fig. 3.9.2 8-bit BCD adder using IC 74283

Example 3.9.2 Design a BCD to Excess-3 code converter using binary parallel adder.

Solution :

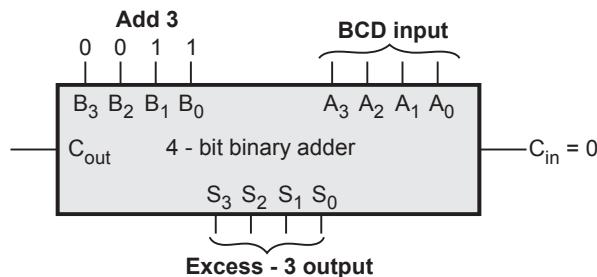


Fig. 3.9.3

Review Questions

1. Explain with suitable example rules for BCD addition and design 1-digit BCD adder using IC 74LS83. **SPPU : Dec.-05,14,16, Marks 6**
2. Draw and explain the basic circuit of single digit BCD adder using IC 7483. How will you make two digit BCD adder ? Explain the logic of the circuit. **SPPU : May-05,12, Marks 8**
3. Draw and explain 4-bit BCD adder using IC 7483. Explain any two BCD addition operations. **SPPU : May-10, Marks 10**

4. How to convert 4-bit binary adder to BCD adder ? Explain with the help of circuit diagram.

SPPU : Dec.-06, May-07, Marks 6

5. How will make 3-digit BCD adder using 4-bit binary adder as a basic building block ? Explain with the help of suitable diagram.

SPPU : May-08, Marks 8

6. Draw and explain 4-bit BCD adder using IC 7483. Also explain with reference to your design addition of $(9+5)_{BCD}$ and $(7+2)_{BCD}$.

SPPU : Dec.-10, Marks 8

7. Draw and explain 4-bit BCD adder using IC7483. Also explain with example addition of numbers with carry.

SPPU : Dec.-12, Marks 8

3.10 Look-Ahead Carry Adder

SPPU : May-06,14,17, Dec.-14,17

The parallel adder discussed in the last paragraph is ripple carry type in which the carry output of each full-adder stage is connected to the carry input of the next higher-order stage. Therefore, the sum and carry outputs of any stage cannot be produced until the input carry occurs; this leads to a time delay in the addition process. This delay is known as *carry propagation delay*, which can be best explained by considering the following addition.

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

Addition of the LSB position produces a carry into the second position. This carry, when added to the bits of the second position (stage), produces a carry into the third position. The latter carry, when added to the bits of the third position, produces a carry into the last position. The key thing to notice in this example is that the sum bit generated in the last position (MSB) depends on the carry that was generated by the addition in the previous positions. This means that, adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in each full-adder. For example, if each full-adder is considered to have a propagation delay of 30 ns, then S_3 will not reach its correct value until 90 ns after LSB carry is generated. Therefore, total time required to perform addition is $90 + 30 = 120$ ns.

Obviously, this situation becomes much worse if we extend the adder circuit to add a greater number of bits. If the adder were handling 16-bit numbers, the carry propagation delay could be 480 ns.

One method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition**. This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions : carry generate and carry propagate.

Consider the circuit of the full-adder shown in Fig. 3.10.1. Here, we define two functions : carry generate and carry propagate.

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

G_i is called a carry generate and it produces on carry when both A_i and B_i are one, regardless of the input carry. P_i is called a carry propagate because it is term associated with the propagation of the carry from C_i to C_{i+1} .

Now the Boolean function for the carry output of each stage can be written as follows.

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$\begin{aligned} C_4 &= G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 C_1) \\ &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1 \end{aligned}$$

From the above Boolean function it can be seen that C_4 does not have to wait for C_3 and C_2 to propagate; in fact C_4 is propagated at the same time as C_2 and C_3 .

The Boolean functions for each output carry are expressed in sum-of product form, thus they can be implemented using AND-OR logic or NAND-NAND logic. Fig. 3.10.2 shows implementation of Boolean functions for C_2 , C_3 and C_4 using AND-OR logic.

Using a look ahead carry generator we can easily construct a 4-bit parallel adder with a look ahead carry scheme. Fig. 3.10.3 shows a 4-bit parallel adder with a look ahead carry scheme. As shown in the Fig. 3.10.3, each sum output requires two exclusive-OR gates. The output of first exclusive-OR gate generates P_i ,

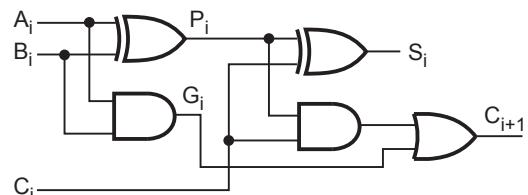


Fig. 3.10.1 Full-adder circuit

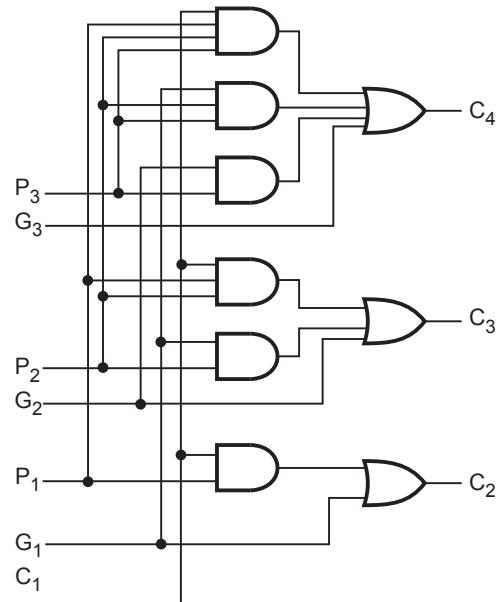


Fig. 3.10.2 Logic diagram of a look ahead carry generator

and the AND gate generates G_i . The carries are generated using look ahead carry generator and applied as inputs to the second exclusive-OR gate. Other inputs to exclusive-OR gate is P_i . Thus second exclusive-OR gate generates sum outputs. Each output each generated after a delay of two levels of gate. Thus outputs S_2 through S_4 have equal propagation delay times.

IC 74182 is a look ahead carry generator. Fig. 3.10.3 shows pin diagram and logic symbol for IC 74182. As shown in the logic symbol, the 74182 carry look ahead generator accepts up to four pairs of active low carry propagate ($\bar{P}_0, \bar{P}_1, \bar{P}_2, \bar{P}_3$) and carry generate ($\bar{G}_0, \bar{G}_1, \bar{G}_2, \bar{G}_3$) signals and an active high carry input (C_n) and provides anticipated active high carries ($C_{n+x}, C_{n+y}, C_{n+z}$) across four groups of binary adders. The 74182 also has active low carry propagate (\bar{P}) and carry generate (\bar{G}) outputs which may be used for further levels of look ahead.

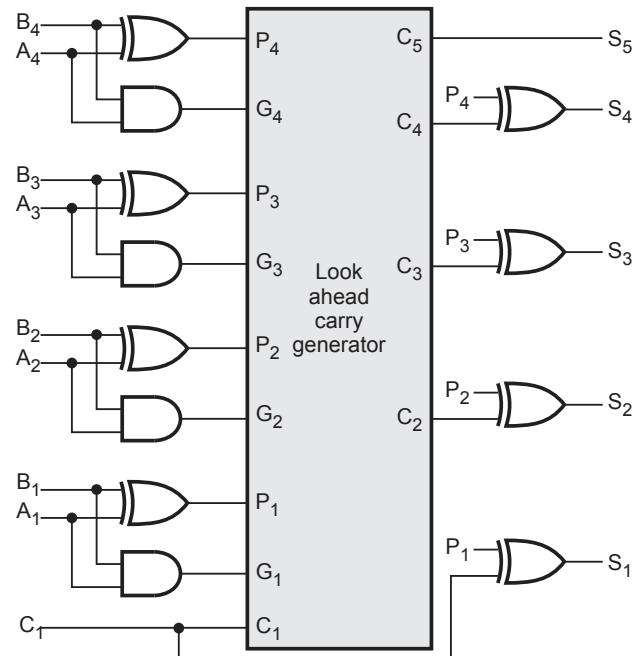
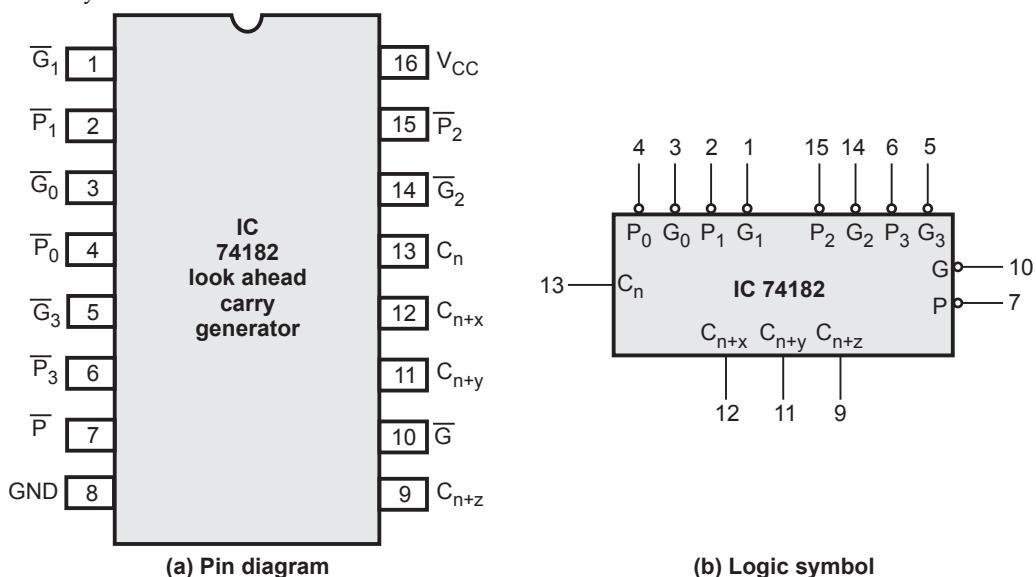


Fig. 3.10.3 4-bit parallel adder with look ahead carry generator



The logic equations provided at the outputs of 74182 are :

$$C_{n+x} = G_0 + P_0 C_n$$

$$C_{n+y} = G_1 + P_1 G_0 + P_1 P_0 C_n$$

$$C_{n+z} = G_2 + P_2 G_1 + P_2 P_1 G_0$$

$$\bar{G} = \overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0}$$

$$\bar{P} = \overline{P_3 P_2 P_1 P_0}$$

Example 3.10.1 Show the construction of 4-bit parallel adder using IC 74182.

Solution : Fig. 3.10.5 shows the construction of 4-bit parallel adder using IC 74182. The last carry output can be generated using \bar{P} and \bar{G} outputs of IC 74182, as shown in the Fig. 3.10.5

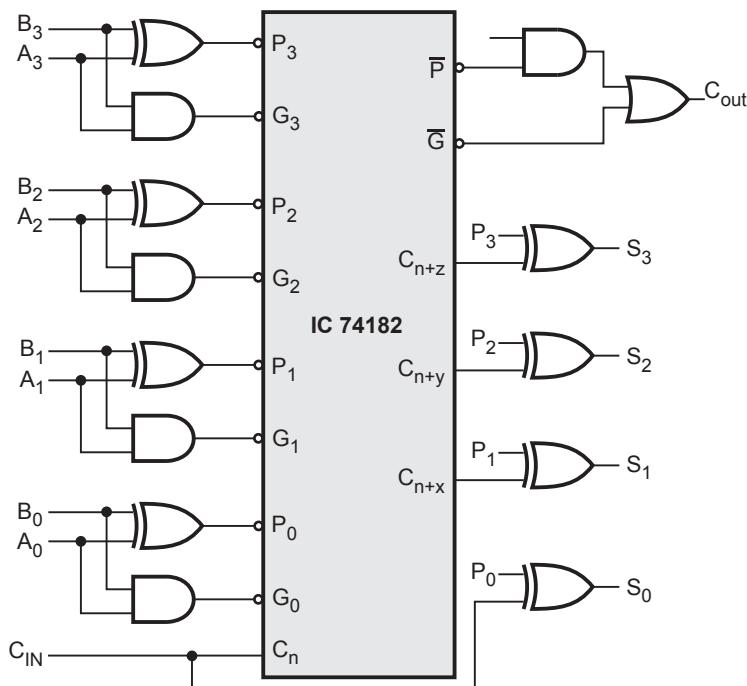


Fig. 3.10.5 4-bit parallel adder using IC 74182

Example 3.10.2 Construct the look ahead carry generator to accommodate higher word size.

Solution : Look ahead carry generators can be cascaded to increase the word size in multiples of 4-bits. Fig. 3.10.6 shows the cascading two look ahead carry generators (IC 74182s) to get word size of 8-bits. (Refer Fig. 3.10.6 on next page)

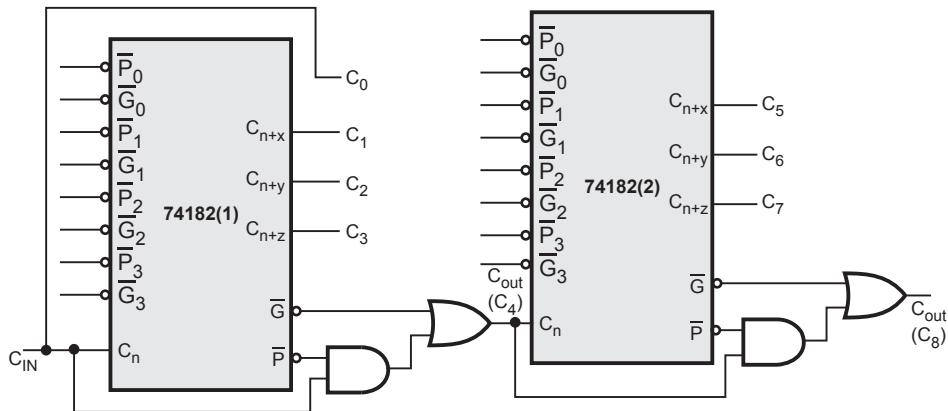


Fig. 3.10.6 Cascading two look ahead carry generators

Review Questions

1. Explain look ahead carry generator.
2. How will you generate look ahead carry for your 4-bit adder circuit. **SPPU : May-06, Marks 6**
3. Explain in detail look ahead carry generator. **SPPU : May-14,17, Dec.-14,17, Marks 6**

3.11 Multiplexers (MUX) SPPU : May-05,10,11,12,13,17,Dec.-10,12,16,17,18,19

In digital systems, many times it is necessary to select single data line from several data-input lines, and the data from the selected data line should be available on the output. The digital circuit which does this task is a multiplexer.

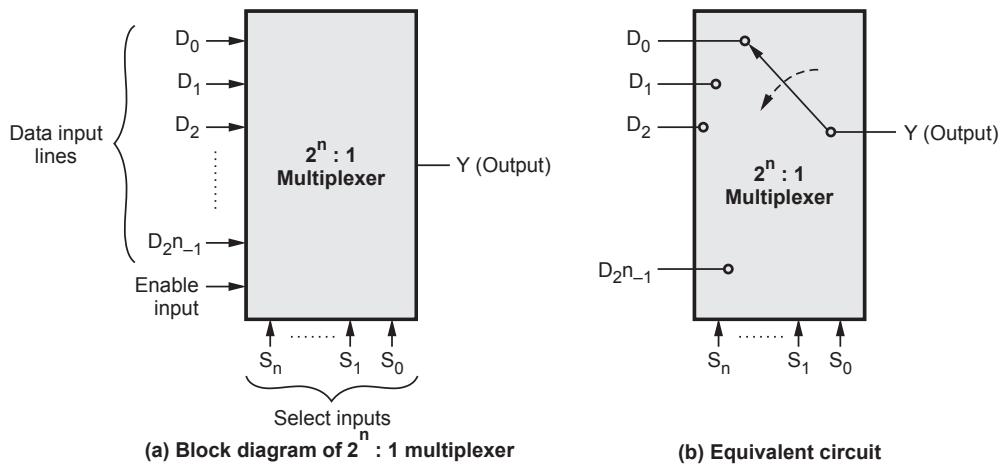


Fig. 3.11.1

It is a digital switch. It allows digital information from several sources to be routed onto a single output line, as shown in the Fig. 3.11.1. The basic multiplexer has several data-input lines and a single output line. The selection of a particular input line is controlled by a set of selection lines. Since multiplexer selects one of the input and routes it to output, it is also known as **data selector**. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. Therefore, multiplexer is '**many into one**' and it provides the digital equivalent of an analog selector switch.

3.11.1 2 : 1 Multiplexer

Fig. 3.11.2 (a) shows 2 : 1 multiplexer. D_0 is applied as an input to one AND gate and D_1 is applied as an input to another AND gate. Enable input is applied to both gates as one input. Selection line S is connected as second input to second AND gate. An inverted S is applied as second input to first AND gate. Outputs of both AND gates are applied as inputs to OR gate.

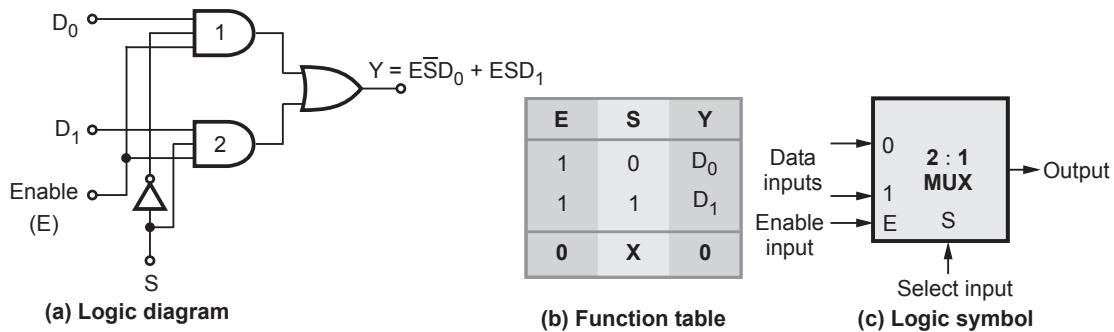


Fig. 3.11.2

Working

When $E = 0$, output is 0, i.e. $Y = 0$ irrespective of any input condition. When $E = 1$ the circuit works as follows :

When $S = 0$, the inverted S , that is 1 gets applied as second input to first AND gate. Since S is applied directly as input to second AND gate; its output goes zero irrespective of first input. Since the second input of first AND gate is 1, its output is equal to its first input, that is D_0 . Hence $Y = D_0$.

Exactly opposite is the case when $S = 1$. In this case, second AND gate output is equal to its first input D_1 and first AND gate output is 0. Hence $Y = D_1$. Both these cases are summarized in truth table shown in Fig. 3.11.2 (b).

Deriving realization expression

The Table 3.11.1 shows the truth table for 2 : 1 multiplexer. From the truth table it is clear that $Y = 1$ when $E \bar{S} D_0 = 1$ or $E S D_1 = 1$ as indicated by shaded rows.

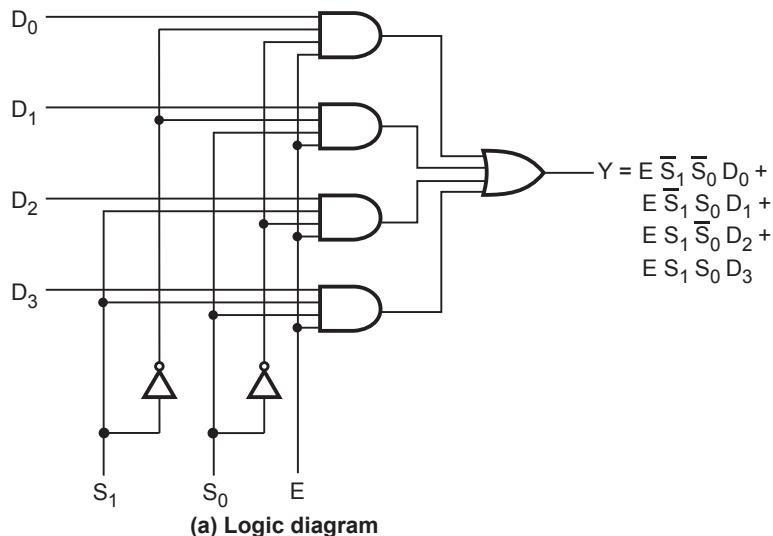
$$\therefore Y = E \bar{S} D_0 + E S D_1$$

Enable (E)	Select (S)	D_1	D_0	Output Y
1	0	X	0	0
1	0	X	1	1
1	1	0	X	0
1	1	1	X	1
0	X	X	X	0

Table 3.11.1 Truth table for 2 : 1 multiplexer

3.11.2 4 : 1 Multiplexer

Fig. 3.11.3 (a) shows 4-to-1 line multiplexer. Each of the four lines, D_0 to D_3 , is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.



E	S_1	S_0	Y
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3
0	X	X	0

(b) Function table

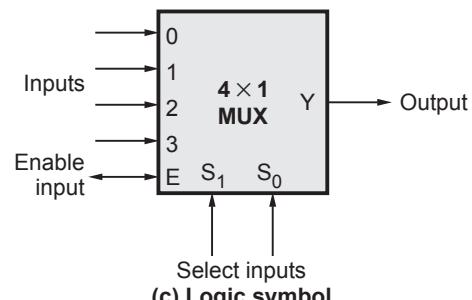


Fig. 3.11.3 4 to 1 line multiplexer

For example, when $S_1 S_0 = 01$, the AND gate associated with data input D_1 has two of its inputs equal to 1 and the third input connected to D_1 . The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of D_1 , thus we can say data bit D_1 is routed to the output when $S_1 S_0 = 01$.

3.11.3 8 : 1 Multiplexer

Fig. 3.11.4 shows 8 : 1 multiplexer.

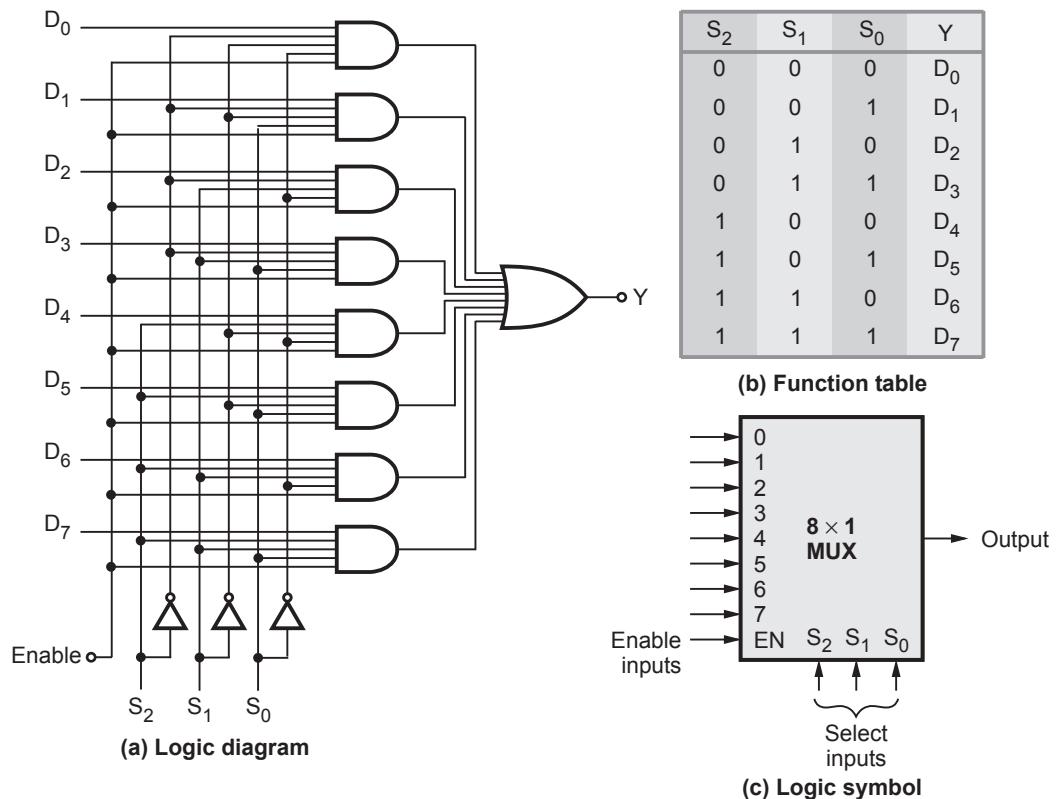


Fig. 3.11.4 8 : 1 Multiplexer

There are eight input lines one output line and three select lines. As shown in the function table, the selection of a particular input line is controlled by three selection lines.

3.11.4 Quadruple 2 to 1 Multiplexer

In some cases, two or more multiplexers are enclosed within one IC package, as shown in the Fig. 3.11.5. The Fig. 3.11.5 shows quadruple 2-to-1 line multiplexer, i.e. four multiplexers, each capable of selecting one of two input lines. Output Y_1 can be selected

to be equal to either A_1 or B_1 . Similarly output Y_2 may have the value of A_2 or B_2 , and so on. The selection line S selects one of two lines in all four multiplexers. The control input E enables the multiplexers in the 0 state and disables them in the 1 state. When $E = 1$, outputs have all 0's, regardless of the value of S .

Function Table

E	S	Output Y
1	X	All 0s
0	0	Select A
0	1	Select B

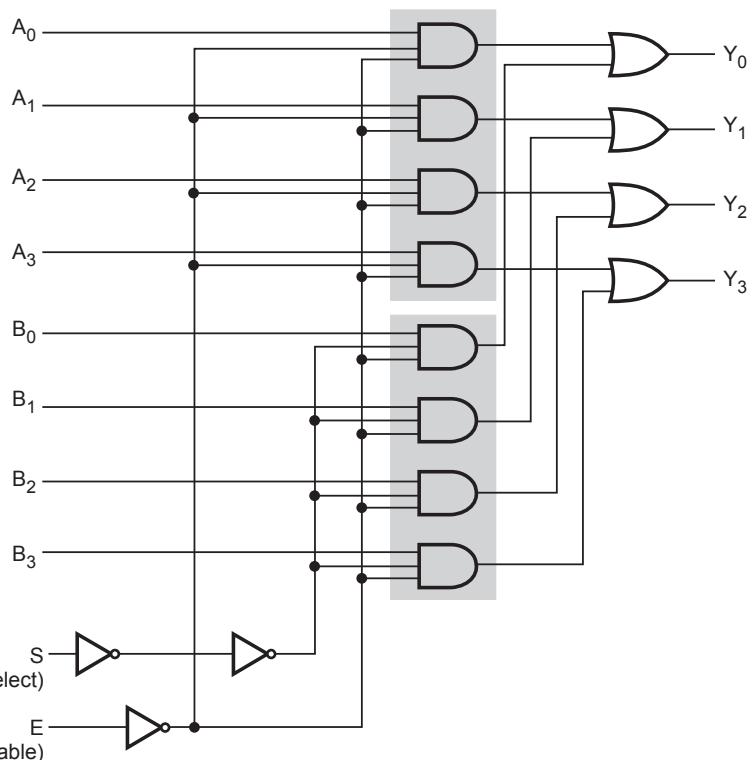


Fig. 3.11.5 Quadruple 2-to-1 line multiplexer

3.11.5 The 74151 Multiplexer

The 74XX151 is a 8 to 1 multiplexer. It has eight inputs. It provides two outputs, one is active high, the other is active low. The Fig. 3.11.6 shows the logic symbol for 74XX151. As shown in the logic symbol, there are three select inputs C, B and A which select one of the eight inputs. The 74XX151 is provided with active low enable input.

The Table 3.11.2 shows the truth table for 74XX151. In this truth table for each input combinations output is not specified in 1s and 0s. Because, we know that, multiplexer is a data switch, it does not generate any data of its own, but it simply passes external input data from the selected input to the output. Therefore, the two output column represent data by D_n and \bar{D}_n .

Input			Outputs		
Select		Enable			
C	B	A	\overline{EN}	Y	\overline{Y}
x	x	x	1	0	1
0	0	0	0	D_0	\overline{D}_0
0	0	1	0	D_1	\overline{D}_1
0	1	0	0	D_2	\overline{D}_2
0	1	1	0	D_3	\overline{D}_3
1	0	0	0	D_4	\overline{D}_4
1	0	1	0	D_5	\overline{D}_5
1	1	0	0	D_6	\overline{D}_6
1	1	1	0	D_7	\overline{D}_7

Table 3.11.2 Truth table for 74XX151, 8 to 1 multiplexer

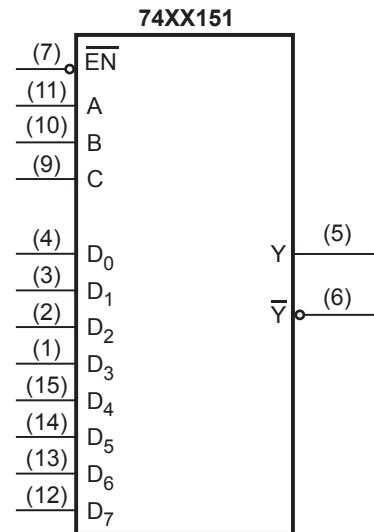


Fig. 3.11.6 Logic symbol for 74XX151, 8 to 1 multiplexer

3.11.6 The 74XX153 Dual 4 to 1 Multiplexer

The 74XX153 is a dual 4 to 1 multiplexer. Fig. 3.11.7 shows the logic symbol for 74XX153. It contains two identical and independent 4-to-1 multiplexers. Each multiplexer has separate enable inputs. The Table 3.11.3 shows the truth table for 74XX153.

Inputs				Outputs	
1EN	2EN	B	A	1Y	2Y
0	0	0	0	$1D_0$	$2D_0$
0	0	0	1	$1D_1$	$2D_1$
0	0	1	0	$1D_2$	$2D_2$
0	0	1	1	$1D_3$	$2D_3$
0	1	0	0	$1D_0$	0
0	1	0	1	$1D_1$	0
0	1	1	0	$1D_2$	0
0	1	1	1	$1D_3$	0
1	0	0	0	0	$2D_0$
1	0	0	1	0	$2D_1$
1	0	1	0	0	$2D_2$
1	0	1	1	0	$2D_3$
1	1	x	x	0	0

Table 3.11.3 Truth table for 74XX153, dual 4-to-1 multiplexer

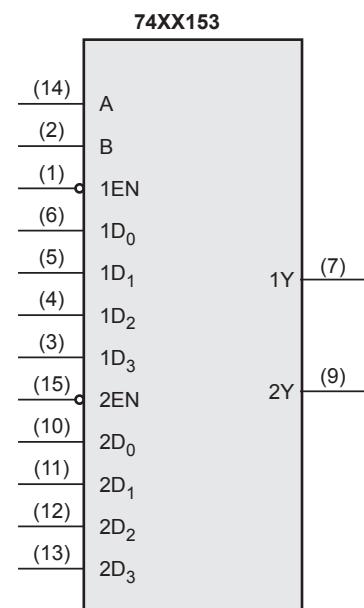


Fig. 3.11.7 Logic symbol for 74XX153

3.11.7 Expanding Multiplexers

It is possible to expand range of inputs for multiplexer beyond the available range by interconnecting several multiplexers in cascade. The circuit with two or more multiplexers connected to obtain the multiplexer with more number of inputs is known as **multiplexer tree**.

Illustrative Examples

Example 3.11.1 Design $16 : 1$ multiplexer using $8 : 1$ multiplexer.

Solution :

Step 1 : Connect the select lines (S_2, S_1 and S_0) of two multiplexers in parallel.

Step 2 : Connect most significant select line (S_3) such that when $S_3 = 0$ MUX 1 is enabled and when $S_3 = 1$, MUX 2 is enabled.

Step 3 : Logically OR the outputs of two multiplexers to obtain the final output Y .

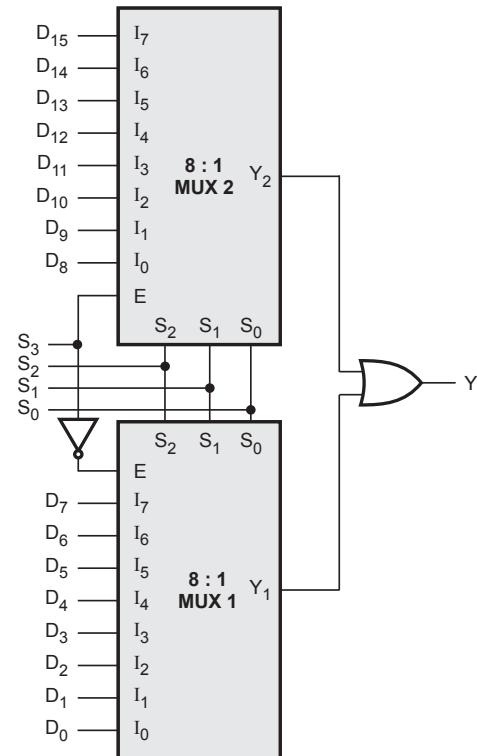


Fig. 3.11.8 $16 : 1$ multiplexer using two $8 : 1$ multiplexers

Example 3.11.2 Design $16 : 1$ multiplexer using $4 : 1$ multiplexers.

Explain the truth table of your design.

SPPU : May-11, Dec.-16, Marks 8

Solution : Since there are 16-inputs for the multiplexers we require four $4 : 1$ multiplexers to satisfy input needs. The four outputs of $4 : 1$ multiplexers are again multiplexed by $4 : 1$ multiplexer to generate final output.

Step 1 : Connect the select lines (S_1 and S_0) of four multiplexers in parallel.

Step 2 : Connect the most significant select lines (S_3 and S_2) to the MUX 5.

Step 3 : Connect the outputs Y_0, Y_1, Y_2 and Y_4 of four multiplexers as data inputs for the MUX 5, as shown in the Fig. 3.11.9.

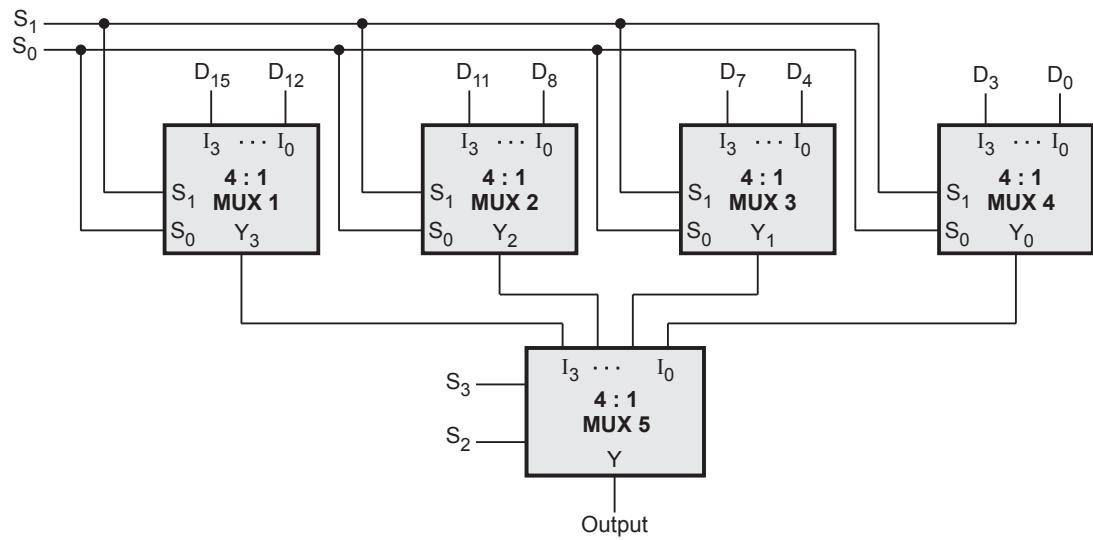


Fig. 3.11.9

Truth Table

S_3	S_2	S_1	S_0	Y_3	Y_2	Y_1	Y_0	Y
0	0	0	0	D_{12}	D_8	D_4	D_0	D_0
0	0	0	1	D_{13}	D_9	D_5	D_1	D_1
0	0	1	0	D_{14}	D_{10}	D_6	D_2	D_2
0	0	1	1	D_{15}	D_{11}	D_7	D_3	D_3
0	1	0	0	D_{12}	D_8	D_4	D_0	D_4
0	1	0	1	D_{13}	D_9	D_5	D_1	D_5
0	1	1	0	D_{14}	D_{10}	D_6	D_2	D_6
0	1	1	1	D_{15}	D_{11}	D_7	D_3	D_7
1	0	0	0	D_{12}	D_8	D_4	D_0	D_8
1	0	0	1	D_{13}	D_9	D_5	D_1	D_9
1	0	1	0	D_{14}	D_{10}	D_6	D_2	D_{10}
1	0	1	1	D_{15}	D_{11}	D_7	D_3	D_{11}
1	1	0	0	D_{12}	D_8	D_4	D_0	D_{12}
1	1	0	1	D_{13}	D_9	D_5	D_1	D_{13}
1	1	1	0	D_{14}	D_{10}	D_6	D_2	D_{14}
1	1	1	1	D_{15}	D_{11}	D_7	D_3	D_{15}

Truth table shows the output of MUX 1 to MUX 4, i.e. Y_3 to Y_0 . According to select line $S_1 S_0$, the input data line is connected at the output. The outputs Y_3 to Y_0 acts as data inputs for MUX 5. Truth table also shows the output Y for Y_3 to Y_0 as data lines for MUX 5, according the select inputs S_3 and S_2 .

Examples for Practice**Example 3.11.3** Draw the block diagram of a 4 : 1 multiplexer using 2 : 1 MUX.**Example 3.11.4** Design 32 : 1 MUX using 8 : 1 MUX.**Example 3.11.5** Design 14 : 1 mux using 4 : 1 mux (with enable inputs). Explain the truth table of your circuit in short. **SPPU : May-10, Marks 8****Example 3.11.6** Design 28 : 1 mux using 8 : 1 mux (with enable inputs). Explain truth table of your design in short. [Hint : You can use separate mux for enable of respective IC's] **SPPU : Dec.-10, Marks 8****3.11.8 Implementation of Combinational Logic using MUX**

A multiplexer consists of a set of AND gates whose outputs are connected to single OR gate. Because of this construction any Boolean function in a SOP form can be easily realized using multiplexer. Each AND gate in the multiplexer represents a minterm. In 8 to 1 multiplexer, there are 3 select inputs and 23 minterms. By connecting the function variables directly to the select inputs, a multiplexer can be made to select the AND gate that corresponds to the minterm in the function. If a minterm exists in a function, we have to connect the AND gate data input to logic 1; otherwise we have to connect it to logic 0. This is illustrated in the following example.

Illustrative Examples**Example 3.11.7** Implement the given function using multiplexer.

$$F(x, y, z) = \Sigma(0, 2, 6, 7).$$

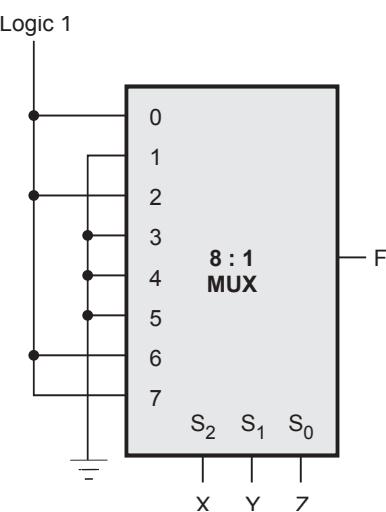
Solution :

Step 1 : Select the multiplexer. Here, Boolean expression has 3 variables, thus we require $2^3 = 8 : 1$ multiplexer.

Step 2 : Connect inputs corresponds to the present minterms to logic 1.

Step 3 : Connect remaining inputs to logic 0.

Step 4 : Connect input variables to select lines of MUX.

**Fig. 3.11.10**

Example 3.11.8 Implement the Boolean function represented by the given truth table using multiplexer.

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

Solution :

Step 1 : Select the multiplexer. Here, there are three input variables, thus we require $2^3 = 8 : 1$ multiplexer.

Step 2 : Find the minterm expression.

Minterm expression for given truth table is $\sum m (1, 2, 5, 7)$.

Step 3 : Connect inputs corresponds to the present minterms to logic 1.

Step 4 : Connect remaining inputs to logic 0.

Step 5 : Connect input variables to select lines of MUX.

In the above example, we have seen the method for implementing Boolean function of 3 variables with $2^3(8)$ -to-1 multiplexer. Similarly, we can implement any Boolean function of n variables with 2^n -to-1 multiplexer. However, it is possible to do better than this. If we have Boolean function of $n + 1$ variables, we take n of these variables and connect them to the selection lines of a multiplexer. The remaining single variable of the function is used for the inputs of the multiplexer. In this way we can implement any Boolean function of n variables with 2^{n-1} -to-1 multiplexer. Let us see some example.

Example 3.11.9 Implement the following Boolean function using 4 : 1 multiplexer.

$$F(A, B, C) = \sum m (1, 3, 5, 6)$$

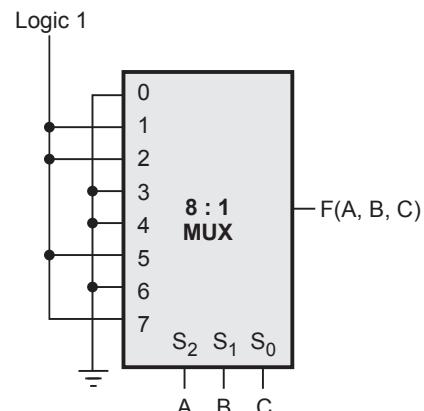


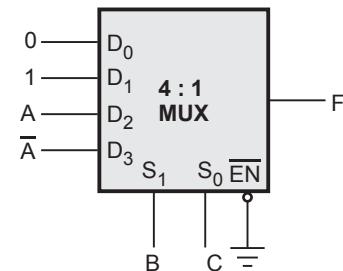
Fig. 3.11.11

Solution : Step 1 : Connect least significant variables as a select inputs of multiplexer. Here, connect C to S₀ and B to S₁.

Step 2 : Derive inputs for multiplexer using implementation table.

	D ₀	D ₁	D ₂	D ₃
Ā	0	①	2	③
A	4	⑤	⑥	7
Most significant variable	0	1	A	Ā

(a) Implementation table
Fig. 3.11.12



(b) Multiplexer implementation

As shown in the Fig. 3.11.12 (a) the implementation table is nothing but the list of the inputs of the multiplexer and under them list of all the minterms in two rows. The first row lists all those minterms where A is complemented, and the second row lists all the minterms with A uncomplemented. The minterms given in the function are circled and then each column is inspected separately as follows :

- If the two minterms in a column are not circled, 0 is applied to the corresponding multiplexer input (see column 0).
 - If the two minterms in a column are circled, 1 is applied to the corresponding multiplexer input (see column 1).
 - If the minterm in the second row is circled and minterm in the first row is not circled, A is applied to the corresponding multiplexer input (see column 2).
 - If the minterm in the first row is circled and minterm in the second row is not circled, \bar{A} is applied to the corresponding multiplexer input (see column 3).

Example 3.11.10 Implement the following Boolean function using 8 : 1 multiplexer

$$F(A, B, C, D) \equiv \overline{A} \cdot B \cdot \overline{D} + A \cdot C \cdot D + \overline{B} \cdot C \cdot D + \overline{A} \cdot \overline{C} \cdot D$$

Solution : Step 1 : Express Boolean function in the minterm form.

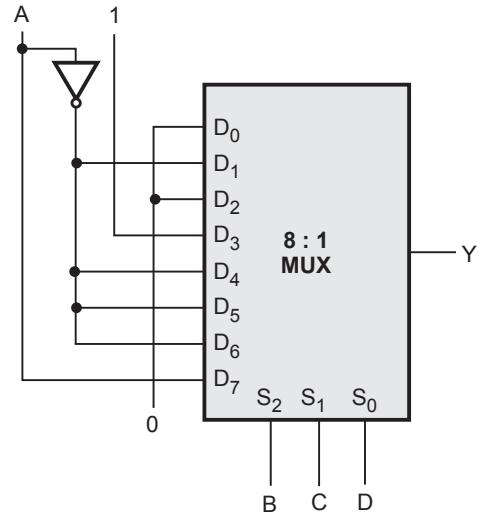
The given Boolean expression is not in standard SOP form. Let us first convert this in standard SOP form

$$\begin{aligned}
 F(A, B, C, D) &= \overline{A} B \overline{D} (C + \overline{C}) + A C D (B + \overline{B}) + \overline{B} C D (A + \overline{A}) + \overline{A} \overline{C} D (B + \overline{B}) \\
 &= \overline{A} B C \overline{D} + \overline{A} B \overline{C} \overline{D} + A B C D + A \overline{B} C D \\
 &\quad + A \overline{B} C D + \overline{A} \overline{B} C D + \overline{A} B \overline{C} D + \overline{A} \overline{B} \overline{C} D \\
 &= \overline{A} B C \overline{D} + \overline{A} B \overline{C} \overline{D} + A B C D + A \overline{B} C D + \overline{A} \overline{B} C D + \overline{A} B \overline{C} D + \overline{A} \overline{B} \overline{C} D \\
 &= \sum m(6, 4, 15, 11, 3, 5, 1) \\
 &= \sum m(1, 3, 4, 5, 6, 11, 15)
 \end{aligned}$$

Step 2 : Implement it using implementation table.

From the Boolean function in the minterm form can be implemented using 8 : 1 multiplexer as follows :

	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
A	0	①	2	③	④	⑤	⑥	7
A	8	9	10	⑪	12	13	14	⑯
	0	\bar{A}	0	1	\bar{A}	\bar{A}	\bar{A}	A



(a) Implementation table

(b) Multiplexer implementation

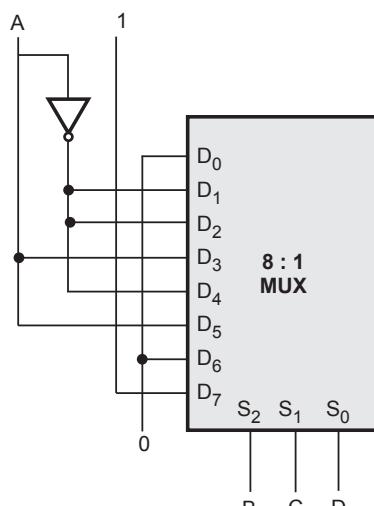
Fig. 3.11.13

Example 3.11.11 Implement the following Boolean function with 8 : 1 multiplexer

$$F(A, B, C, D) = \pi M(0, 3, 5, 8, 9, 10, 12, 14)$$

Solution : Here, instead of minterms, maxterms are specified. Thus, we have to circle maxterms which are not included in the Boolean function. Fig. 3.11.14 shows the implementation of Boolean function with 8 : 1 multiplexer.

	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
A	0	①	②	3	④	5	6	⑦
A	8	9	10	⑪	12	⑬	14	⑯
	0	\bar{A}	\bar{A}	A	\bar{A}	A	0	1



(a) Implementation table

(b) Multiplexer implementation

Fig. 3.11.14

Example 3.11.12 Implement the following Boolean function with 8 : 1 multiplexer.

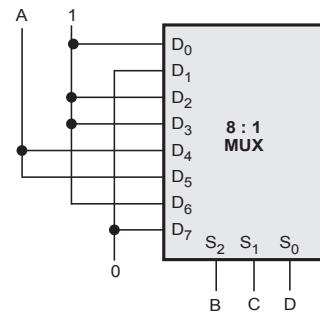
$$F(A, B, C, D) = \sum m(0, 2, 6, 10, 11, 12, 13) + d(3, 8, 14)$$

Solution : In the given Boolean function three don't care conditions are also specified. We know that don't care conditions can be treated as either 0s or 1s. Fig. 3.11.15 shows the implementation of given Boolean function using 8 : 1 multiplexer.

	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
	1	0	1	1	A	A	1	0

Here, don't cares are treated as 1s

(a) Implementation table



(b) Implementation

Fig. 3.11.15

In this example, by taking don't care conditions 8 and 14 as 1s we have eliminated \bar{A} term and hence the inverter.

Example 3.11.13 Implement the expression using a 8 : 1 multiplexer

$$f(a, b, c, d) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

SPPU : May-17, Marks 4

Solution :

Implementation 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
	1	A	\bar{A}	\bar{A}	A	0	1	0

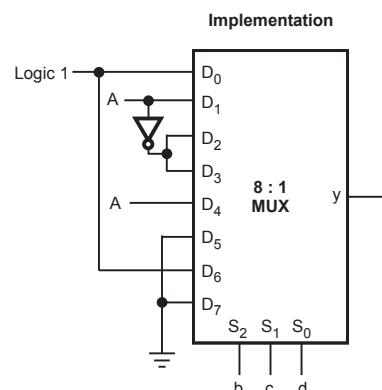


Fig. 3.11.16

Example 3.11.14 Implement full adder using 8 : 1 multiplexer and draw the diagram.

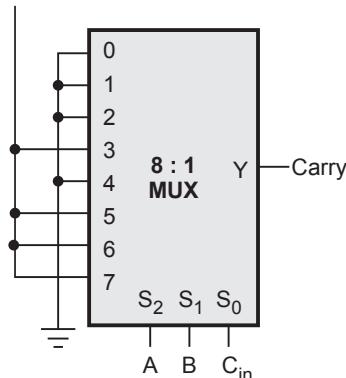
SPPU : Dec.-17, Marks 6

Solution :

Inputs			Outputs	
A	B	C _{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 3.11.4

Logic 1



Logic 1

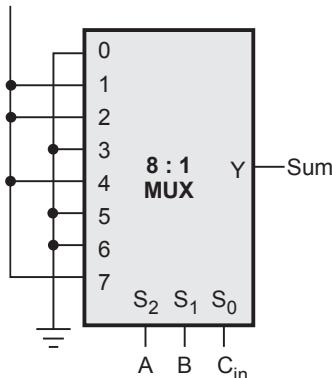


Fig. 3.11.17

Example 3.11.15 Design full subtractor using multiplexer IC 74151.

SPPU : Dec.-18, Marks 4

Solution :

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

Table 3.11.5 Truth table for full-subtractor

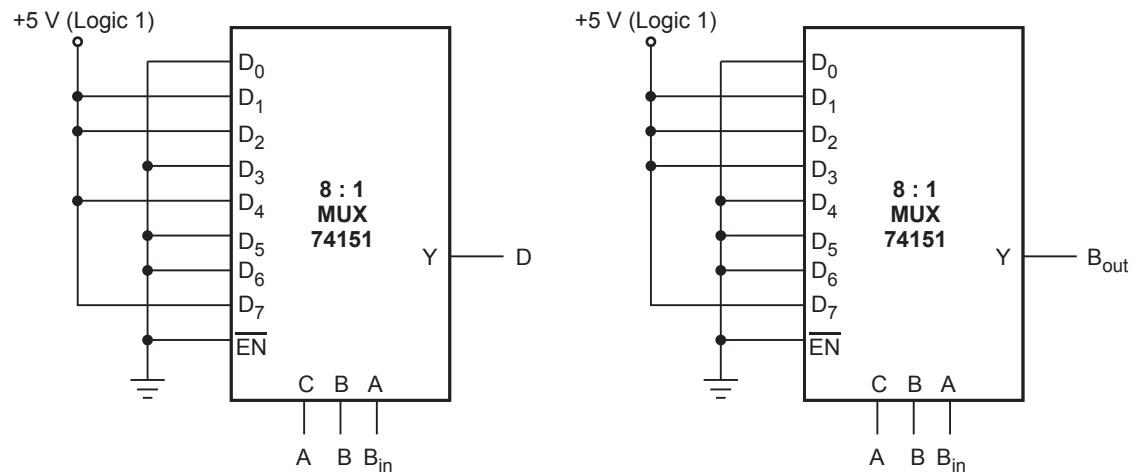


Fig. 3.11.18

Example 3.11.16 Implement the following function using 8 : 1 MUX and logic gates.

$$F(A, B, C, D) = \sum(0, 2, 5, 8, 10, 15)$$

SPPU : Dec.-19, Marks 6

Solution :

Implementation

Implementation 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)
	1	0	1	0	0	\bar{A}	0	A

Fig. 3.11.19 (a)

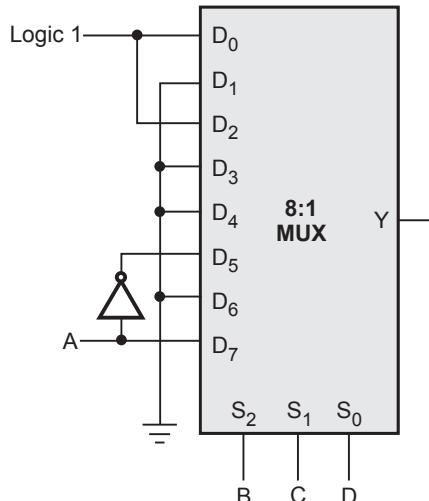


Fig. 3.11.19 (b)

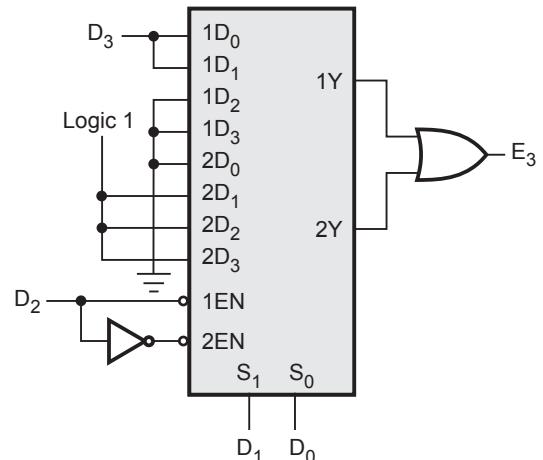
Example 3.11.17 Design and implement BCD to Excess-3 code converter using dual 4:1 multiplexers and some logic gates.

SPPU : May-13, Marks 8

Solution : Refer Table 3.2.2 for truth table of BCD to Excess-3 code conversion.

	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
D ₃	0	1	2	3	4	5	6	7
D ₃	8	9	10	11	12	13	14	15
	D ₃	D ₃	0	0	0	1	1	1

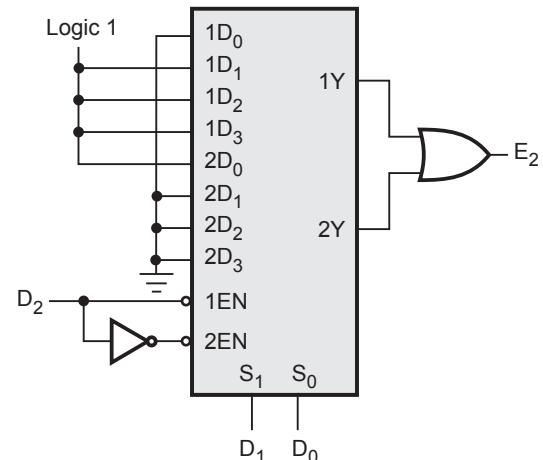
Note : Don't care minterms 13, 14 and 15 are considered as logic 1.

(a) Implementation table for E₃

(b) Implementation

	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
D ₃	0	1	1	1	1	5	6	7
D ₃	8	9	10	11	12	13	14	15
	0	1	1	1	1	0	0	0

Note : Don't care minterms 10, 11 and 12 are considered as logic 1.

(a) Implementation table for E₂

(b) Implementation

Fig. 3.11.21

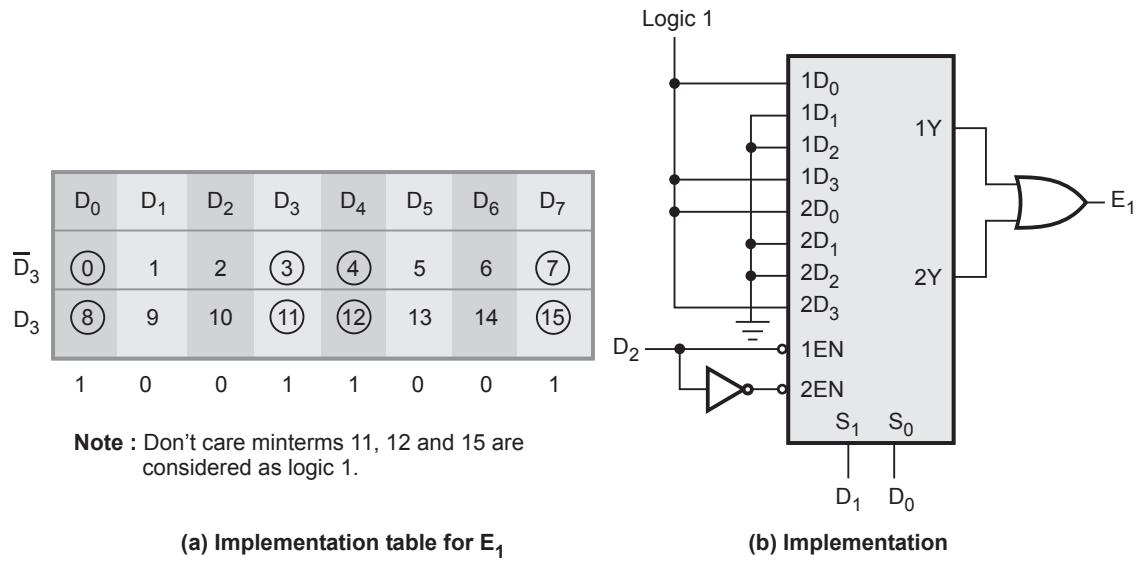


Fig. 3.11.22



Examples for Practice

Example 3.11.18 Implement the following expression using 8 : 1 multiplexer :

$$f(A, B, C, D) = \Sigma m(2, 4, 6, 7, 9, 10, 11, 12, 15).$$

SPPU : Dec.-12, Marks 8

Example 3.11.19 Implement Boolean function $f = AB + \overline{C}D + A\overline{B}C$ using 8 : 1 multiplexer.

3.11.9 Applications of Multiplexer

1. They are used as a data selector to select one out of many data inputs.
2. They can be used to implement combinational logic circuit.
3. They are used in time multiplexing systems.
4. They are used in frequency multiplexing systems.
5. They are used in A/D and D/A converter.
6. They are used in data acquisition systems.

3.11.10 Multiplexer ICs

IC number	Function
74150	16 : 1 multiplexer
74151	8 : 1 multiplexer
74153	Dual 4 : 1 multiplexer
74157	Quad 2-input multiplexer

Table 3.11.4 Multiplexer ICs

Review Questions

1. What is multiplexer ?
2. Explain the working of 8:1 multiplexer.
3. Obtain an 8 : 1 multiplexer with a dual 4-line to 1-line multiplexers having separate enable inputs but common selection lines. **SPPU : May-05, Marks 4**
4. Design an 8 : 1 multiplexer using two 4 : 1 multiplexers. Explain with the help of the truth table. Implement the function $f(A, B, C) = \sum m(1, 3, 7)$ using the same. **SPPU : May-12, Marks 8**

3.12 Demultiplexers (DEMUX)

SPPU : Dec.-11

A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of specific output line is controlled by the values of n selection lines.

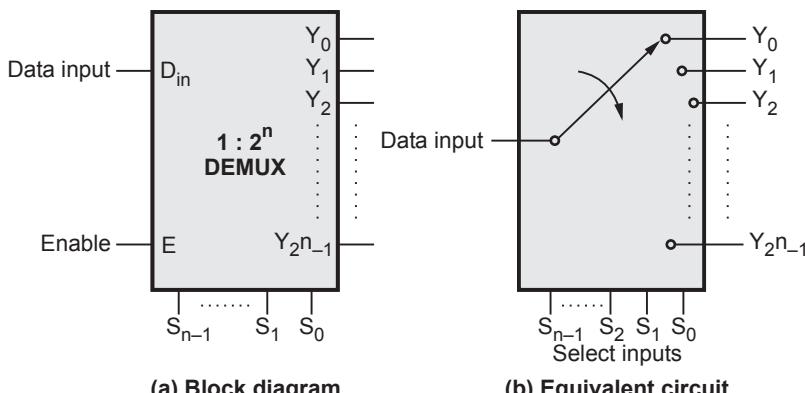


Fig. 3.12.1

The Fig. 3.12.1 shows the block diagram of a demultiplexer. It has one input data line, 2^n output lines, n select lines and one enable input.

Differentiate between Multiplexer and Demultiplexer

Parameter	Multiplexer	Demultiplexer
Definition	Multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line.	Demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines
Number of data inputs	2^n	1
Number of data outputs	1	2^n
Relationship of input and output	Many to one	One to many
Applications	<ul style="list-style-type: none"> Used as a data selector In time division multiplexing at the transmitting end 	<ul style="list-style-type: none"> Used as a data distributor In time division multiplexing at the receiving end

3.12.1 Types of Demultiplexers

3.12.1.1 1 : 4 Demultiplexer

Fig. 3.12.2 shows 1 : 4 demultiplexer. The single input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines depending on the select inputs. Enable input should be high to enable demultiplexer.

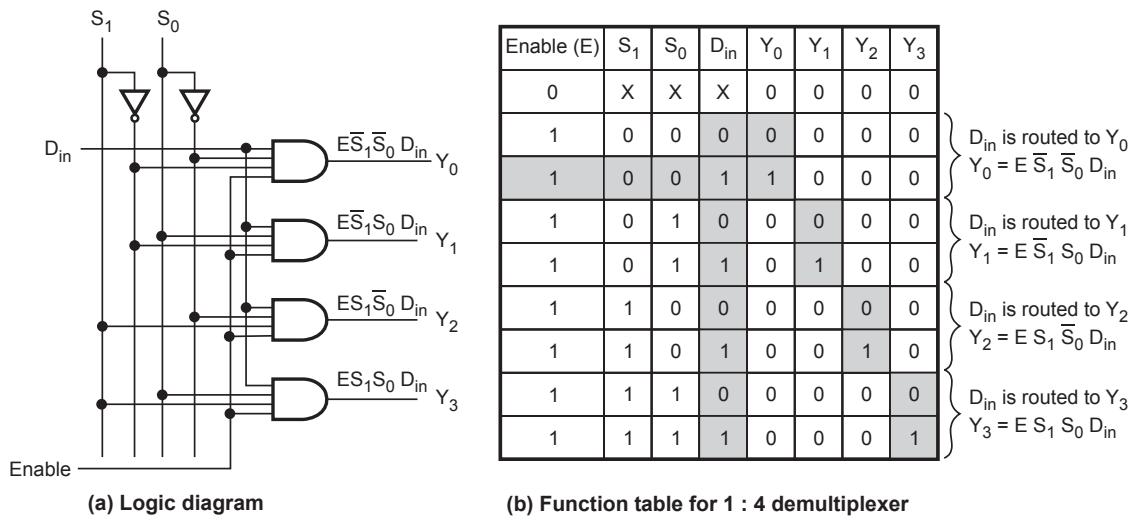
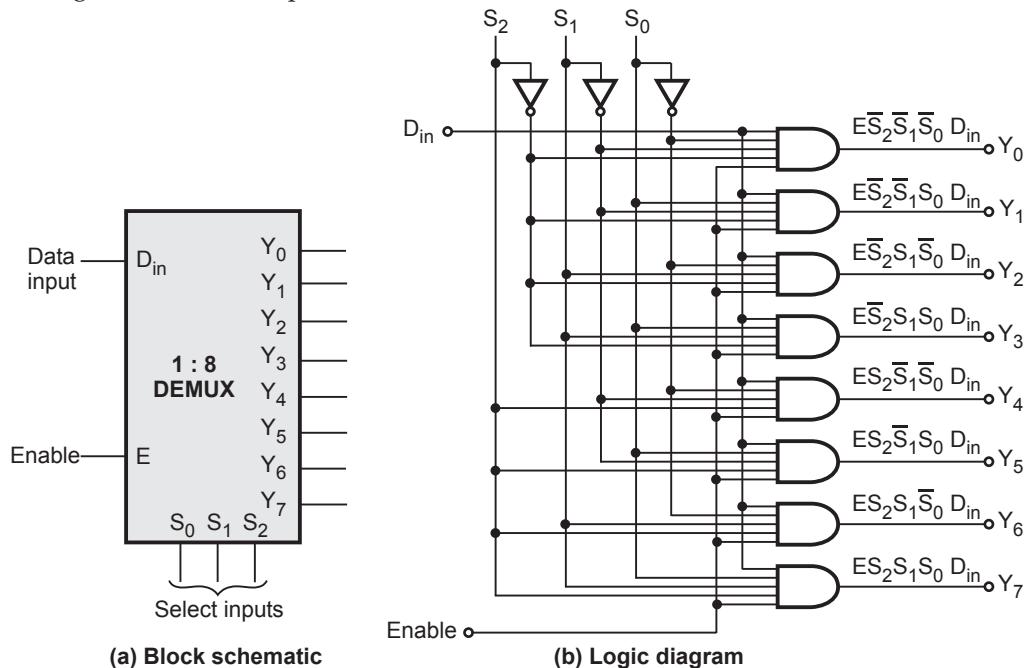


Fig. 3.12.2

3.12.1.2 1 : 8 Demultiplexer

The Fig. 3.12.3 shows 1 : 8 demultiplexer. The single input data D_{in} has a path to all eight outputs, but the input information is directed to only one of the output lines depending on the select inputs.



Enable	Select inputs			Outputs								
	E	S ₂	S ₁	S ₀	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
0	X	X	X	0	0	0	0	0	0	0	0	0
1	0	0	0	D _{in}	0	0	0	0	0	0	0	0
1	0	0	1	0	D _{in}	0	0	0	0	0	0	0
1	0	1	0	0	0	D _{in}	0	0	0	0	0	0
1	0	1	1	0	0	0	D _{in}	0	0	0	0	0
1	1	0	0	0	0	0	0	D _{in}	0	0	0	0
1	1	0	1	0	0	0	0	0	D _{in}	0	0	0
1	1	1	0	0	0	0	0	0	0	D _{in}	0	0
1	1	1	1	0	0	0	0	0	0	0	D _{in}	0

(c) Function table

Fig. 3.12.3 1 : 8 demultiplexer

3.12.2 Expanding Demultiplexers

To provide larger output needs we can cascade two or more demultiplexer to get demultiplexer with more number of output lines. Such a connection is known as **demultiplexer tree**.

Example 3.12.1 Design 1 : 8 demultiplexer using two 1 : 4 demultiplexers.

Solution :

Step 1 : Connect D_{in} signal to D_{in} input of both the demultiplexers.

Step 2 : Connect select lines B and C to select lines S_1 and S_0 of the both demultiplexers, respectively.

Step 3 : Connect most significant select line (A) such that when $A = 0$ DEMUX 1 is enabled and when $A = 1$ DEMUX 2 is enabled.

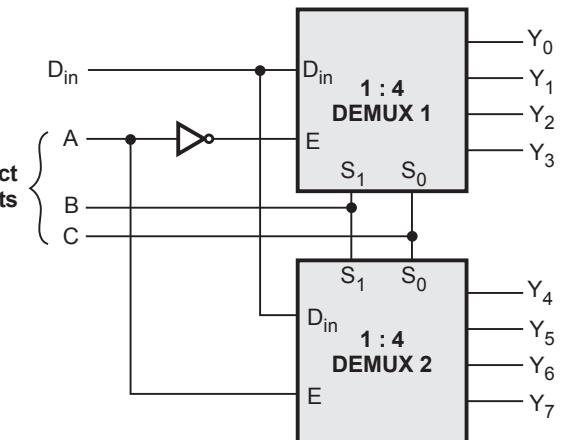


Fig. 3.12.4 Cascading of demultiplexers

Example 3.12.2 Implement 1 : 16 demultiplexer using 1 : 4 demultiplexers.

Solution : The 1 : 16 demultiplexer has 16 outputs. To select one of the 16 output, the circuit needs 4 ($\because 2^4 = 16$) select lines. Each 1 : 4 demultiplexer requires 2 select lines.

Step 1 : Connect two least significant select lines (S_1, S_0) to select lines of four 4 : 1 demultiplexer.

Step 2 : Connect one more 4 : 1 demultiplexer such that its four outputs are routed to the data inputs of the four demultiplexers. Connect higher select lines (S_3, S_2) to the select lines of this demultiplexer. (See Fig. 3.12.5 on next page).

Examples for Practice

Example 3.12.3 Draw 1 : 64 demultiplexer tree using 1 : 16 demultiplexer.

Example 3.12.4 Draw 1 : 64 demultiplexer tree using 1 : 8 demultiplexer.

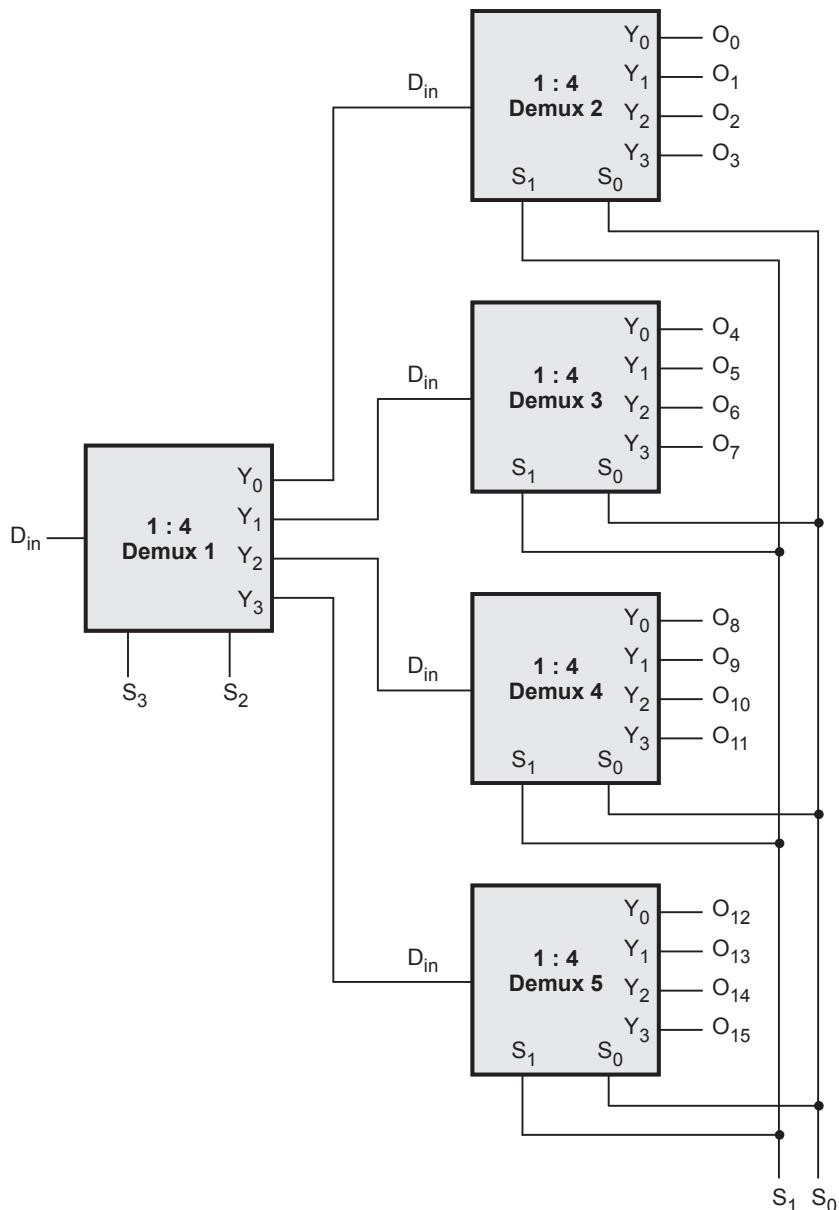


Fig. 3.12.5 1 : 16 Demux using 1 : 4 Demux

3.12.3 Implementation of Combinational Logic using Demultiplexer

Example 3.12.5 Implement full subtractor using demultiplexer.

Solution : **Step 1 :** Write the truth table of full subtractor.

Step 2 : Represent output of full-subtractors in minterm form.

For full subtractor difference D function can be written as $D = f(A, B, C) = \sum m(1, 2, 4, 7)$ and B_{out} function can be written as,

$$B_{out} = F(A, B, C) = \sum m(1, 2, 3, 7)$$

Step 3 : Logically OR the outputs corresponding to minterms.

With D_{in} input 1, demultiplexer gives minterms at the output so by logically ORing required minterms we can implement Boolean functions for full subtractor. Fig. 3.12.6 shows the implementation of full subtractor using demultiplexer.

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

Table 3.12.1 Truth table of full subtractor

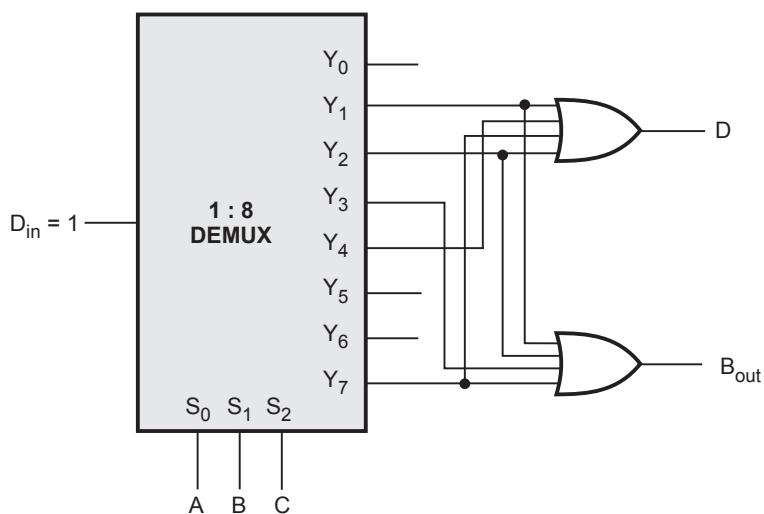


Fig. 3.12.6 Full subtractor using 1 : 8 demultiplexer

Example 3.12.6 Implement the following functions using demultiplexer :

$$f_1(A, B, C) = \sum m(0, 3, 7)$$

$$f_2(A, B, C) = \sum m(1, 2, 5).$$

Solution : $f_1(A, B, C) = \sum m(0, 3, 7)$

$$f_2(A, B, C) = \sum m(1, 2, 5)$$

Implementation using 1 : 8 demultiplexer. (See Fig. 3.12.7 on next page)

Examples for Practice

Example 3.12.7 Implement full adder using demultiplexer.

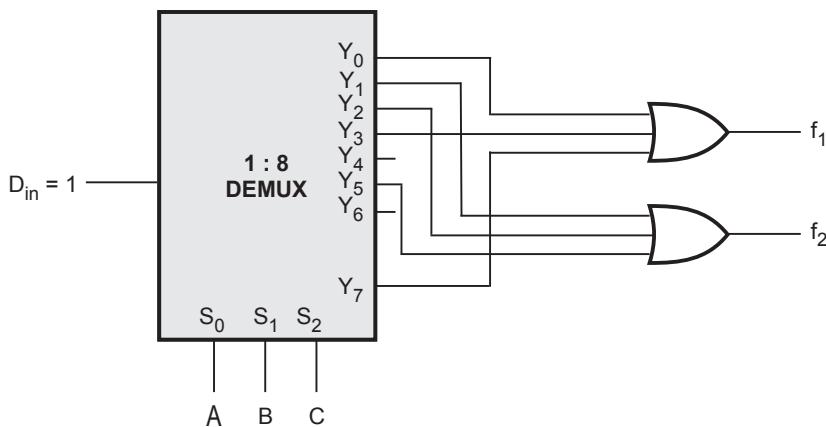


Fig. 3.12.7

Example 3.12.8 Implement the following functions using demultiplexer

$$f_1(A, B, C) = \sum m(1, 5, 7), \quad f_2(A, B, C) = \sum m(3, 6, 7)$$

Example 3.12.9 Implement two bit comparator using 1 : 16 demultiplexer (active low output). Draw the truth table of two bit comparator and explain the design in steps.

SPPU : Dec.-11, Marks 8

3.12.4 Applications of Demultiplexer

1. It can be used as a decoder.
2. It can be used as a data distributor.
3. It is used in time division multiplexing at the receiving end as a data separator.
4. It can be used to implement Boolean expressions.

3.12.5 Demultiplexer ICs

IC Number	Function
74154	1 : 16 Demultiplexer
74155	Dual 1 : 4 Demultiplexer

Table 3.12.2 Demultiplexer ICs

Review Questions

1. What is demultiplexer ?
2. Explain the working of 1:8 demux.

3.13 Decoder

SPPU : May-10,11, Dec.-10,13

A decoder is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different.

The Fig. 3.13.1 shows the general structure of the decoder circuit. As shown in the Fig. 3.13.1, the encoded information is presented as n inputs producing 2^n possible outputs. The 2^n output values are from 0 through $2^n - 1$.

Usually, a decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

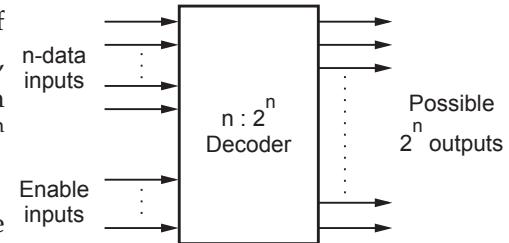


Fig. 3.13.1 General structure of decoder

3.13.1 Binary Decoder

A decoder which has an n -bit binary input code and a one activated output out of 2^n output code is called **binary decoder**. A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n -bit input value.

Fig. 3.13.2 shows 2 to 4 decoder. Here, 2 inputs are decoded into four outputs, each output representing one of the minterms of the 2 input variables. The two inverters provide the complement of the inputs, and each one of four AND gates generates one of the minterms.

Inputs			Outputs			
EN	A	B	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Table 3.13.1 Truth table for a 2 to 4 decoder

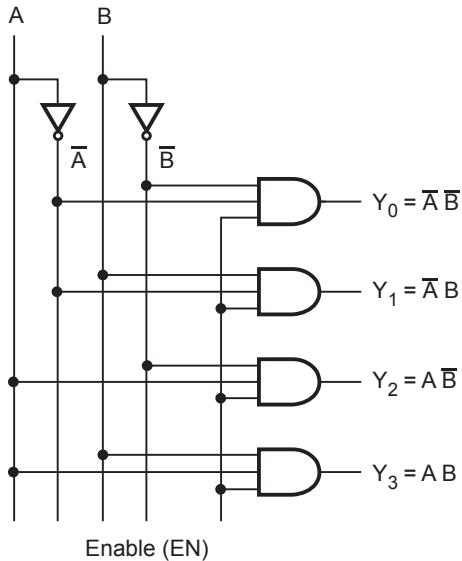


Fig. 3.13.2 2 to 4 line decoder

The Table 3.13.1 shows the truth table for a 2 to 4 decoder. As shown in the truth table, if enable input is 1 ($EN = 1$), one, and only one, of the outputs Y_0 to Y_3 , is active for a given input. The output Y_0 is active, i.e. $Y_0 = 1$ when inputs $A = B = 0$, the output Y_1 is active when inputs $A = 0$ and $B = 1$. If enable input is 0, i.e. $EN = 0$, then all the outputs are 0.

Example 3.13.1 Draw the circuit for 3 to 8 decoder and explain.

Solution : Fig. 3.13.3 shows 3 to 8 line decoder. Here, 3 inputs are decoded into eight outputs, each output represent one of the minterms of the 3 input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. Enable input is provided to activate decoded output based on data inputs A, B, and C. The table shows the truth table for 3 to 8 decoder.

EN	Inputs			Outputs							
	A	B	C	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	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table 3.13.2 Truth table for a 3 to 8 decoder

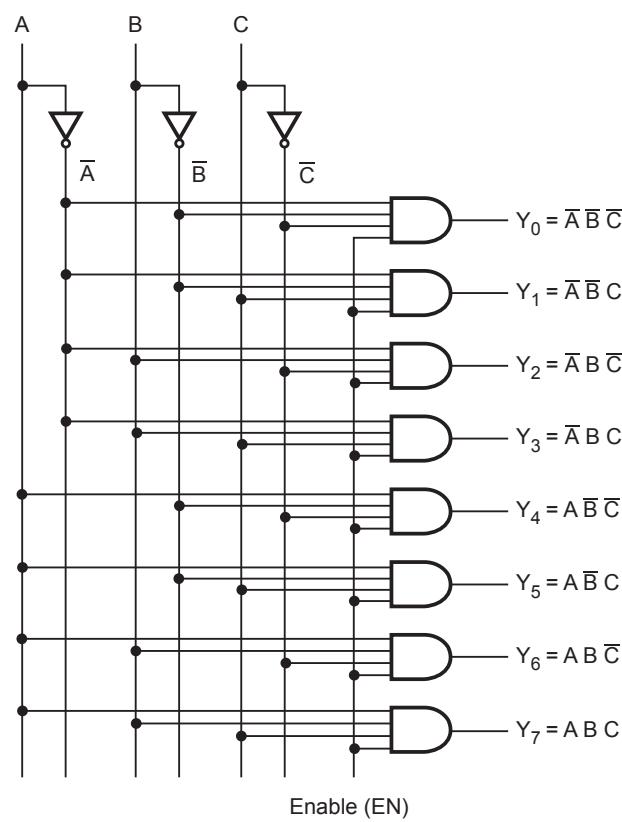


Fig. 3.13.3 3 : 8 line decoder

3.13.2 The 74X138 3-to-8 Decoder

The 74X138 is a commercially available 3-to-8 decoder. It accepts three binary inputs (A, B, C) and when enabled, provides eight individual active low outputs (Y_0 - Y_7). The

device has three enable inputs : two active low ($\overline{G}_{2A}, \overline{G}_{2B}$) and one active high (G_1). Fig. 3.13.4 and Table 3.13.3 show logic symbol and function table respectively.

Inputs						Outputs							
G_{2B}	G_{2A}	G_1	C	B	A	\bar{Y}_7	\bar{Y}_6	\bar{Y}_5	\bar{Y}_4	\bar{Y}_3	\bar{Y}_2	\bar{Y}_1	\bar{Y}_0
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	0	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	0	1	1	1	0	1	1	1	1
0	0	1	1	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1

Table 3.13.3 Function table

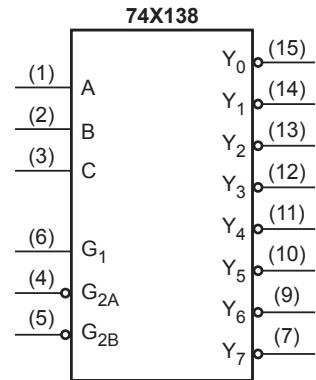


Fig. 3.13.4 Logic symbol

3.13.3 Expanding Cascading Decoders

Binary decoder circuits can be connected together to form a larger decoder circuit. Fig. 3.13.5 shows the 4×16 decoder using two 3×8 decoders.

Here, one input line (D) is used to enable/disable the decoders.

When D = 0, the top decoder is enabled and the other is disabled. Thus the bottom decoder outputs are all 1s, and the top eight outputs generate minterms 0 0 0 0 to 0 1 1 1. When D=1, the enable conditions are reversed and thus bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 1s.

Example 3.13.2 Design 5-to-32 decoder using one 2-to-4 and four 3-to-8 decoder ICs.

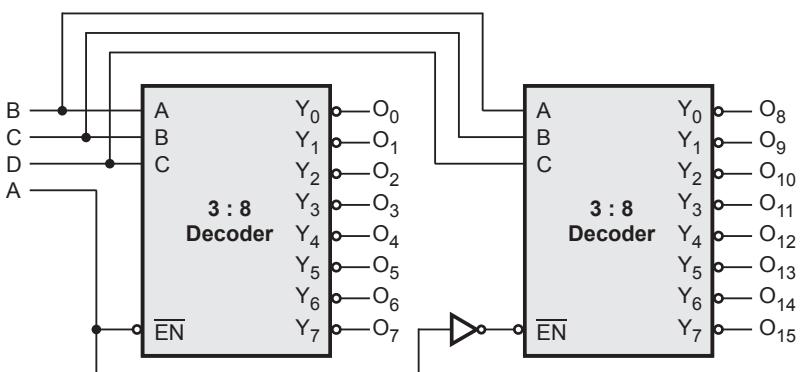


Fig. 3.13.5

Solution : The Fig. 3.13.6 shows the construction of 5-to-32 decoder using four 74LS138s and half 74LS139. The half section of 74LS139 IC is used as a 2-to-4 decoder to decode

the two higher order inputs, D and E. The four outputs of this decoder are used to enable one of the four 3 to 8 decoders. The three lower order inputs A, B and C are connected in parallel to four 3 to 8 decoders. This means that the same output pin of each of the four 3-to-8 decoders is selected but only one is enabled. The remaining enable signals of four 3-to-8 decoder ICs are connected in parallel to construct enable signals for 5-to-32 decoder.

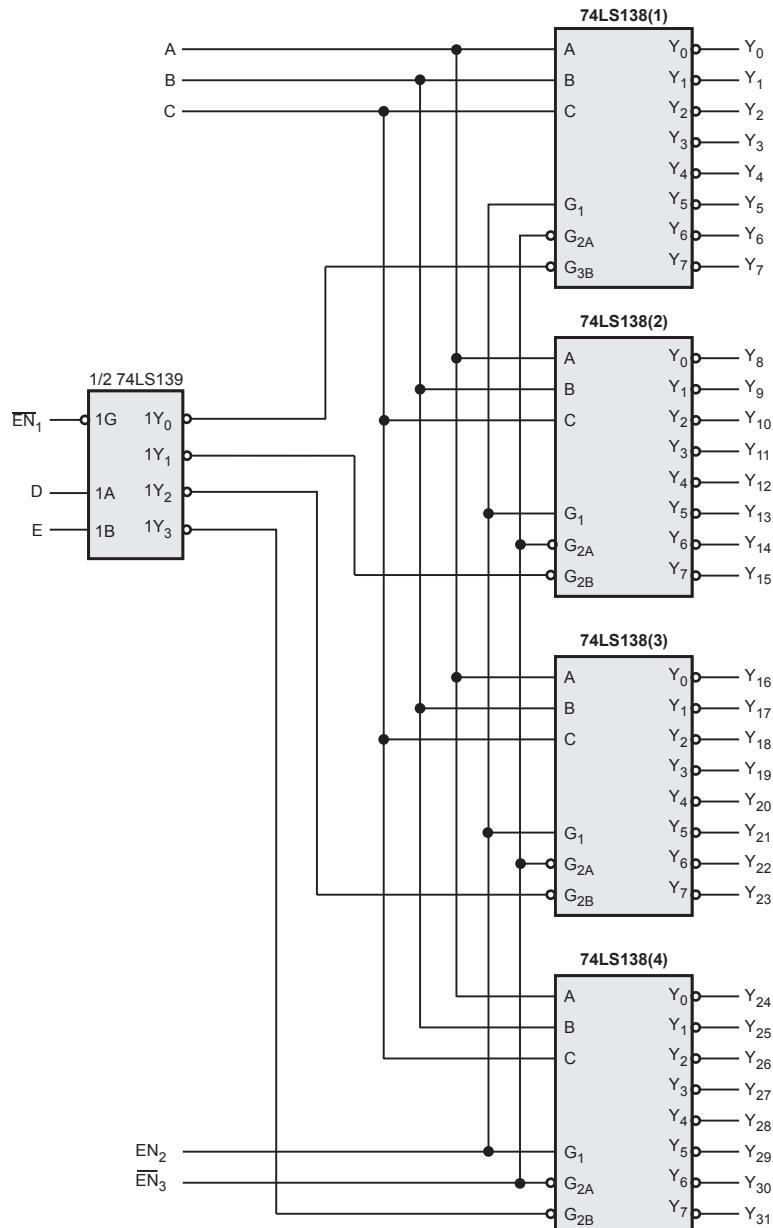


Fig. 3.13.6 5-to-32 decoder using 74LS138 and 74LS139

3.13.4 Realization of Boolean Function using Decoder

The combination of decoder and external logic gates can be used to implement single or multiple output functions. We know that decoder can have one of the two output states; either active low or active high. Let us see the significance of these output states in the implementation of binary function.

When decoder output is active high, it generates minterms (product terms) for input variables; i.e. it makes selected output logic 1. In such case to implement SOP function we have to take sum of selected product terms generated by decoder.

Illustrative Examples

Example 3.13.3 Implement Boolean function $F = \sum m(1, 2, 3, 7)$ using 3 : 8 decoder.

Solution : Step 1 : Connect function variables as inputs to the decoder.

Step 2 : Logically OR the outputs correspond to present minterms to obtain the output.

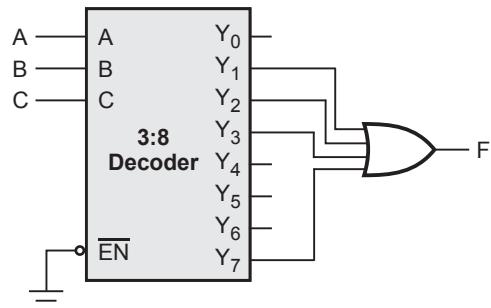


Fig. 3.13.7

Example 3.13.4 Design and implement a full adder circuit using a 3 : 8 decoder.

SPPU : May-10, Dec.-10, Marks 4

Solution : Truth table for full adder is as shown in the Table 3.13.4.

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0

Table 3.13.4 Truth table for full-adder

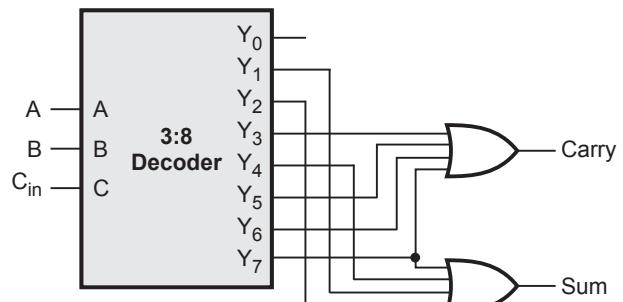


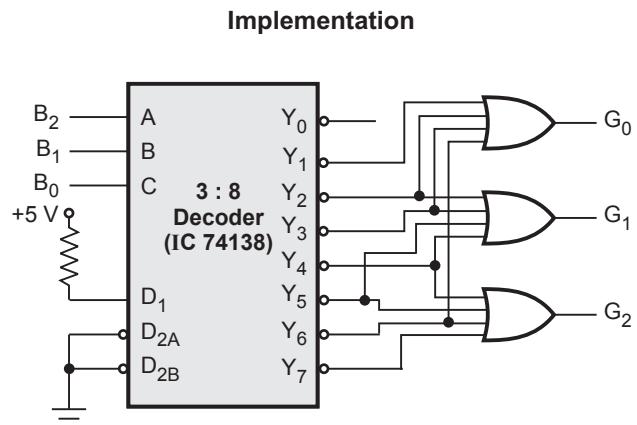
Fig. 3.13.8

Example 3.13.5 Design a 3-bit binary to 3-bit gray code converter using IC-74138.

SPPU : Dec.-13, Marks 4

Solution : Truth table

Inputs			Outputs		
B ₂	B ₁	B ₀	G ₂	G ₁	G ₀
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

**Fig. 3.13.9****Examples for Practice****Example 3.13.6** Implement the following Boolean functions using decoder and OR gates :

$$F_1(A, B, C, D) = \sum (2, 4, 7, 9)$$

$$F_2(A, B, C, D) = \sum (10, 13, 14, 15)$$

Example 3.13.7 Implement the logic circuit for full subtractor using decoder.**SPPU : May-10, Dec.-10, Marks 4****Example 3.13.8** Explain decoder (1 : 8) as a binary to gray code converter. Show your design.**SPPU : May-11, Marks 8****3.13.5 Applications of Decoder**

The uses of decoders are :

- Code converters
- Address decoding
- Implementation of combinational circuits
- BCD to 7-segment decoder

3.13.6 Decoder ICs

IC Number	Function
74138	3 : 8 Decoder
74139	Dual 2 : 4 Decoder
7442	BCD to decimal decoder
7447	BCD to 7-segment decoder

Table 3.13.5 Decoder ICs

Review Questions

1. What is decoder?
2. What is difference between demultiplexer and decoder?
3. Compare and contrast : Multiplexer and decoder.
4. Give applications of decoder.

3.14 Magnitude Comparator using 7485**SPPU : May-10,12,18, Dec.-10,12**

A comparator is a special combinational circuit designed primarily to compare the relative magnitude of two binary numbers.

Fig. 3.14.1 shows the block diagram of an n-bit comparator. It receives two n-bit numbers A and B as inputs and the outputs are $A > B$, $A = B$ and $A < B$. Depending upon the relative magnitudes of the two numbers, one of the outputs will be high.

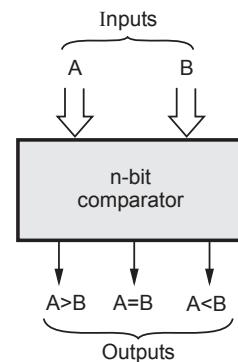


Fig. 3.14.1 Block diagram of n-bit comparator

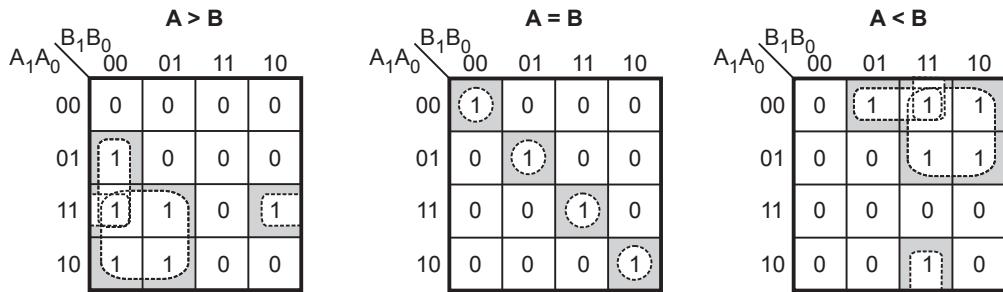
Example 3.14.1 Design 2-bit comparator using gates.

SPPU : May-18, Marks 4

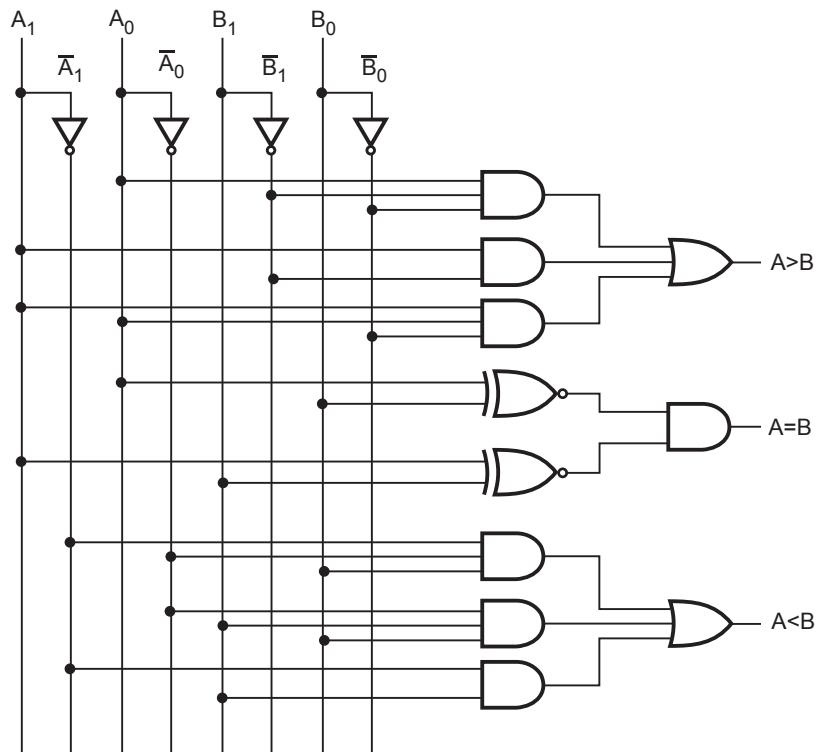
Solution : The truth table for 2-bit is given in Table 3.14.1.

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

Table 3.14.1

K-map simplification**Fig. 3.14.2**

$$\begin{aligned}
 (A = B) &= \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0 \\
 &= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0) \\
 &= (A_0 \odot B_0)(A_1 \odot B_1) \\
 (A < B) &= \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0 + \bar{A}_1B_1
 \end{aligned}$$

Logic diagram**Fig. 3.14.3**

Example 3.14.2 Design a 1-bit comparator using basic gates.

Solution : Consider two one bit number A and B. The truth table is as shown.

Inputs		Outputs		
A	B	$Y_{A=B}$	$Y_{A>B}$	$Y_{A<B}$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

3.14.1 IC 7485 (4-bit Comparator)

$$\begin{array}{c} \text{A} \quad \bar{B} \quad B \\ \bar{A} \quad 0 \quad 1 \\ \hline \text{A} \quad 0 \quad 1 \\ \bar{A} \quad 1 \quad 0 \end{array}$$

$$Y_{A=B} = \bar{A}\bar{B} + AB \\ = A \oplus B \\ = A \odot B$$

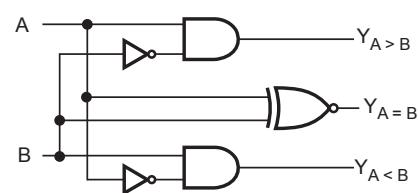
$$\begin{array}{c} \text{A} \quad B \quad 0 \quad 1 \\ \bar{A} \quad 0 \quad 0 \quad 1 \\ \hline \text{A} \quad 1 \quad 0 \quad 1 \\ \bar{A} \quad 0 \quad 1 \quad 0 \end{array}$$

$$Y_{A>B} = A\bar{B}$$

$$\begin{array}{c} \text{A} \quad B \quad 0 \quad 1 \\ \bar{A} \quad 0 \quad 0 \quad 1 \\ \hline \text{A} \quad 0 \quad 0 \quad 1 \\ \bar{A} \quad 1 \quad 0 \quad 0 \end{array}$$

$$Y_{A<B} = \bar{A}B$$

(a) k-map simplification



(b) Logic diagram

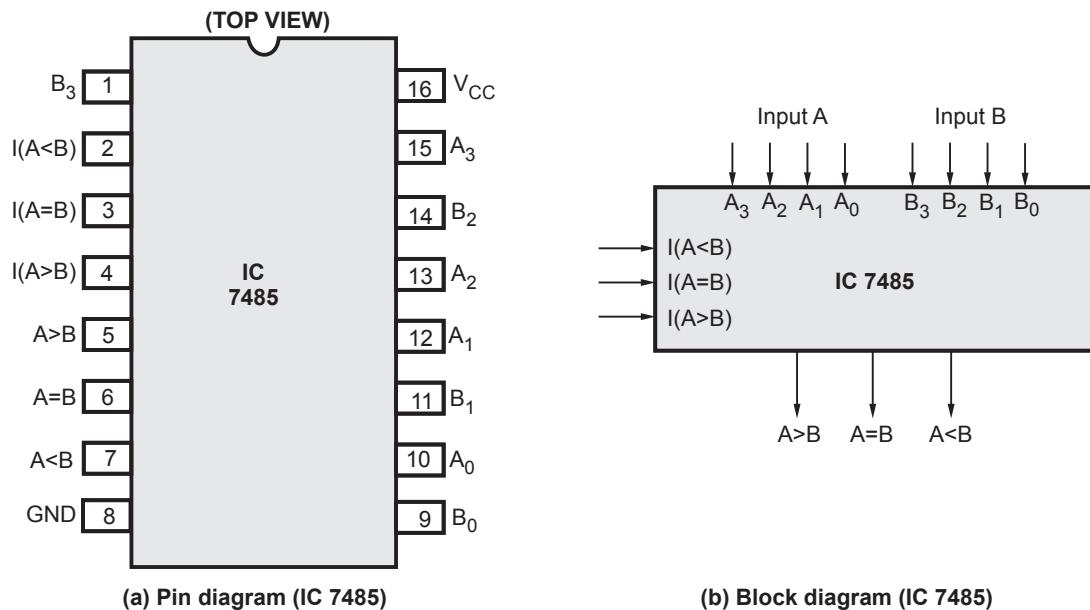
Fig. 3.14.4

IC 7485 is a 4-bit comparator. It can be used to compare two 4-bit binary words by grounding I ($A < B$), and I ($A > B$), and connector input I ($A = B$) to V_{CC} . These ICs, can be cascaded to compare words of almost any length. Its 4-bit inputs are weighted ($A_0 - A_3$) and ($B_0 - B_3$), where A_3 and B_3 are the most significant bits.

Fig. 3.14.5 shows the logic symbol and pin diagram of IC 7485. The operation of IC 7485 is described in the function table, showing all possible logic conditions.

Comparing Inputs	Cascading Inputs			Outputs				
	A	B	I($A > B$)	I($A = B$)	I($A < B$)	$A > B$	$A = B$	$A < B$
$A > B$	X	X	X			1	0	0
$A = B$	1	0	0			1	0	0
	X	1	X			0	1	0
	0	0	1			0	0	1
	0	0	0			1	0	1
	1	0	1			0	0	0
$A < B$	X	X	X			0	0	1

Table 3.14.2 Function table for IC 7485



Example 3.14.3 Design an 8-bit comparator using two 7485 ICs.

Solution : Fig. 3.14.6 shows an 8-bit comparator using two 7485 ICs.

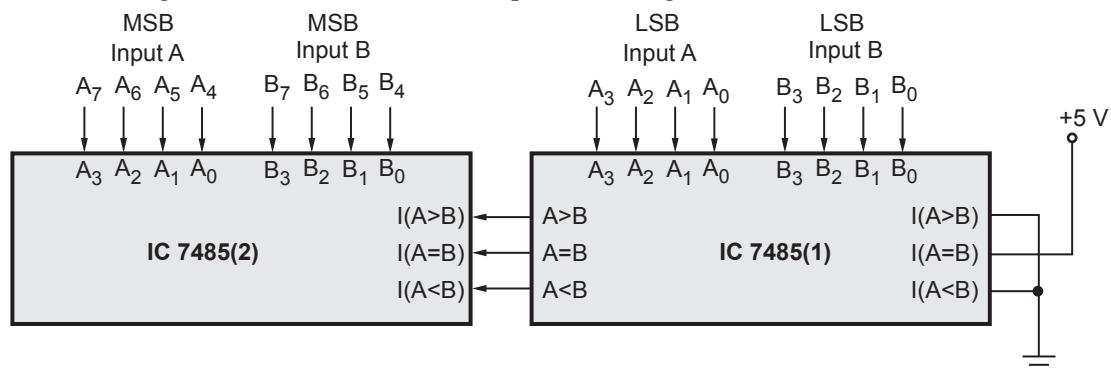


Fig. 3.14.6 8-bit comparator

Example 3.14.4 Design a 5-bit magnitude comparator using comparator IC 7485

Solution : Truth Table

A ₀	B ₀	I(A > B)	I(A = B)	I(A < B)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

K-map simplification

		$A > B$		$A = B$		$A < B$				
		B_0	0	1	B_0	0	1	B_0	0	1
A_0	0	0	0	0	1	0	1	0	0	1
1	1	1	0	0	0	1	1	1	0	0

$$(A > B) = A_0 \bar{B}_0$$

$$(A = B) = \bar{A}_0 \bar{B}_0 + A_0 B_0 = A_0 \oplus B_0$$

$$(A < B) = \bar{A}_0 B_0$$

Implementation

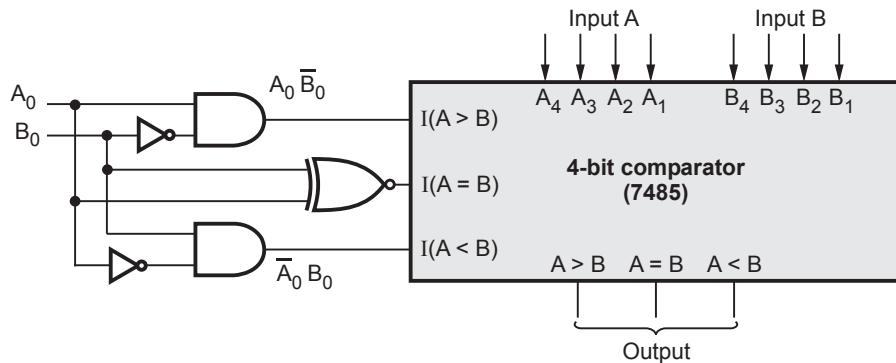


Fig. 3.14.7

Example 3.14.5 Design a 4-bit magnitude comparator to compare two 4-bit numbers.

Solution : If the number of bits to be compared is more than two bits, the truth tables and hence the circuit becomes more complicated. Thus, we can implement 4-bit comparator with some different approach. In this approach 4-bit adder is used to generate $A \neq B$, $A > B$ and $A < B$ signals. Consider, there are two numbers A and B, each of n bits. Let us assume A is greater than B. This condition can be written as

$$A > B \Rightarrow A - B > 0$$

$$\Rightarrow A + \bar{B} + 1 > 0 \quad \because \bar{B} + 1 \text{ is 2's complement of } B$$

$$\Rightarrow A + \bar{B} > -1$$

We represent -1 , in n bit binary numbers as

$$1_n 1_{n-1} \dots 1_2 1_1 1_0$$

The above equation says that, if A is greater than B, then the sum of A and \bar{B} should be greater than $1_n 1_{n-1} \dots 1_2 1_1 1_0$. If n adder is used to add A and \bar{B} , and the final carryout is 1 then we can say that $A + \bar{B} > 1_n 1_{n-1} \dots 1_2 1_1 1_0$, i.e. $A > B$.

If this condition is not satisfied, we can say that $A \leq B$. Now we can check of equality by checking the following equation.

$$A + \bar{B} = 1_n 1_{n-1} \dots 1_2 1_1 1_0$$

If $A > B$ and $A = B$, both conditions are not true then we can say that $A < B$.

The Fig. 3.14.8 shows the circuit diagram of comparator for $n = 4$

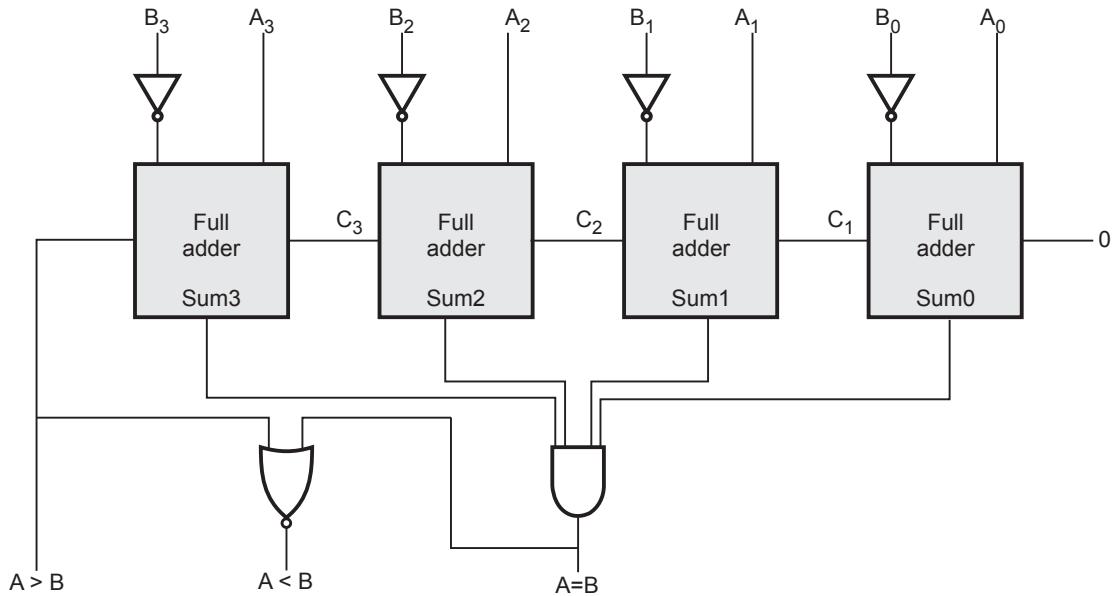


Fig. 3.14.8 4-bit magnitude comparator

Review Questions

1. Explain with block diagram, the function table of 4-bit comparator IC 7485.
2. Explain the working of cascaded mode magnitude comparator using IC 7485 ?

SPPU : May-10, Dec.-10, Marks 6

3. Explain the working of magnitude comparator 7485. Discuss the truth table for the same.

SPPU : May-12, Marks 8

4. Design 2-bit magnitude comparator using logic gates. Assume that A and B are 2-bit inputs. The outputs of comparator should be $A > B$, $A = B$, $A < B$.

SPPU : Dec.-12, Marks 8

3.15 Parity Generator and Checker using 74180

SPPU : Dec.-11

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even. The message, including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker. In this section, we will study the design of parity generator and parity checker.

3-bit Message			Odd Parity Bit	Even Parity Bit
A	B	C		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

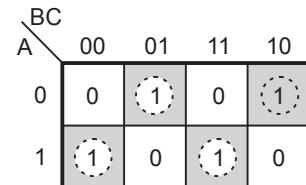
Table 3.15.1 Parity generator truth table for even and odd parity

In even parity the added parity bit will make the total number of 1s an even amount. In odd parity the added parity bit will make the total number of 1s an odd amount. Table 3.15.1 shows the 3-bit message with even parity and odd parity.

Let us design the even parity generator.

K-map simplification

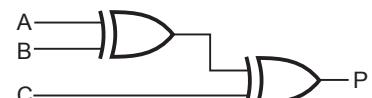
$$\begin{aligned}
 P &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB C \\
 &= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC) \\
 &= \bar{A}(B \oplus C) + A(B \odot C) \\
 &= \bar{A}(B \oplus C) + A(\overline{B \oplus C}) \\
 &= A \oplus (B \oplus C)
 \end{aligned}$$

**Fig. 3.15.1 (a)**

Logic diagram

The three bits in the message together with the parity bit are transmitted to their destination, where they are applied to the parity checker circuit. The parity checker circuit checks for possible errors in the transmission.

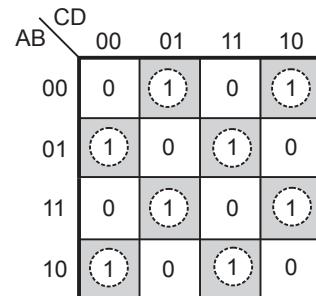
Since the information was transmitted with even parity, the four bits received must have an even number of 1s. An error occurs during the transmission if the four bits received have an odd number of 1s, indicating that one bit has changed in value during transmission. The output of the parity checker, is denoted by PEC (Parity Error Check). It will be equal to 1, if an error occurs, that is, if the four bits received have an odd number of 1s. The table shows the truth table for the even parity checker.

**Fig. 3.15.1 (b)**

Four bits received				Parity Error Check
A	B	C	D	PEC
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
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	1
1	1	1	0	1
1	1	1	1	0

K-map simplification

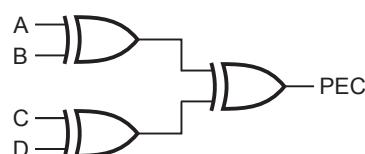
$$\begin{aligned}
 PEC &= \overline{A}\overline{B}(\overline{C}D + C\overline{D}) + \overline{A}B(\overline{C}\overline{D} + CD) \\
 &\quad + A\overline{B}(\overline{C}D + C\overline{D}) + A\overline{B}(\overline{C}\overline{D} + CD) \\
 &= \overline{A}\overline{B}(C \oplus D) + \overline{A}B(\overline{C} \oplus \overline{D}) \\
 &\quad + A\overline{B}(C \oplus D) + A\overline{B}(\overline{C} \oplus \overline{D}) \\
 &= (\overline{A}\overline{B} + A\overline{B})(C \oplus D) + (\overline{A}B + A\overline{B})(\overline{C} \oplus \overline{D}) \\
 &= (A \oplus B)(C \oplus D) + (A \oplus B)(\overline{C} \oplus \overline{D}) \\
 &= (A \oplus B) \oplus (C \oplus D)
 \end{aligned}$$



$$PEC = (A \oplus B) \oplus (C \oplus D)$$

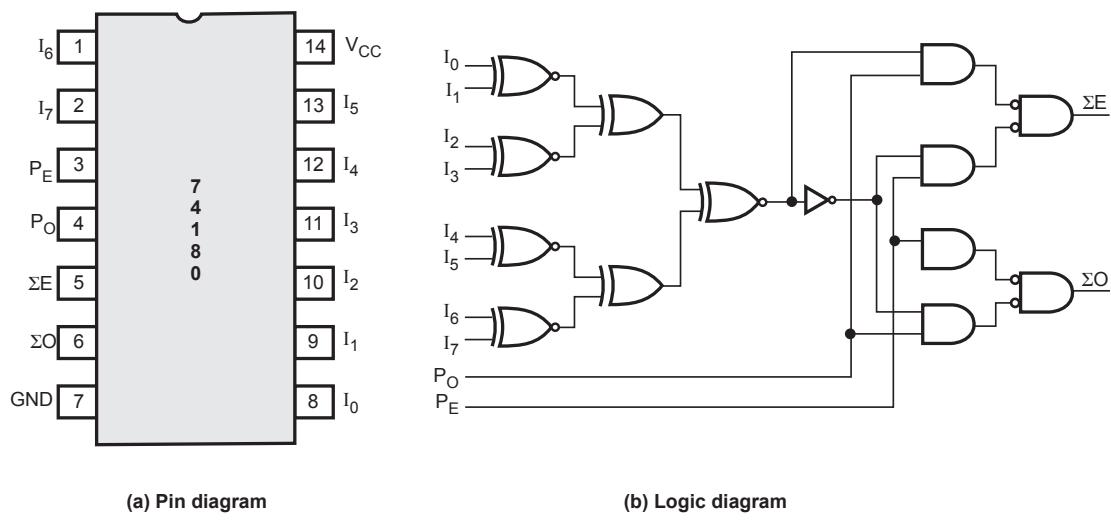
Fig. 3.15.2 (a)**Table 3.15.2 Truth table for even parity checker****Logic diagram**

Let us study the standard IC for parity generator and checker.

**Fig. 3.15.2 (b)****IC 74180 : Parity Generator/Checker**

The 74180 is a 9-bit parity generator or checker commonly used to detect errors in high speed data transmission or data retrieval systems. Both **even** and **odd** parity enable inputs and parity outputs are available for generating or checking parity on 8-bits.

The Fig. 3.15.3 (a, b) shows pin diagram and logic diagram for 74180.



(a) Pin diagram

(b) Logic diagram

Fig. 3.15.3

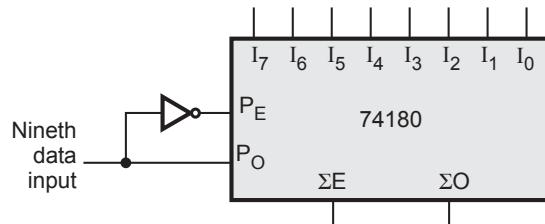
Table 3.15.3 shows the function table for 74180

As shown in the Table 3.15.3, true active-HIGH or true active-LOW parity can be generated at both the even or odd outputs. True active-HIGH parity is established with Even parity enable input (P_E) set HIGH and the Odd parity enable input (P_O) set LOW. True active LOW parity is established when P_E is LOW and P_O is HIGH. When both are enable inputs are at the same logic level, both outputs will be forced to the opposite logic level.

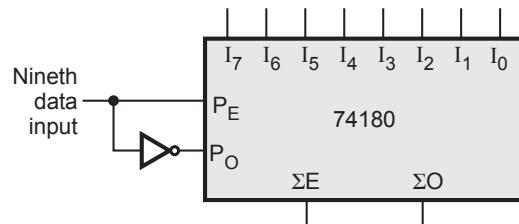
Number of HIGH data Inputs (I ₀ - I ₇)	Inputs		Outputs	
	P _E	P _O	Σ E	Σ O
Even	1	0	1	0
Odd	1	0	0	1
Even	0	1	0	1
Odd	0	1	1	0
X	1	1	0	0
X	0	0	1	1

Table 3.15.3 Function table for 74180

Parity checking of a 9-bit word (8-bits plus parity) is possible by using the two enable inputs plus an inverter as the ninth data input. To check for true active-HIGH parity,



(a) Check for true active-HIGH parity



(b) Check for true active-LOW parity

Fig. 3.15.4

the ninth data input is tied to the P_O input and an inverter is connected between the P_O and P_E inputs as shown in the Fig. 3.15.4 (a). To check for true active-LOW parity, the ninth data input is tied to the P_E input and an inverter is connected between the P_E and P_O inputs, as shown in the Fig. 3.15.4 (b).

Expansion to larger word sizes is accomplished by serially cascading the 74180 in 8-bit increments. The even and odd parity outputs of the first stages are connected to the corresponding P_E and P_O inputs, respectively, of the succeeding stage, as shown in the Fig. 3.15.5.

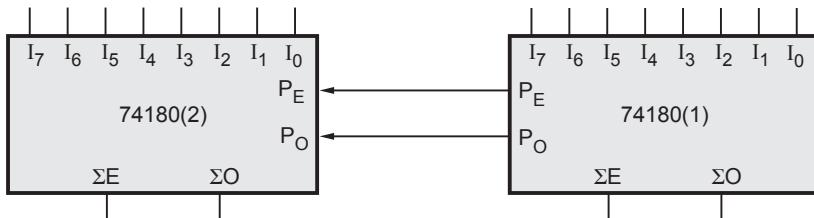


Fig. 3.15.5 Expansion to the larger word

Review Questions

1. What do you mean by parity ? How are EX-OR gates used in even parity generator ?
2. Write a short note on parity generator/checker.
3. What do you mean by parity ? How does IC 74180 work ? Design 9 bit even parity generator using the same.

SPPU : Dec.-11, Marks 8



Notes

SUBJECT CODE : 210245

As per Revised Syllabus of
SAVITRIBAI PHULE PUNE UNIVERSITY
Choice Based Credit System (CBCS)
S.E. (Comp.) Semester - I

DIGITAL ELECTRONICS AND LOGIC DESIGN

(For END SEM Exam - 70 Marks)

Atul P. Godse

M.S. Software Systems (BITS Pilani)
B.E. Industrial Electronics

Formerly Lecturer in Department of Electronics Engg.
Vishwakarma Institute of Technology
Pune

Dr. Mrs. Deepali A. Godse

M.E., Ph.D. (Computer Engg.)
Head of Information Technology Department,
Bharati Vidyapeeth's College of Engineering for Women,
Pune

Dr. Vinod Vijaykumar Kimbahune

Ph.D in Computer Engineering
Associate Professor
Smt. Kashibai Navale College of Engineering.
Vadgoan(bk)

Dr. Sunil M. Sangve

Ph.D in Computer Science and Engineering
Professor and Head, Computer Engineering
Zeal College of Engineering and Research,
Narhe, Pune



DIGITAL ELECTRONICS AND LOGIC DESIGN

(For END SEM Exam - 70 Marks)

Subject Code : 210245

S.E. (Computer Engineering) Semester - I

First Edition : September 2020

© Copyright with A. P. Godse, Dr. D. A. Godse

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-332-2151-1



PREFACE

The importance of **Digital Electronics and Logic Design** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Digital Electronics and Logic Design**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors

A. P. Godse

Dr. D. A. Godse

Dr. V. V. Kimbahune

Dr. S. M. Sangee

Dedicated to God

SYLLABUS

Digital Electronics and Logic Design (210245)

Credit	Examination Scheme and Marks
03	End_Semester (TH) : 70 Marks

Unit III Sequential Logic Design

Flip-Flop : SR, JK,D,T, Preset and Clear, Master Slave JK Flip Flops, Truth Tables and Excitation tables, Conversion from one type to another type of Flop-Flop. Registers: SISO, SIPO, PISO, PIPO, Shift Registers, Bidirectional Shift Register, Ring Counter, Universal Shift Register Counters : Asynchronous Counter, Synchronous Counter, BCD Counter, Johnson Counter, Modulus of the counter (IC7490), Synchronous Sequential Circuit Design :Models- Moore and Mealy, State diagram and State Table ,Design Procedure, Sequence Generator and detector. **(Chapters - 4, 5, 6, 7)**

Unit IV Algorithmic State Machines and Programmable Logic Devices

Algorithmic State Machines : Finite State Machines (FSM) and ASM, ASM charts, notations, construction of ASM chart and realization for sequential circuits. PLDS : PLD, ROM as PLD, Programmable Logic Array (PLA), Programmable Array Logic (PAL), Designing combinational circuits using PLDs. **(Chapters - 8, 9)**

Unit V Logic Families

Classification of logic families : Unipolar and Bipolar Logic Families, Characteristics of Digital ICs : Fan-in, Fan-out, Current and voltage parameters, Noise immunity, Propagation Delay, Power Dissipation, Figure of Merits, Operating Temperature Range, power supply requirements. **Transistor-Transistor Logic :** Operation of TTL NAND Gate (Two input), TTL with active pull up, TTL with open collector output, Wired AND Connection, Tristate TTL Devices, TTL characteristics. CMOS : CMOS Inverter, CMOS characteristics, CMOS configurations- Wired Logic, Open drain outputs. **(Chapter - 10)**

Unit VI Introduction to Computer Architecture

Introduction to Ideal Microprocessor - Data Bus, Address Bus, Control Bus. Microprocessor based Systems - Basic Operation, Microprocessor operation, Block Diagram of Microprocessor.

Functional Units of Microprocessor - ALU using IC 74181, Basic Arithmetic operations using ALU IC 74181, 4-bit Multiplier circuit using ALU and shift registers. Memory Organization and Operations, digital circuit using decoder and registers for memory operations. **(Chapter - 11)**

TABLE OF CONTENTS

Unit - III

Chapter - 4 Flip-Flops	(4 - 1) to (4 - 36)
4.1 Introduction	4 - 2
4.1.1 Comparison between Combinational and Sequential Logic Circuits	4 - 2
4.1.2 Clock.....	4 - 3
4.2 One-Bit Memory Cell	4 - 3
4.3 Latches	4 - 4
4.3.1 SR Latch	4 - 4
4.3.2 Gated SR Latch	4 - 6
4.3.3 Gated D Latch	4 - 6
4.4 Flip-Flops.....	4 - 7
4.4.1 Latches Vs Flip-Flops.....	4 - 7
4.4.2 Level and Edge Triggering	4 - 7
4.4.3 SR Flip-Flop	4 - 9
4.4.4 D Flip-Flop	4 - 11
4.4.5 JK Flip-Flop.....	4 - 13
4.4.5.1 JK Flip-Flop using NAND Gates	4 - 15
4.4.5.2 Race-around Condition	4 - 15
4.4.6 Master-Slave SR Flip-Flop.....	4 - 16
4.4.7 Master-Slave JK Flip-Flop	4 - 17
4.4.8 T Flip-Flop	4 - 18
4.4.9 Preset and Clear	4 - 19
4.5 Excitation Tables.....	4 - 22
4.5.1 SR Flip-Flop	4 - 22
4.5.2 JK Flip-Flop.....	4 - 23
4.5.3 D Flip-Flop	4 - 24
4.5.4 T Flip-Flop	4 - 24
4.6 Conversion from One Type to Another Type of Flip-Flop.....	4 - 24

4.6.1 SR Flip-Flop to D Flip-Flop	4 - 24
4.6.2 SR Flip-Flop to JK Flip-Flop	4 - 25
4.6.3 SR Flip-Flop to T Flip-Flop.....	4 - 26
4.6.4 JK Flip-Flop to T Flip-Flop	4 - 27
4.6.5 JK Flip-Flop to D Flip-Flop.....	4 - 28
4.6.6 D Flip-Flop to T Flip-Flop	4 - 28
4.6.7 T Flip-Flop to D Flip-Flop	4 - 29
4.6.8 JK Flip-Flop to SR Flip-Flop	4 - 30
4.6.9 D Flip-Flop to SR Flip-Flop	4 - 30
4.6.10 T Flip-Flop to SR Flip-Flop.....	4 - 31
4.6.11 D Flip-Flop to JK Flip-Flop.....	4 - 32
4.7 Applications of Flip-Flops.....	4 - 32
4.7.1 Bounce Elimination Switch.....	4 - 33
4.7.2 Registers	4 - 34
4.7.3 Counters	4 - 34

Chapter - 5 Registers	(5 - 1) to (5 - 16)
5.1 Introduction	5 - 2
5.1.1 Buffer Register	5 - 2
5.1.2 Controlled Buffer Register	5 - 2
5.2 Shift Registers	5 - 3
5.3 Types of Shift Registers.....	5 - 4
5.3.1 Serial In Serial Out (SISO) Shift Register	5 - 4
5.3.2 Serial In Parallel Out (SIPO) Shift Register	5 - 8
5.3.3 Parallel In Serial Out (PISO) Shift Register	5 - 8
5.3.4 Parallel In Parallel Out (PIPO) Shift Register.....	5 - 9
5.3.5 Bidirectional Shift Register.....	5 - 10
5.4 Universal Shift Register.....	5 - 12
5.5 Applications of Shift Registers	5 - 14
5.5.1 Delay Line	5 - 14
5.5.2 Serial-to-Parallel Converter	5 - 14
5.5.3 Parallel-to-Serial Converter	5 - 14
5.5.4 Shift Register Counters.....	5 - 14

5.5.5 Pseudo-Random Binary Sequence (PRBS) Generator	5 - 15
5.5.6 Sequence Generator.....	5 - 15
5.5.7 Sequence Detector	5 - 15

Chapter - 6 Counters	(6 - 1) to (6 - 42)
----------------------------------	----------------------------

6.1 Introduction	6 - 2
6.2 Ripple / Asynchronous Counters	6 - 4
6.2.1 Asynchronous / Ripple Down Counter	6 - 7
6.2.2 Asynchronous Up / Down Counter	6 - 9
6.2.3 Decoding Gates.....	6 - 10
6.2.4 Problem Faced by Ripple Counters (Glitch Problem).....	6 - 12
6.3 Design of Ripple (Asynchronous) Counters	6 - 13
6.4 Synchronous Counters.....	6 - 16
6.4.1 2-bit Synchronous Binary Up Counter.....	6 - 16
6.4.2 3-bit Synchronous Binary Up Counter.....	6 - 17
6.4.3 4-bit Synchronous Binary Up Counter.....	6 - 18
6.4.4 Synchronous Down and Up/Down Counters	6 - 19
6.5 Design of Synchronous Counters	6 - 20
6.6 Lock-Out.....	6 - 27
6.7 IC 7490 (Decade Binary Counter).....	6 - 30
6.8 Ring Counter	6 - 36
6.9 Johnson or Twisting Ring or Switch Tail Counter.....	6 - 38

Chapter - 7 Synchronous Sequential Circuit Design	(7 - 1) to (7 - 64)
---	----------------------------

7.1 Clocked Sequential Circuits.....	7 - 2
7.1.1 Moore Model	7 - 2
7.1.2 Mealy Model	7 - 3
7.1.3 Moore vs Mealy Circuit Models.....	7 - 4
7.1.4 Representations of Sequential Circuits.....	7 - 4
7.1.4.1 State Diagram	7 - 4
7.1.4.2 State Table	7 - 5
7.1.4.3 Transition Table	7 - 5
7.2 Design of Clocked Sequential Circuits.....	7 - 7

7.2.1 State Reduction.....	7 - 7
7.2.2 State Assignment	7 - 10
7.2.3 Choice of Flip-Flops and Derivation of Next State and Output.....	7 - 11
7.2.4 State Assignment Problem.....	7 - 17
7.2.5 Design with Unused States	7 - 20
7.3 Sequence Generator.....	7 - 37
7.3.1 Sequence Generator using Counters	7 - 37
7.3.2 Sequence Generator using Shift Register	7 - 39
7.4 Sequence Detector	7 - 46

Unit - IV

Chapter - 8 Algorithmic State Machines	(8 - 1) to (8 - 24)
8.1 Introduction	8 - 2
8.1.1 ASM Symbols / Notations.....	8 - 2
8.1.2 Salient Features of ASM Chart	8 - 4
8.2 Design of Simple Controller	8 - 5
8.2.1 K-map Simplification Method	8 - 7
8.2.2 Multiplexer Control Method	8 - 8
8.3 ASM Examples : Sequence Generator, Types of Counters	8 - 11
Chapter - 9 Programmable Logic Devices	(9 - 1) to (9 - 42)
9.1 Introduction	9 - 2
9.2 PROM (Programmable Read Only Memory).....	9 - 3
9.2.1 AND Matrix	9 - 4
9.2.2 OR Matrix.....	9 - 7
9.2.3 Invert / Non-invert Matrix	9 - 8
9.2.4 Combinational Logic Implementation using PROM	9 - 8
9.3 PLA (Programmable Logic Array)	9 - 13
9.3.1 Input Buffer.....	9 - 14
9.3.2 Output Buffer	9 - 14
9.3.3 Output through Flip-Flops	9 - 14
9.3.4 Implementation of Combination Logic Circuit using PLA	9 - 15

9.4 PAL (Programmable Array Logic)	9 - 30
9.4.1 Implementation of Combinational Logic Circuit using PAL	9 - 31
9.5 Comparison between PROM, PLA and PAL.....	9 - 41

Unit - V

Chapter - 10 Logic Families	(10 - 1) to (10 - 28)
10.1 Classification of Logic Families.....	10 - 2
10.2 Characteristics of Digital ICs	10 - 3
10.3 Transistor-Transistor Logic (TTL).....	10 - 6
10.3.1 2-Input TTL NAND Gates	10 - 6
10.3.2 Totem-Pole Output / Active Pull-Up	10 - 7
10.3.3 Wired Logic - Open Collector Output.....	10 - 8
10.3.4 Comparison between Totem-Pole and Open-Collector Outputs.....	10 - 10
10.3.5 Tri-state TTL Inverter.....	10 - 10
10.3.6 Tri-state TTL NAND Gate.....	10 - 12
10.3.7 Standard TTL Characteristics	10 - 13
10.3.8 Advantages and Disadvantages of TTL Family.....	10 - 14
10.4 Complementary MOS (CMOS)	10 - 16
10.4.1 CMOS Inverter	10 - 16
10.4.2 CMOS NAND Gate	10 - 18
10.4.3 CMOS NOR Gate	10 - 20
10.4.4 CMOS Characteristics	10 - 22
10.4.5 Wired Logic	10 - 23
10.4.6 Open Drain Outputs	10 - 23
10.4.7 Advantages and Disadvantages of CMOS Family	10 - 23
10.5 Interfacing TTL and CMOS Families	10 - 24
10.5.1 TTL Driving CMOS	10 - 25
10.5.2 CMOS Driving TTL	10 - 26
10.6 Comparison between TTL and CMOS Families	10 - 28

Unit - VI

Chapter - 11 Introduction to Computer Architecture (11 - 1) to (11 - 36)

11.1 Introduction to Ideal Microprocessor.....	11 - 2
11.2 Data Bus, Address Bus and Control Bus.....	11 - 3
11.2.1 Address Bus	11 - 4
11.2.2 Data Bus.....	11 - 5
11.2.3 Control Bus	11 - 5
11.2.4 Multiplexed Address and Data Bus	11 - 6
11.2.5 Address Bus and Data Bus Drivers.....	11 - 6
11.2.6 Sharing of Address, Data and Control Bus with DMA	11 - 8
11.2.7 Control Signals to Provide Facility of Interrupt Driven I/O	11 - 9
11.3 Microprocessor Based Systems - Basic Operation.....	11 - 9
11.4 Microprocessor Operation.....	11 - 14
11.5 Block Diagram of Microprocessor with its Functional Units.....	11 - 16
11.5.1 Arithmetic and Logic Unit (ALU)	11 - 16
11.5.2 Registers	11 - 17
11.5.3 Instruction Decoder	11 - 20
11.5.4 Control Logic	11 - 20
11.5.5 Internal Data Bus	11 - 20
11.5.6 Subroutine, Stack and Stack Pointer.....	11 - 20
11.6 ALU using IC 74181 - Basic Arithmetic operations	11 - 22
11.7 The 4-bit Multiplier Circuit using ALU and Shift Registers.....	11 - 28
11.8 Memory Organization	11 - 30
11.8.1 Block Diagram of Memory Unit.....	11 - 31
11.8.2 Digital Circuit using Decoder and Registers for Memory Operations	11 - 32
11.9 Memory Operation	11 - 33
11.9.1 Read Operation.....	11 - 33
11.9.2 Write Operation	11 - 35

UNIT - III

4

Flip-Flops

Syllabus

SR, JK,D,T, Preset and Clear, Master Slave JK Flip Flops, Truth Tables and Excitation tables, Conversion from one type to another type of Flop-Flop.

Contents

4.1	<i>Introduction</i>	May-17,19,	Marks 4
4.2	<i>One-Bit Memory Cell</i>				
4.3	<i>Latches</i>				
4.4	<i>Flip-Flops</i>	May-05,06,07,11,12,13,		
		Dec.-05,06,10	Marks 10
4.5	<i>Excitation Tables</i>	Dec.-13, May-15,	Marks 2
4.6	<i>Conversion from One Type to Another Type of Flip-Flop</i>	May-06,10,12,		
		Dec.-16,17,19	Marks 4
4.7	<i>Applications of Flip-Flops</i>				

4.1 Introduction

SPPU : May-17,19

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement cannot be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback from the output back to the input.

Fig. 4.1.1 shows the block diagram of sequential circuit/Finite State Machine (FSM). As shown in the Fig. 4.1.1, memory elements are connected to the combinational circuit as a feedback path.

The information stored in the memory elements at any given time defines the **present state** of the sequential circuit. The present state and the external inputs determine the outputs and the **next state** of the sequential circuit. Thus we can specify the sequential circuit by a time sequence of external inputs, internal states (present states and next states), and outputs. The counters and registers are the common examples of sequential circuits.

The memory element used in sequential circuits is a flip-flop which is capable of storing 1-bit binary information.

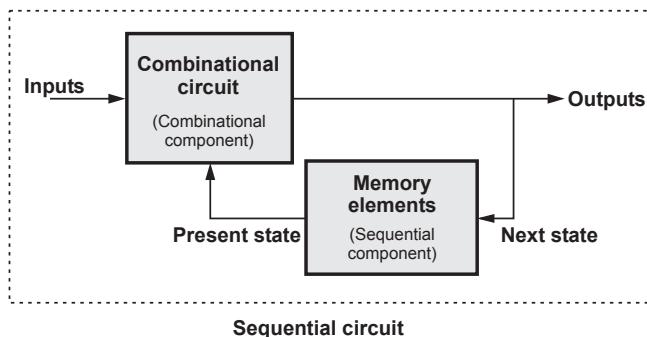


Fig. 4.1.1 Block diagram of sequential circuit / FSM

4.1.1 Comparison between Combinational and Sequential Logic Circuits

Sr. No.	Combinational circuits	Sequential circuits
1.	In combinational circuits, the output variables are at all times dependent on the combination of input variables.	In sequential circuits, the output variables depend not only on the present input variables but they also depend upon the past history of these input variables.
2.	Memory unit is not required in combinational circuits.	Memory unit is required to store the past history of input variables in the sequential circuit.
3.	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.	Sequential circuits are slower than the combinational circuits.

4.	Combinational circuits are easy to design.	Sequential circuits are comparatively harder to design.
5.	Parallel adder is a combinational circuit.	Serial adder is a sequential circuit.

Table 4.1.1 Comparison between combinational and sequential circuits**4.1.2 Clock**

A clock signal is a particular type of signal that oscillates between a high and a low state and is utilized to co-ordinate actions of circuits. It is produced by a clock generator. The most common clock signal is in the form of a square wave with a 50 % duty cycle, usually with a fixed, constant frequency as shown in Fig. 4.1.2. Circuits using the clock signal for synchronization may become active at either the rising edge, falling edge or in the case of double data rate, both in the rising and in the falling edges of the clock cycle.

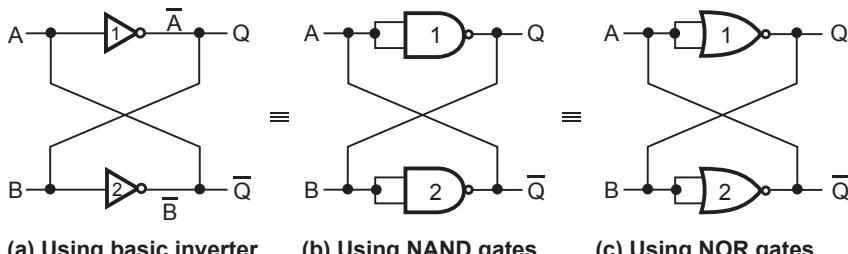
The time required to complete one cycle is called '**clock period**' or '**clock cycle**'. Ideally, the clock signal should have sharp transitions from one level to other as shown in Fig. 4.1.2.

**Fig. 4.1.2 Clock signal****Review Questions**

1. Define sequential logic circuit.
 2. What is flip-flop ?
 3. Give the comparison between combinational and sequential logic circuits.
- SPPU : May-17,19, Marks 4**
4. What is clock ? State its use.

4.2 One-Bit Memory Cell

The Fig. 4.2.1 shows the basic bistable element used in latches and flip-flops. The basic bistable element has two outputs Q and \bar{Q} . It has two cross-coupled inverters, i.e., the output of the first inverter is connected as an input to the second inverter and the output of second inverter is connected as an input to the first inverter.

**Fig. 4.2.1 Basic bistable element (1-bit memory cell)**

The basic bistable element circuit has two stable states logic 0 and logic 1, hence the name 'bistable'. To illustrate this, assume $A = 0$. When $A = 0$, the output of inverter 1 is 1 (\bar{A}), i.e., $Q = 1$. Since the output of inverter 1 is the input to the inverter 2, $\bar{A} = B = 1$. Consequently, the output of inverter 2, i.e., \bar{B} is 0. Since the output of the inverter 2 is connected to the input of the inverter 1, $\bar{Q} = \bar{B} = A = 0$. We have assumed same value for A. Thus, the circuit is stable with $\bar{Q} = A = \bar{B} = 0$ and $Q = \bar{A} = B = 1$. Using similar explanation it is easy to show that if it is assumed that $A = 1$, the basic bistable element is stable with $\bar{Q} = A = \bar{B} = 1$ and $Q = \bar{A} = B = 0$. This is a second stable condition of the basic bistable element.

The two stable states of basic bistable elements are used to store two binary elements, 0 and 1. In positive logic system, state $Q = 1$ is used to store logic 1, and state $Q = 0$ is used to store logic 0. It is important to note that the two outputs are complementary. That is when $Q = 0$, $\bar{Q} = 1$; and when $Q = 1$, $\bar{Q} = 0$.

From the above discussion we can note following things about the basic bistable element.

1. The outputs Q and \bar{Q} are always complementary.
2. The circuit has two stable states. The state corresponds to $Q = 1$ is referred to as **1 state or set state** and state corresponds to $Q = 0$ is referred to as **0 state or Reset state**.
3. If the circuit is in the set (1) state, it will remain in the set state and if the circuit is in the reset (0) state, it will remain in the reset state. This property of the circuit shows that it can store 1-bit of digital information. Therefore, the circuit is called a **1-bit memory cell**.
4. The 1-bit information stored in the circuit is locked or latched in the circuit. Therefore, this circuit is also referred to as a **latch**.

Review Question

1. Explain the operation of one-bit memory cell.

4.3 Latches

4.3.1 SR Latch

Fig. 4.3.1 shows SR latch which is 1-bit memory cell. As shown in the Fig. 4.3.1, two inverters 3 and 4 are connected to enter the digital information. Input for gate 3 is S and input for gate 4 is R. This latch is also called **RS latch**.

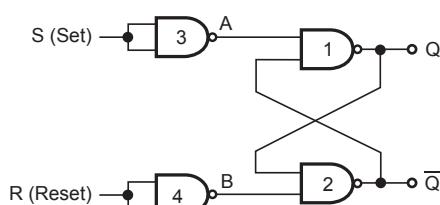


Fig. 4.3.1 SR latch

For understanding the circuit operation, we must first determine the output of NAND gate whose one of the input is logic 0 and accordingly we have to determine the output of other NAND gate in the cross coupled circuit. Because the output of NAND gate is 1 if any one input is 0. The circuit operation is as follows. In Fig. 4.3.2, the output of shaded NAND gate is determined first, and the 0 input that decides the output of shaded NAND as 1 is shown in bold.

Case 1 : $S = R = 0$

In this case, $\bar{S} = \bar{R} = 1$. If Q is 1, Q and B inputs for NAND gate 2 are both 1 and hence output $\bar{Q} = 0$. Since $\bar{Q} = 0$ and $\bar{S} = 1$, the output of NAND gate 1 is 1, i.e. $Q = 1$.

If Q is 0, Q and \bar{R} inputs for NAND gate 2 are 0 and 1, and hence output $\bar{Q} = 1$. Since $\bar{Q} = 1$ and $\bar{S} = 1$, the output of NAND gate 1 is 0, i.e., $Q = 0$.

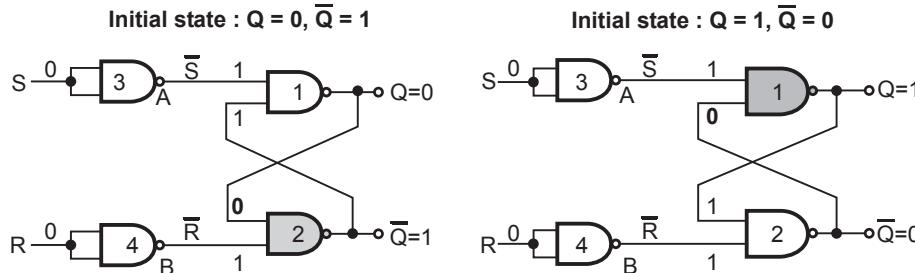


Fig. 4.3.2

This shows that when $S = R = 0$, the outputs do not change.

Case 2 : $S = 1$ and $R = 0$

In this case, $\bar{S} = 0$ and $\bar{R} = 1$. Since $\bar{S} = 0$, the output of NAND gate 1, $Q = 1$ (Recall that, for NAND any one or more input is 0, the output is 1) For NAND gate 2, both inputs Q and \bar{R} are 1, thus output $\bar{Q} = 0$.

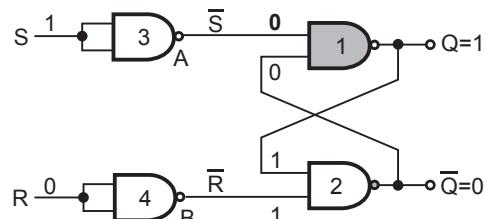


Fig. 4.3.3

The inputs $S = 1$ and $R = 0$, makes $Q = 1$, i.e., **set** state.

Case 3 : $S = 0$ and $R = 1$

In this case, $\bar{S} = 1$ and $\bar{R} = 0$. Since $\bar{R} = 0$, the output of NAND gate 2, $\bar{Q} = 1$. For NAND gate 1, both inputs \bar{Q} and \bar{S} are 1, thus output $Q = 0$.

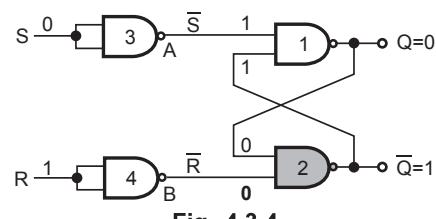


Fig. 4.3.4

The inputs $S = 0$ and $R = 1$, makes $Q = 0$, i.e., **reset** state.

Case 4 : S = 1 and R = 1

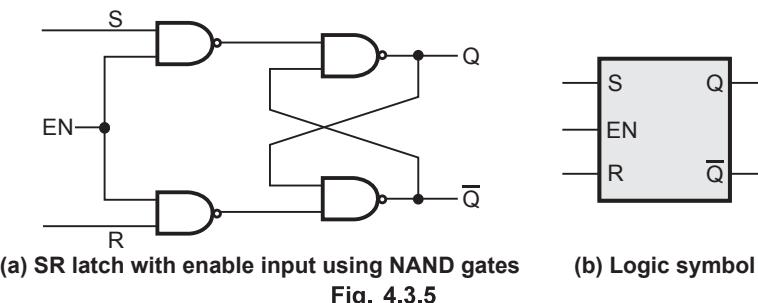
When S = R = 1, both the outputs Q and \bar{Q} try to become 1 which is not allowed and therefore, this input condition is prohibited.

4.3.2 Gated SR Latch

In the SR latch we have seen that output changes occur immediately after the input changes occur i.e. the latch is sensitive to its S and R inputs at all times. However, it can easily be modified to create a latch that is sensitive to these inputs only when an enable input is active. Such a latch with enable input is known as **gated SR latch**. It is as shown in the Fig. 4.3.5. The Table 4.3.1 shows the truth table for gated latch. As shown by truth table, the circuit behaves like a SR latch when EN = 1, and retains its previous state when EN = 0.

EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No Change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate
1	1	1	1	X	
0	X	X	0	0	No Change (NC)
0	X	X	1	1	

Table 4.3.1 Truth table for SR latch with enable input



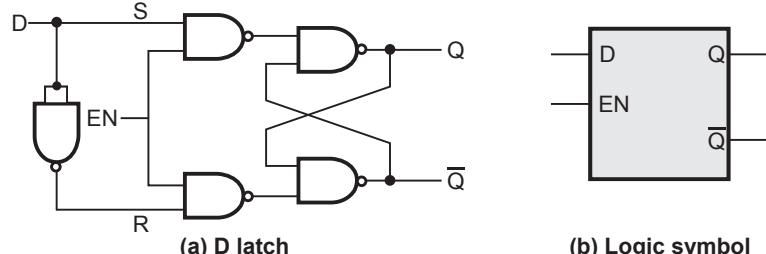
(a) SR latch with enable input using NAND gates (b) Logic symbol

Fig. 4.3.5

4.3.3 Gated D Latch

Looking at the truth table of the SR latch we can realize that when both inputs are same the output either does not change or it is invalid (Inputs \rightarrow 00, no change and inputs \rightarrow 11, invalid). In many practical applications, these input conditions are not required. These input conditions can be avoided by making them complement of each other. This modified SR latch is known as D latch.

Fig. 4.3.6 shows the D latch. The NAND gates 1,



(a) D latch

(b) Logic symbol

Fig. 4.3.6

2, 3 and 4 form the basic SR latch with enable input. The fifth NAND gate is used to provide the complemented inputs.

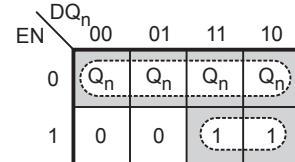
As shown in the Fig. 4.3.6, D input goes directly to the S input, and its complement is applied to the R input, through gate 5. Therefore, only two input conditions exists, either $S = 0$ and $R = 1$ or $S = 1$ and $R = 0$. The truth table for D latch is as shown in the Table 4.3.2.

As shown in the truth table, the Q output follows the D input. For this reason D latch is sometimes called **transparent latch**.

Looking at the truth table for D latch with enable input and simplifying Q_{n+1} function by k-map we get the characteristic equation for D latch with enable input as $Q_{n+1} = EN \cdot D + \bar{EN} \cdot Q_n$. This is illustrated in Fig. 4.3.7.

EN	D	Q_n	Q_{n+1}	State
1	0	X	0	Reset
1	1	X	1	Set
0	X	X	Q_n	No Change (NC)

Table 4.3.2 Truth table for D latch



$$Q_{n+1} = EN \cdot D + \bar{EN} \cdot Q_n$$

Fig. 4.3.7 Characteristic equation

Review Questions

1. What is SR latch ? Explain it's operation.
2. What is gated SR latch ?
3. Explain the working of gated D latch with truth table and characteristic equation.

4.4 Flip-Flops

SPPU : May-05,06,07,11,12,13, Dec.-05,06,10

4.4.1 Latches Vs Flip-Flops

Latches and flip-flops are the basic building blocks of the most sequential circuits. The main difference between latches and flip-flops is in the method used for changing their state.

A simple latch forms the basis for the flip-flop. Latches are controlled by enable signal, and they are level triggered, either positive level triggered or negative level triggered. The output state is free to change according to the S and R input values, when active level is maintained at the enable input. Flip-flops are different from latches. Flip-flops are pulse or clock edge triggered instead of level triggered.

4.4.2 Level and Edge Triggering

Level Triggering

In the level triggering, the output state is allowed to change according to input(s) when active level (either positive or negative) is maintained at the enable input. There are two types of level triggered latches :

- Positive level triggered :** The output of flip-flop responds to the input changes only when its enable input is 1 (HIGH).

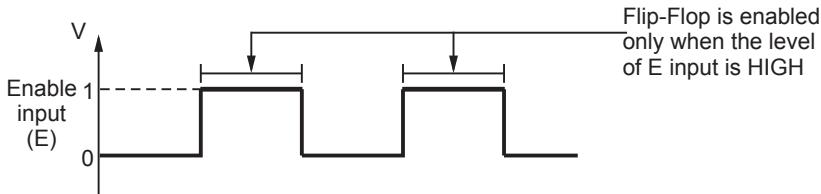


Fig. 4.4.1 Positive level triggering

- Negative level triggered :** The output of flip-flop responds to the input changes only when its enable input is 0 (LOW).

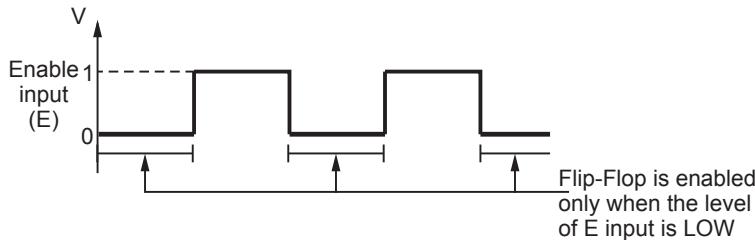


Fig. 4.4.2 Negative level triggering

Edge Triggering

In the edge triggering, the output responds to the changes in the input only at the positive or negative edge of the clock pulse at the clock input. There are two types of edge triggering.

- Positive edge triggering :** Here, the output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.

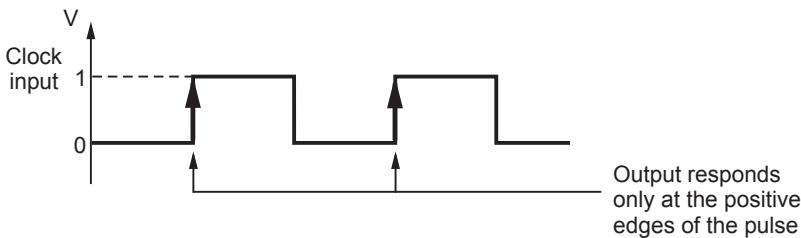


Fig. 4.4.3 Positive edge triggering

- Negative edge triggering :** Here, the output responds to the changes in the input only at the negative edge of the clock pulse at the clock input.

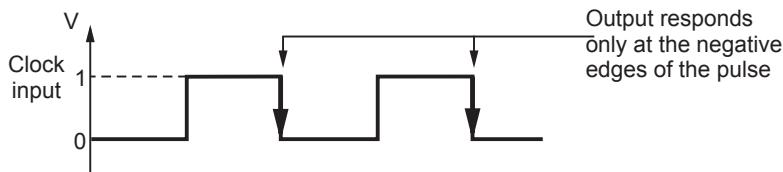


Fig. 4.4.4 Negative edge triggering

4.4.3 SR Flip-Flop

Positive Edge Triggered SR Flip-Flop

The Fig. 4.4.5 shows the positive edge triggered clocked SR flip-flop. The circuit is similar to SR latch except enable signal is replaced by the Clock Pulse (CP) followed by the positive edge detector circuit. The edge detector circuit is a differentiator. The Fig. 4.4.7 shows input and output waveforms for positive edge triggered clocked SR flip-flop. As shown in Fig. 4.4.7 the circuit output responds to the S and R inputs only at the positive edges of the clock pulse. At any other instants of time, the SR flip-flop will not respond to the changes in input.

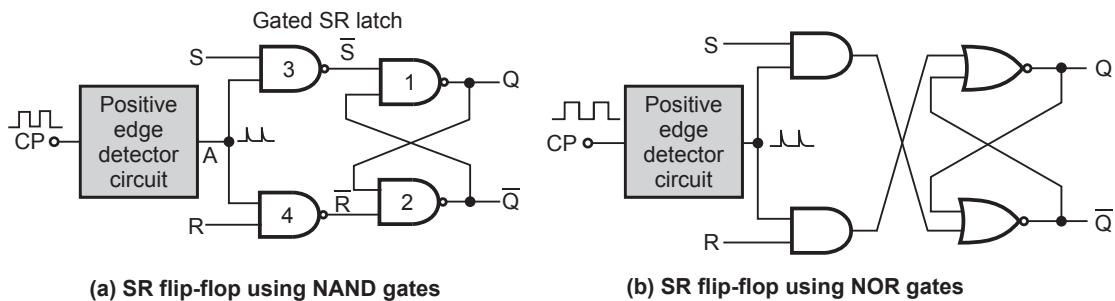
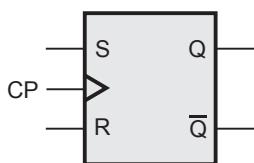


Fig. 4.4.5 Clocked SR flip-flop

The Fig. 4.4.6 shows the logic symbol and truth table of clocked SR flip-flop.



CP	S	R	Q_n	Q_{n+1}	State
\uparrow	0	0	0	0	No Change(NC)
\uparrow	0	0	1	1	
\uparrow	0	1	0	0	
\uparrow	0	1	1	0	Reset
\uparrow	1	0	0	1	
\uparrow	1	0	1	1	
\uparrow	1	1	0	X	
\uparrow	1	1	1	X	Set
0	X	X	0	0	
0	X	X	1	1	No Change(NC)

Q_n	0	1
SR	00	0
	01	0
	11	X
	10	1

$$Q_{n+1} = S + \bar{R} Q_n$$

(a) Logic symbol

(b) Truth table for positive edge clocked SR flip-flop

(c) Characteristic equation

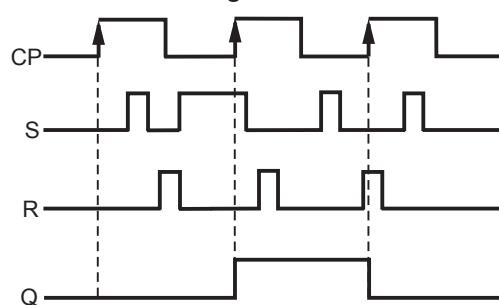
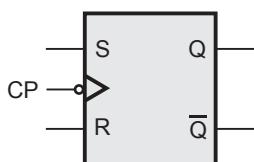


Fig. 4.4.7 Input and output waveforms for positive edge triggered clocked SR flip-flop

- Case 1 :** If $S = R = 0$ and the clock pulse is applied, the output do not change, i.e. $Q_{n+1} = Q_n$. This is indicated in the first row of the truth table.
- Case 2 :** If $S = 0, R = 1$ and the clock pulse is applied, $Q_{n+1} = 0$. This is indicated in the second row of the truth table.
- Case 3 :** If $S = 1, R = 0$ and the clock pulse is applied, $Q_{n+1} = 1$. This is indicated in the third row of the truth table.
- Case 4 :** If $S = R = 1$ and the clock pulse is applied, the state of the flip-flop is undefined and therefore is indicated as indeterminate in the fourth row of the truth table.

Negative Edge Triggered SR Flip-Flop

In the negative edge triggered SR flip-flop, the negative edge detector circuit is used and the circuit output responds at the negative edges of the clock pulse. The Fig. 4.4.8 and Fig. 4.4.9 shows the logic symbol, truth table, and input and output waveforms for negative edge triggered SR flip-flop. The bubble at the clock input indicates that the flip-flop is negative edge triggered.



(a) Logic symbol

CP	S	R	Q_n	Q_{n+1}	State
↓	0	0	0	0	No change(NC)
↓	0	0	1	1	
↓	0	1	0	0	
↓	0	1	1	0	Reset
↓	1	0	0	1	
↓	1	0	1	1	Set
↓	1	1	0	X	
↓	1	1	1	X	Indeterminate
0	X	X	0	0	No change(NC)
0	X	X	1	1	

(b) Truth Table for negative edge clocked SR flip-flop

Fig. 4.4.8

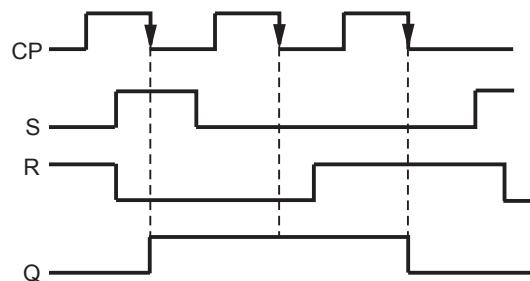


Fig. 4.4.9 Input and output waveforms for negative edge triggered clocked SR flip-flop

Example 4.4.1 Realize SR flip-flop using NOR gates.

Solution :

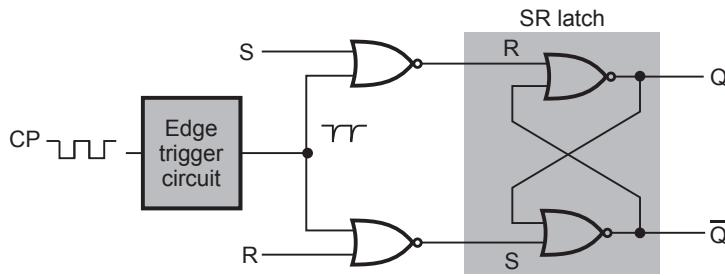
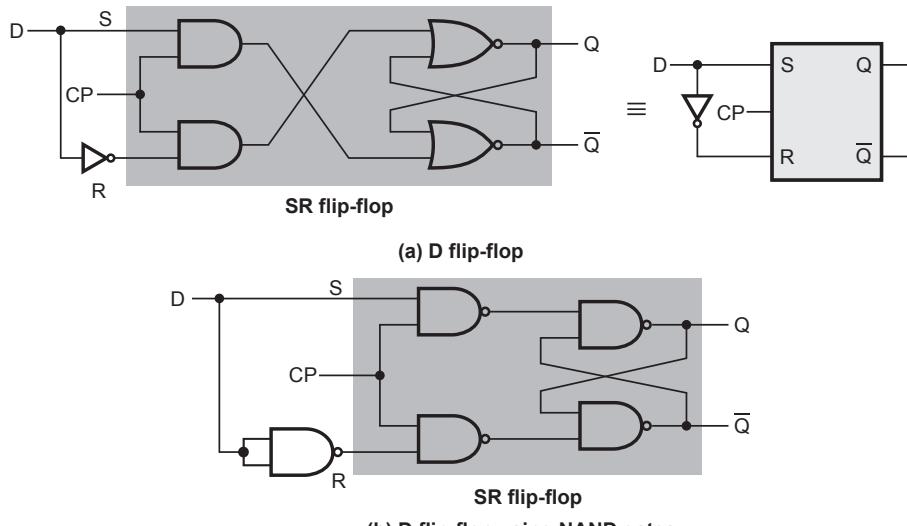


Fig. 4.4.10 Negative edge-triggered SR flip-flop using only NOR gates

4.4.4 D Flip-Flop

The Fig. 4.4.11 shows the logic diagrams of D flip-flop. The basic building block of D flip-flop is a SR flip-flop. The SR flip-flop has two data inputs S and R. The S input is made high to store 1 in the flip-flop and R input is made high to store 0 in the flip-flop.



(b) D flip-flop using NAND gates

Fig. 4.4.11

Looking at the truth table of the SR flip-flop we can realize that when both inputs are same the output either does not change or it is invalid (Inputs \rightarrow 00, no change and inputs \rightarrow 11, invalid). In many practical applications, these input conditions are not required. These input conditions can be avoided by making them complement of each other. This modified SR flip-flop is known as D flip-flop.

As shown in the Fig. 4.4.11, the D input goes directly to the S input, and its complement is applied to the R input. Due to these connections, only two input

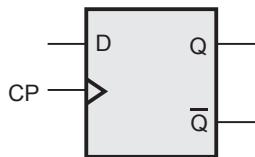


Fig. 4.4.12 (a) Logic symbol

CP	D	Q_{n+1}
\uparrow	0	0
\uparrow	1	1
0	X	Q_n

Fig. 4.4.12 (b) Truth table of D flip-flop

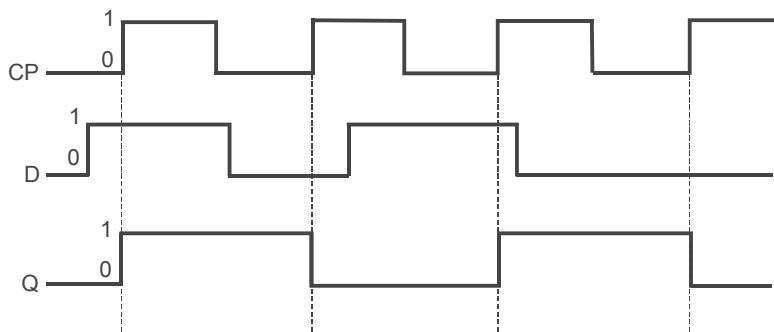


Fig. 4.4.12 (c) Input and output waveforms of clocked D flip-flop

conditions exists, either $S = 0$ and $R = 1$ or $S = 1$ and $R = 0$. The truth table for D flip-flop consider only these two conditions and it is as shown in the Fig. 4.4.12 (b).

Looking at the truth table for D flip-flop we can realize that Q_{n+1} function follows D input at the positive going edges of the clock pulses. Hence the characteristic equation for D flip-flop is $Q_{n+1} = D$. However, the output Q_{n+1} is delayed by one clock period. Thus, D flip-flop is also known as **delay flip-flop**.

If we connect the \bar{Q} output of D flip-flop to its D input as shown in the Fig. 4.4.13, the output of D flip-flop will change either from 0 to 1 or from 1 to 0 at every positive edge of the D flip-flop.

Such change in the output is known as **toggling** of the flip-flop output.

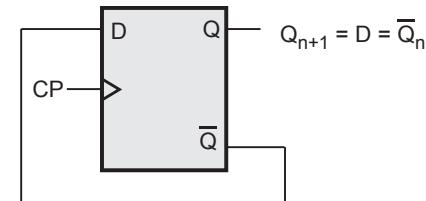


Fig. 4.4.13

Negative Edge Triggered D Flip-Flop

In the previous explanation we have seen the output of D flip-flop is sensitive at the positive edge of the clock input. In case of negative edge triggering, the output is sensitive at the negative edge of the clock input. The Fig. 4.4.14 shows the logic symbol and truth table for negative edge triggered D flip-flop and Fig. 4.4.15 shows input and output waveforms for negative edge triggered D flip-flop. The bubble at the clock input indicates that the flip-flop is negative edge triggered.

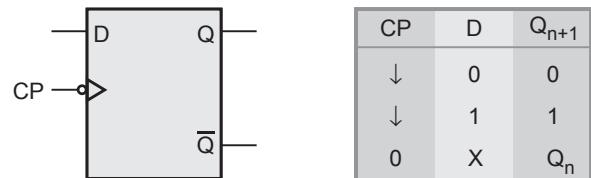


Fig. 4.4.14

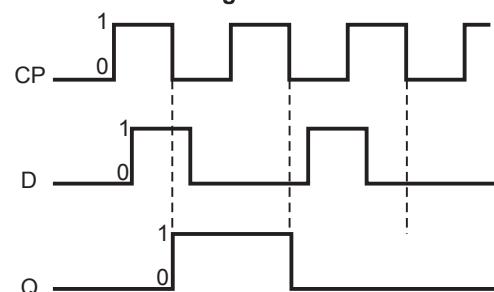


Fig. 4.4.15 Input output waveforms of negative edge triggered D flip-flop

4.4.5 JK Flip-Flop

The uncertainty in the state of an SR flip-flop when $S = R = 1$ can be eliminated by converting it into a JK flip-flop. The data inputs are J and K which are ANDed with Q and \bar{Q} , respectively, to obtain S and R inputs, as shown in the Fig. 4.4.16. Thus, $S = J \cdot \bar{Q}$ and $R = K \cdot Q$.

Let us see the operation of JK flip-flop.

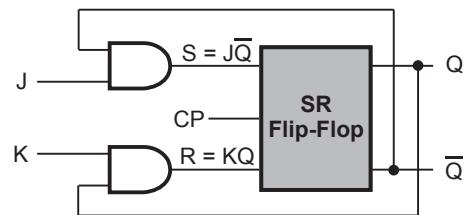


Fig. 4.4.16 JK flip-flop using SR flip-flop

Case 1 : $J = K = 0$

When $J = K = 0$, $S = R = 0$ and according to truth table of SR flip-flop there is no change in the output.

When inputs $J = K = 0$, output does not change.

Case 2 : $J = 1$ and $K = 0$

$Q = 0, \bar{Q} = 1$: When $J = 1, K = 0$ and $Q = 0$, $S = 1$ and $R = 0$. According to truth table of SR flip-flop it is set state and the output Q will be 1.

$Q = 1, \bar{Q} = 0$: When $J = 1, K = 0$ and $Q = 1$, $S = 0$ and $R = 0$. Since SR = 00, there is no change in the output and therefore, $Q = 1$ and $\bar{Q} = 0$.

The inputs $J = 1$ and $K = 0$, makes $Q = 1$, i.e. set state.

Case 3 : $J = 0$ and $K = 1$

$Q = 0, \bar{Q} = 1$: When $J = 0, K = 1$ and $Q = 0$, $S = 0$ and $R = 0$. Since SR = 00, there is no change in the output and therefore, $Q = 0$ and $\bar{Q} = 1$.

$Q = 1, \bar{Q} = 0$: When $J = 0, K = 1$ and $Q = 1$, $S = 0$ and $R = 1$. According to truth table of SR flip-flop it is a reset state and the output Q will be 0.

The inputs $J = 0$ and $K = 1$, makes $Q = 0$, i.e., reset state.

Case 4 : $J = K = 1$

$Q = 0, \bar{Q} = 1$: When $J = K = 1$ and $Q = 0$, $S = 1$ and $R = 0$. According to truth table of SR flip-flop it is a set state and the output Q will be 1.

$Q = 1, \bar{Q} = 0$: When $J = K = 1$ and $Q = 1$, $S = 0$ and $R = 1$. According to truth table of SR flip-flop it is a reset state and the output Q will be 0.

The input $J = K = 1$, toggles the flip-flop output.

The Fig. 4.4.17 shows the logic symbol, truth table and timing diagram of positive edge triggered JK flip-flop.

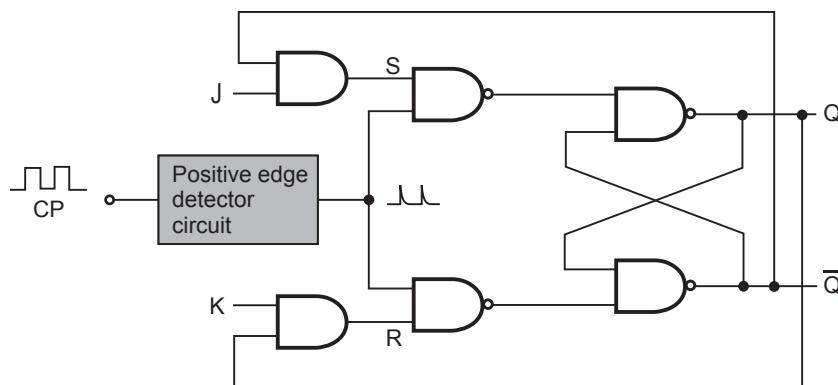
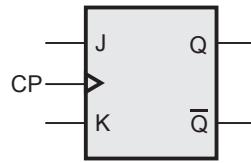


Fig. 4.4.17 (a) Clocked JK flip-flop



Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

Fig. 4.4.17 (b) Logic symbol

Fig. 4.4.17 (c) Truth table

Q_n	JK	00	01	11	10
0		0	0	1	1
1		1	0	0	1

$$Q_{n+1} = \bar{Q}_n J + Q_n \bar{K} = J \bar{Q}_n + \bar{K} Q_n$$

Fig. 4.4.17 (d) Characteristics equation

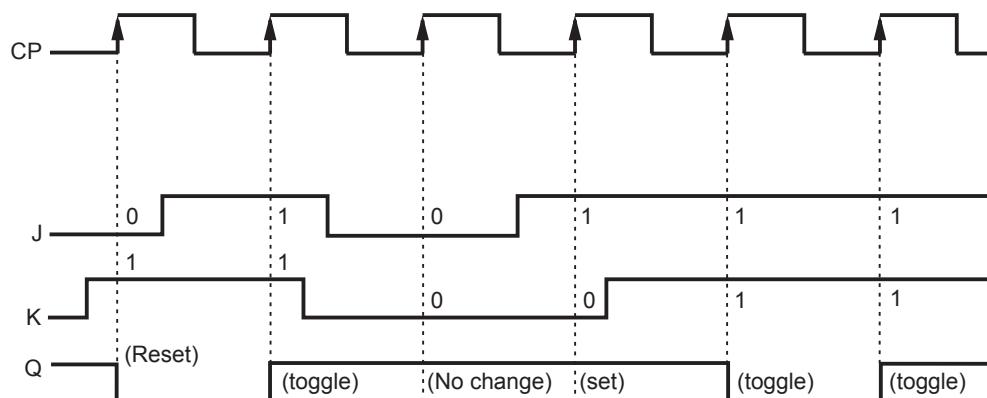


Fig. 4.4.18 Input and output waveforms for positive edge triggered JK flip-flop

Example 4.4.2 Construct a clocked JK flip-flop which is triggered at the positive edge of the clock pulse from a clocked SR flip-flop consisting of NOR gates.

Solution :

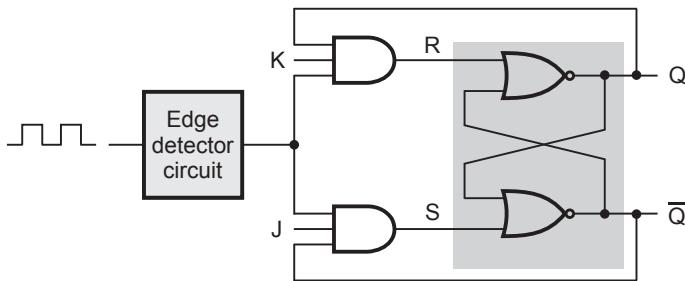


Fig. 4.4.19

4.4.5.1 JK Flip-Flop using NAND Gates

In the previous section we have seen the operation of JK flip-flop using SR flip-flop and AND gates. It is not necessary to use the AND gates of Fig. 4.4.17 (a), since the same function can be performed by adding an extra input terminal to NAND gates 3 and 4 of Fig. 4.4.20. The Fig. 4.4.20 shows the modified circuit of JK flip-flop which has only NAND gates.

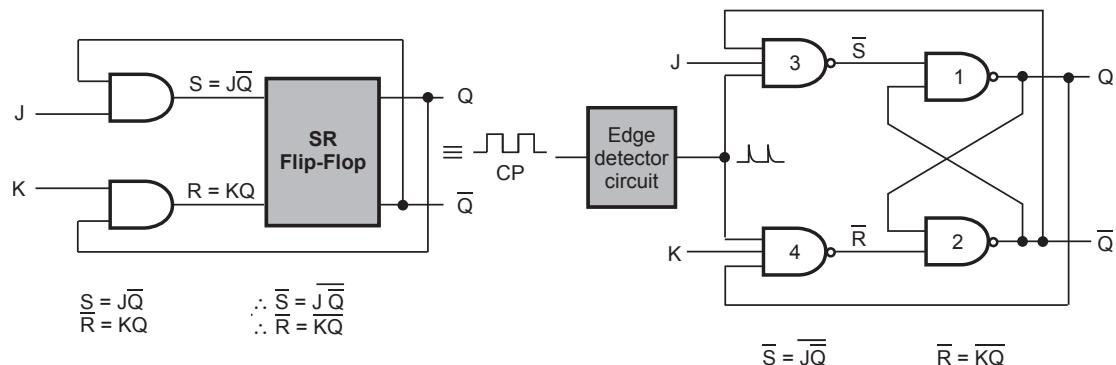


Fig. 4.4.20 JK flip-flop using NAND gates

4.4.5.2 Race-around Condition

In JK flip-flop, when $J = K = 1$, the output toggles (output changes either from 0 to 1 or from 1 to 0). Consider that initially $Q = 0$ and $J = K = 1$. After a time interval Δt equal to the propagation delay through two NAND gates in series, the output will change to $Q = 1$ and after another time interval of Δt the output will change back to $Q = 0$. This toggling will continue until the flip-flop is enabled and $J = K = 1$. At the end of clock pulse the flip-flop is disabled and the value of Q is uncertain. This situation is referred to as the **race-around condition**. This is illustrated in Fig. 4.4.21. This condition exists when $t_p \geq \Delta t$. Thus by keeping $t_p < \Delta t$ we can avoid race around condition.

We can keep $t_p < \Delta t$ by keeping the duration of edge less than Δt . A more practical method for overcoming this difficulty is the use of the Master-Slave (MS) configuration.

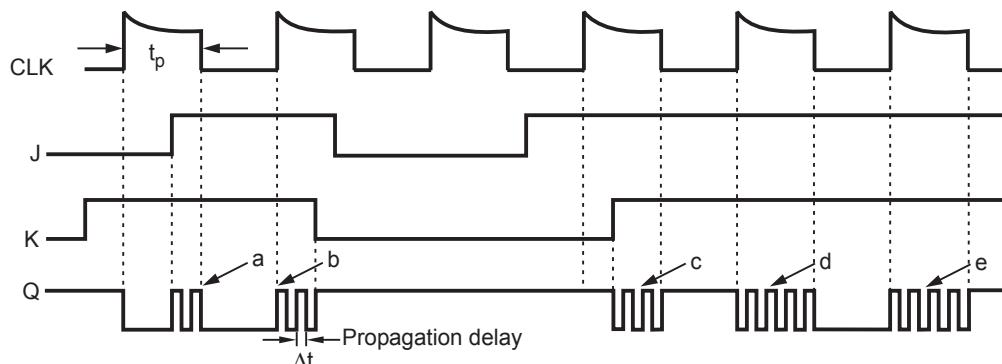


Fig. 4.4.21 Input and output waveforms for clocked JK flip-flop

Example 4.4.3 Realize a JK flip-flop using only NOR gates.

Solution :

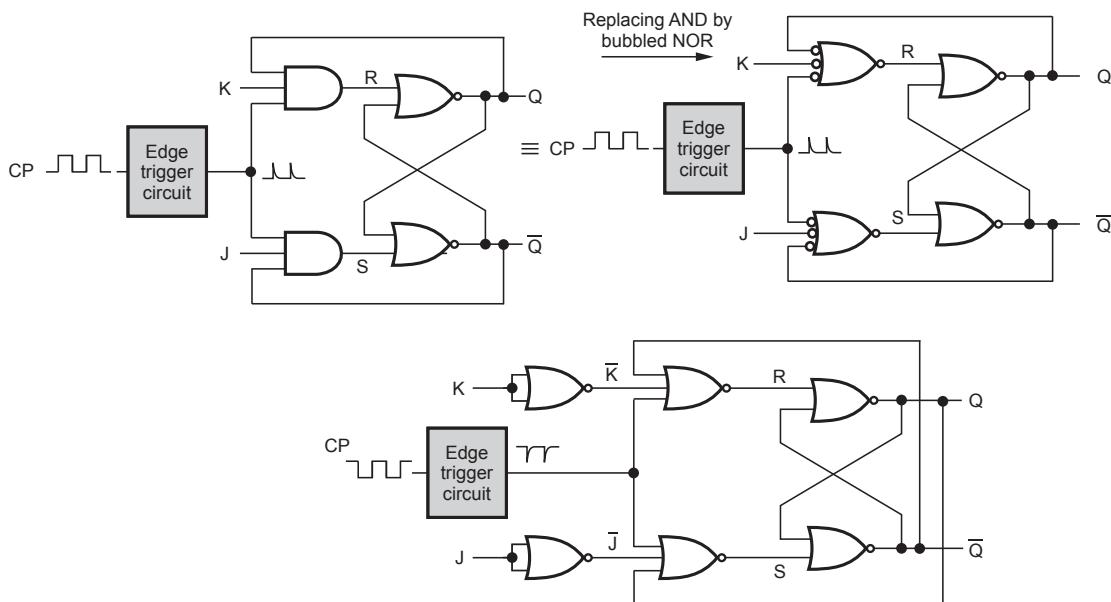


Fig. 4.4.22 Negative edge triggered JK flip-flop using only NOR gates

4.4.6 Master-Slave SR Flip-Flop

A master-slave flip-flop is constructed from two flip-flops. One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a CP master-slave flip-flop. Fig. 4.4.23 shows SR master-slave flip-flop. It

consists of a master flip-flop, a slave flip-flop, and an inverter. Both the flip-flops are

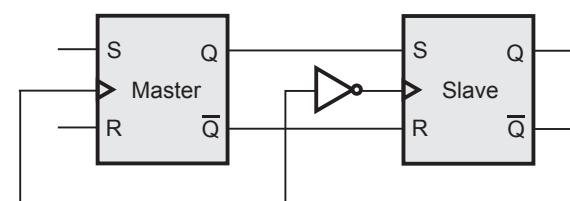


Fig. 4.4.23 Master-slave SR flip-flop

positive level triggered, but inverter connected at the clock input of the slave flip-flop forces it to trigger at the negative level.

The output state of the master flip-flop is determined by the S and R inputs at the positive clock pulse. The output state of the master is then transferred as an input to the slave flip-flop. The slave flip-flop uses this input at the negative clock pulse to determine its output state. The Fig. 4.4.24 illustrates the operation of the master-slave flip-flop.

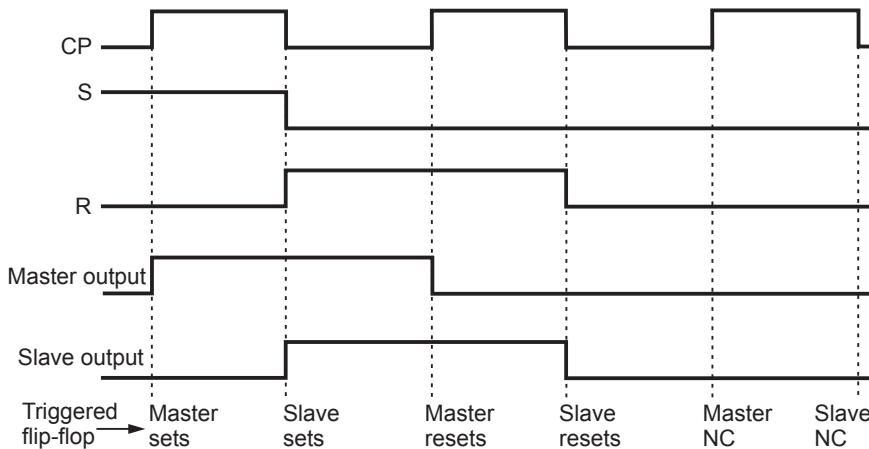


Fig. 4.4.24 Input and output waveforms for master-slave flip-flop

4.4.7 Master-Slave JK Flip-Flop

- Fig. 4.4.25 shows the master-slave JK flip-flop. Positive clock pulses are applied to first flip-flop and inverted (negative) clock pulses are applied to second flip-flop.
- When $CK = 1$, the first flip-flop is enabled and the outputs Q_M and \bar{Q}_M responds to the inputs of J and K according to the Table 1. At this time, the second flip-flop is inhibited because its clock is low, $\bar{CK} = 0$.

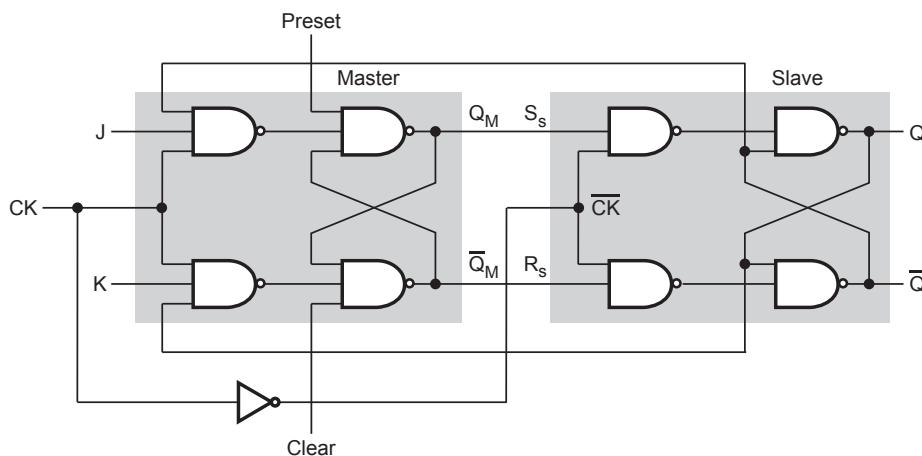


Fig. 4.4.25 Master - slave JK flip - flop

- When CK goes Low ($\overline{CK} = 1$), the first flip-flop is inhibited and second flip-flop is enabled. At this time, the output of second flip-flop (Q and \overline{Q}) follow the outputs Q_M and \overline{Q}_M , respectively.
- Since the second flip-flop follows the first one, it is referred to as the **slave** and the first one as the **master**.
- In master-slave JK flip-flop state change occurs when flip-flop goes through both positive transition (first half) of clock and negative transition of the clock (second half). Thus, race-around condition does not exist in the master-slave JK flip-flop.

4.4.8 T Flip-Flop

T flip-flop is also known as '**Toggle flip-flop**'. The T flip-flop is a modification of the JK flip-flop. As shown in the Fig. 4.4.26, the T flip-flop is obtained from a JK flip-flop by connecting both inputs, J and K together.

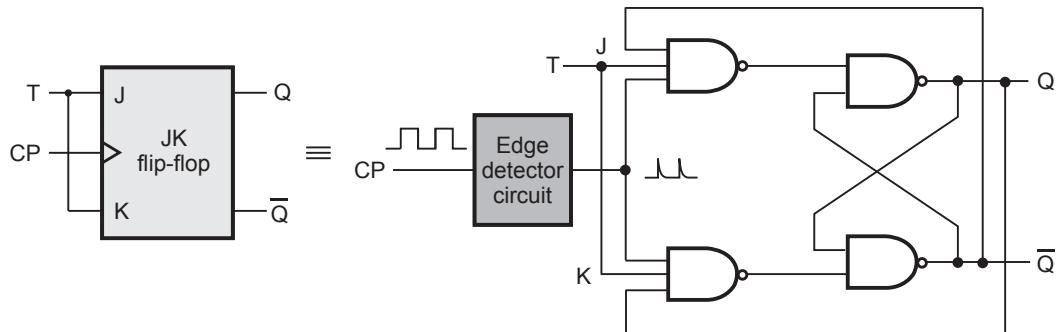


Fig. 4.4.26 T flip-flop using NAND gates

When $T = 0$, $J = K = 0$ and hence there is no change in the output. When $T = 1$, $J = K = 1$ and hence output toggles.

The Fig. 4.4.27 shows logic symbol, truth table and the characteristic equation for T flip-flop.

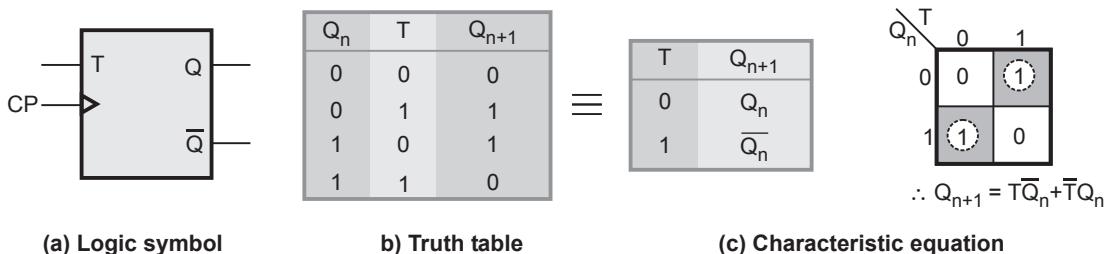


Fig. 4.4.27

Example 4.4.4 Refer Fig. 4.4.28 and determine the Q output waveform if the flip-flop starts out RESET.

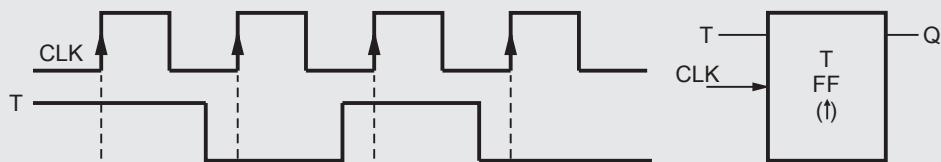


Fig. 4.4.28

Solution :

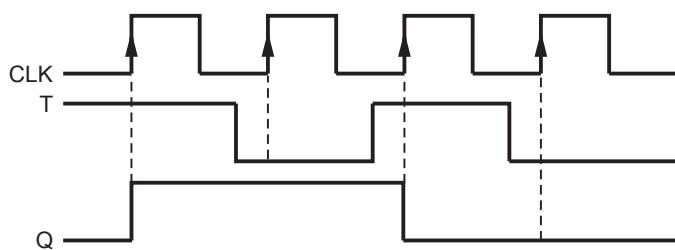


Fig. 4.4.29

4.4.9 Preset and Clear

For the flip-flops discussed so far, the SR, D, JK, and T, the inputs are called synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.

When power is turned ON, the state of the flip-flop is uncertain. It may come to set ($Q = 1$) or reset ($Q = 0$) state. In many applications, it is necessary to initially set or reset the flip-flop. Such initial state of flip-flop can be accomplished by using the direct or asynchronous inputs of the flip-flop. These inputs are : Preset (\bar{P}) and Clear (\bar{C}). They can be applied at any time between clock pulses and are not in synchronism with the clock.

The Fig. 4.4.30 shows the SR and D flip-flops with preset and clear inputs. These are active-low inputs and thus when $\bar{P} = \bar{C} = 1$, the circuit operates in accordance with the truth table of SR flip-flop. If $\bar{P} = 1$ and $\bar{C} = 0$, the flip-flop is reset and $\bar{P} = 0$ and $\bar{C} = 1$, the flip-flop is set.

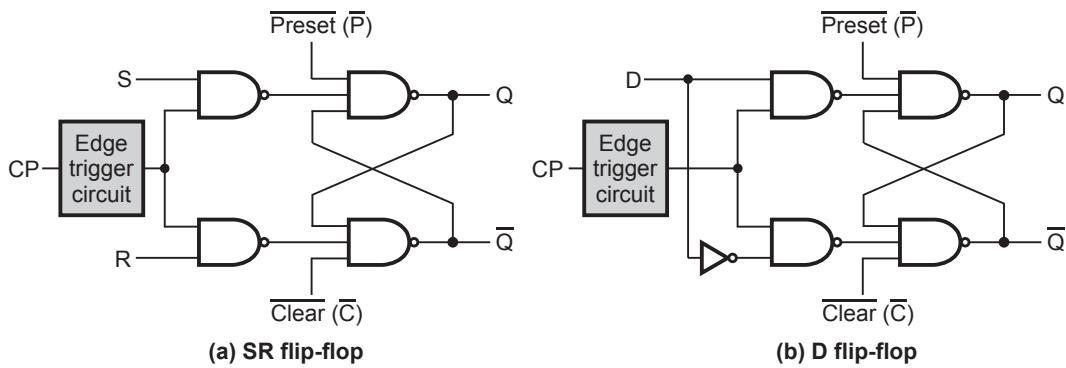
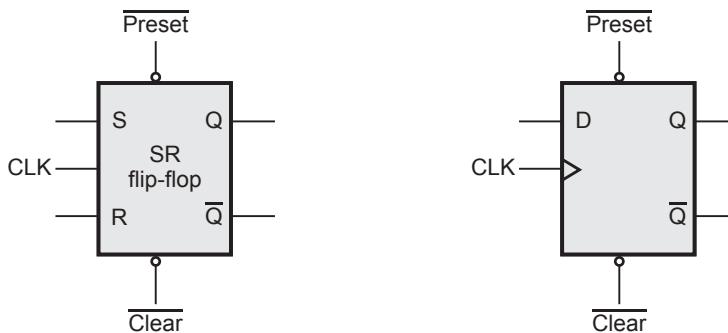


Fig. 4.4.30



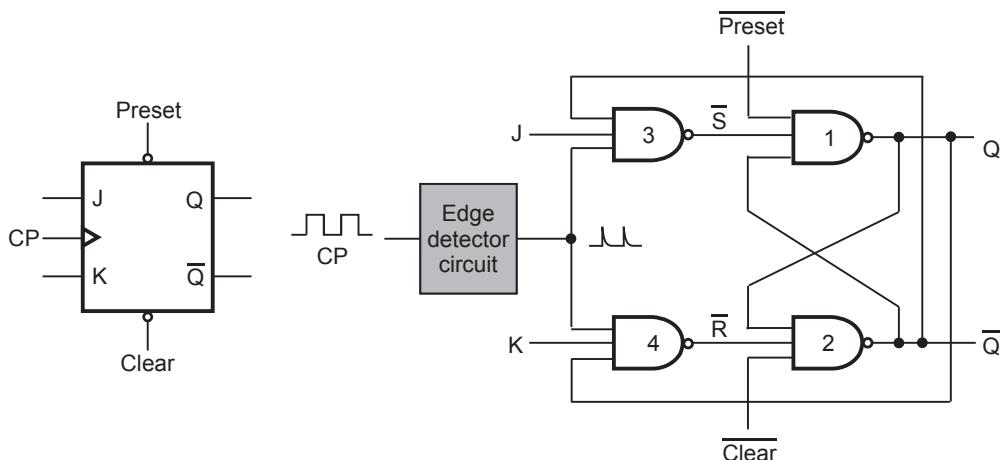
(a) Logic symbol for SR flip-flop

(b) Logic symbol for D flip-flop

Fig. 4.4.31 Logic symbol

Note Condition $\bar{P} = \bar{C} = 0$ must not be used, since this leads to an uncertain state.

The JK flip-flop with preset and clear is shown in Fig. 4.4.32. If preset and clear inputs are 1, the circuit operates in accordance with the truth table of JK flip-flop given



(a) Logic symbol

(b) JK flip-flop with active high preset and clear
Fig. 4.4.32

in the Fig. 4.4.17 (c). If preset = 0 and clear = 1, the output of NAND gate 1 will certainly be 1. Consequently, all the three inputs to NAND gate 2 will be 1 which will make $\bar{Q} = 0$. Hence making preset = 0 sets the flip-flop. Here, preset signal is active when it is low, hence it is active low signal. Similarly, low (0) on the clear input resets the flip-flop making $\bar{Q} = 1$.

A logic symbol for a JK flip-flop with active low preset and clear inputs is shown in the Fig. 4.4.32 (a). These inputs are active low, therefore they must both be kept high for synchronous operation. The Fig. 4.4.33 illustrates the operation of active low preset and clear inputs.

During clock pulse 1 and 2 the Preset is low, keeping the flip-flop set regardless of the synchronous inputs. For clock pulses 3, 4, 5 and 6, toggle operation occurs because J and K both are high with preset and clear inactive. For clock pulse 7, the Clear input is low, keeping the flip-flop reset regardless of the synchronous inputs.

Assume $J = K = 1$

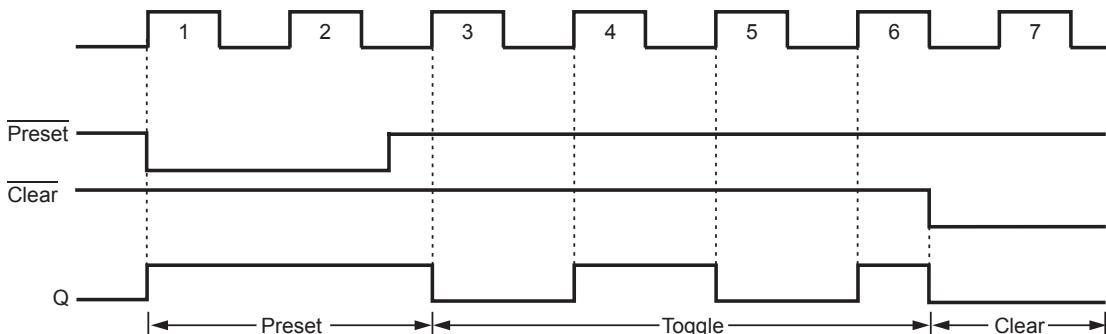


Fig. 4.4.33 Input and output waveforms showing operation of active low preset and clear inputs

In some IC packages preset and clear inputs are active high. In those cases preset and clear input must both be kept low for synchronous operation.

Review Questions

1. Differentiate between latches and flip-flops.
2. Explain the different types of triggering methods used for flip-flops.
3. What is SR flip-flop.
4. What is S-R flip-flop ? Explain working of clocked SR flip-flop. What is edge triggering ?
5. Explain the clocked SR flip-flop using NAND gates.
6. Explain the D flip-flop.
7. If Q output of a D-type flip-flop is connected to D input, it acts as a toggle switch. State whether true or false ? Justify your answer.

SPPU : May-12, Marks 4

SPPU : May-11, Marks 10

8. Justify name Delay for D flip-flop.
9. Explain the application of D flip-flop.

SPPU : May-06, Marks 2

10. Draw and explain the operation of JK flip-flop using logic gates with waveforms.
11. Draw the JK flip-flop using NAND gates.
12. What is Race-around condition ? Explain with the help of timing diagram. How is it removed in basic flip-flop circuit ?

SPPU : May-05,07, Dec.-05, Marks 8

13. Discuss method to avoid race around condition in JK flip-flop.

SPPU : Dec.-05, May-07, Marks 4

14. What do you mean by Master-Slave JK Flip-flop ? Explain the advantage of this Flip-flop. Draw suitable circuit diagram and timing diagram.

SPPU : Dec.-06, Marks 10

15. What is the advantage of M-S flip-flop ? Explain working of MS J-K flip-flop in detail.

SPPU : Dec.-10, Marks 8

16. Justify name toggle for T flip-flop giving truth table and waveforms.

17. Draw and explain the operation of T flip-flop.

18. Explain the application of T flip-flop.

SPPU : May-06, Marks 2

19. Explain the use of reset and preset inputs.

4.5 Excitation Tables

SPPU : Dec.-13, May-15

During the design process we know, from the transition table, the sequence of states, i.e., the transition from each present state to its corresponding next state. From this information we wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need a table that lists the required inputs for a given change of state. Such a table is known as an excitation table of the flip-flop.

We can derive the excitation tables for flip-flops from their truth tables. The excitation table consists of two columns Q_n and Q_{n+1} , and a column for each input to show how the required transition can be achieved. Let us see the truth tables and excitation tables for RS, JK, D and T flip-flops.

4.5.1 SR Flip-Flop

Table 4.5.1 (a) and (b) show the truth table and excitation tables for SR flip-flop, respectively. As shown in the table, there are four possible transitions from the present state to the next state. For each transition, the required input

S	R	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	*

(a) SR truth table

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(b) SR excitation table

Table 4.5.1

condition is derived from the information available in the truth table. Let us see the process by examining each case.

Note The symbol "X" in the table represents a don't care condition, i.e., it indicates that to get required output it does not matter whether the input is either 1 or 0.

0 → 0 Transition : The present state of the flip-flop is 0 and is to remain 0 when a clock pulse is applied. Looking at truth table of SR flip-flop we can understand that, this can happen either when $R = S = 0$ (no-change condition) or when $R = 1$ and $S = 0$. Thus, S has to be at 0, but R can be at either level. The table indicates this with a "0" under S and an "X" (don't care) under R.

0 → 1 Transition : The present state is 0 and is to change to 1. This can happen only when $S = 1$ and $R = 0$ (set condition). Therefore, S has to be 1 and R has to be 0 for this transition to occur.

1 → 0 Transition : The present state is 1 and is to change to a 0. This can happen only when $S = 0$ and $R = 1$ (reset condition). Therefore, S has to be 0 and R has to be 1 for this transition to occur.

1 → 1 Transition : The present state is 1 and is to remain 1. This can happen either when $S = 1$ and $R = 0$ (set condition) or when $S = 0$ and $R = 0$ (no change condition). Thus R has to be 0, but S can be at either level. The table indicates this with a "X" under S and "0" under R.

4.5.2 JK Flip-Flop

The truth table and excitation table for JK flip-flop are shown in Table 4.5.2 (a) and (b) respectively. Let us examine each case.

0 → 0 Transition : When both present state and next state are 0, the J input must remain at 0 and the K input can be either 0 and 1.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

(a) JK truth table

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(b) JK excitation table

Table 4.5.2

0 → 1 Transition : The present state is 0 and is to change to 1. This can happen either when $J = 1$ and $K = 0$ (set condition) or when $J = K = 1$ (toggle condition). Thus, J has to be 1, but K can be at either level for this transition to occur.

1 → 0 Transition : The present state is 1 and is to change to 0. This can happen either when $J = 0$ and $K = 1$ or when $J = K = 1$. Thus, K has to be 1 but J can be at either level.

1 → 1 Transition : When both present state and next are 1, the K input must remain at 0 while the J input can be 0 or 1.

As seen from Table 4.5.2, the excitation table for JK flip-flop has more don't care conditions than the excitation table for RS flip-flop. The don't care terms usually simplify the function. Therefore, the combinational circuits using JK flip-flops for the input functions are likely to be simpler than those using RS flip-flops.

4.5.3 D Flip-Flop

The Table 4.5.3 (a) and (b) show the truth table and excitation table for D flip-flop, respectively. In D flip-flop, the next state is always equal to the D input and it is independent of the present state. Therefore, D must be 0 if Q_{n+1} has to be 0, and 1 if Q_{n+1} has to be 1, regardless of the value of Q_n .

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

(a) D truth table (b) D excitation table

Table 4.5.3

4.5.4 T Flip-Flop

The Table 4.5.4 (a) and (b) show the truth table and the excitation table for T flip-flop, respectively. We know that when input $T = 1$, the state of the flip-flop is complemented; when $T = 0$, the state of the flip-flop remains unchanged. Therefore, for $0 \rightarrow 0$ and $1 \rightarrow 1$ transitions T must be 0 and for $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions T must be 1.

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.5.4

Review Questions

- Derive the excitation tables for SR, D, JK and T flip-flop.
- Draw the excitation table of S-R flip-flop.
- Draw the excitation table of J-K flip-flop.

SPPU : Dec.-13, Marks 2

SPPU : May-15, Marks 2

4.6 Conversion from One Type to Another Type of Flip-Flop

SPPU : May-06,10,12, Dec.-16,17,19

It is possible to convert one flip-flop into another flip-flop with some additional gates or simply doing some extra connection. Let us see few conversions among flip-flops.

4.6.1 SR Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in Table 4.6.1.

Input	Present state	Next state	Flip-flop inputs	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Table 4.6.1

K-map simplification

Logic diagram

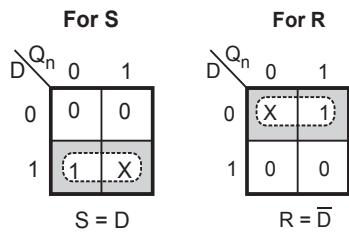


Fig. 4.6.1

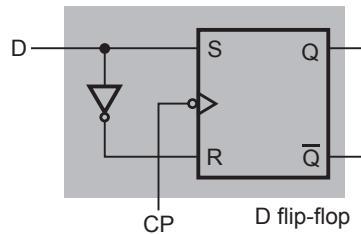


Fig. 4.6.2 SR to D flip-flop conversion

4.6.2 SR Flip-Flop to JK Flip-Flop**SPPU : May-10**

The excitation table for above conversion is as shown in Table 4.6.2.

Inputs		Present state	Next state	Flip-flop inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Table 4.6.2

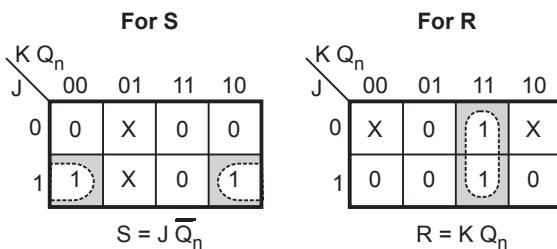
K-map simplification

Fig. 4.6.3

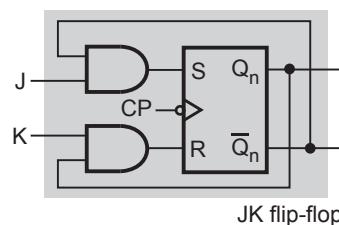
Logic diagram

Fig. 4.6.4 SR to JK flip-flop conversion

4.6.3 SR Flip-Flop to T Flip-Flop

The excitation table for above conversion is as shown in the Table 4.6.3.

Input	Present state		Next state		Flip-flop inputs	
	T	Q_n	Q_{n+1}	S	R	
0	0	0	0	0	X	
0	1	1	1	X	0	
1	0	0	1	1	0	
1	1	1	0	0	1	

Table 4.6.3

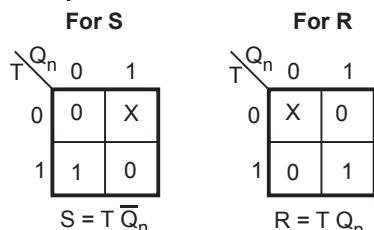
K-map simplification

Fig. 4.6.5

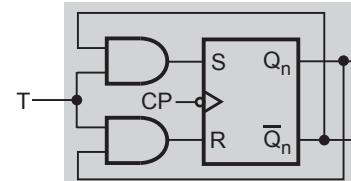
Logic diagram

Fig. 4.6.6 SR to T flip-flop conversion

If we apply clock pulses to the circuit, the circuit output will toggle from 0 to 1 and 1 to 0. Thus, we can build 1-bit counter using SR flip-flop by converting it to T flip-flop.

Example 4.6.1 Prepare the truth table

for the circuit of Fig. 4.6.7 and show that it acts as a T-type flip-flop.

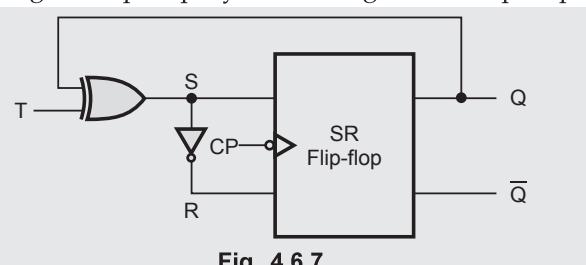


Fig. 4.6.7

Solution : For SR flip-flop,

$$\begin{aligned}
 Q_{n+1} &= S + \bar{R} Q_n \\
 &= S + S Q_n \\
 &= S(1 + Q_n) = S
 \end{aligned}
 \quad \dots \text{Characteristics equation}$$

$\therefore R = \bar{S}$

We have, $S = Q_n \oplus T$

$$\begin{aligned}
 \therefore Q_{n+1} &= Q_n \oplus T \\
 &= T \bar{Q}_n + \bar{T} Q_n
 \end{aligned}
 \quad \dots \text{Characteristic equation of T flip-flop}$$

C _p	T	Q _n	S = Q _n ⊕ T	R = \bar{S}	Q _{n+1} = S
↓	0	0	0	1	0
↓	0	1	1	0	1
↓	1	0	1	0	1
↓	1	1	0	1	0

Table 4.6.4 Truth table for the given circuit

Looking at column 1 and column 5 of the Table 4.6.4 we can conclude that when $T = 0$, the output does not change and when $T = 1$, the output toggles. Thus, the given circuit acts as a T flip-flop. This is another way of implementing T flip-flop using SR flip-flop.

4.6.4 JK Flip-Flop to T Flip-Flop

The excitation table for above conversion is as shown in Table 4.6.5.

Input	Present state	Next state	Flip-flop inputs	
T	Q _n	Q _{n+1}	J _A	K _A
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

Table 4.6.5

K-map simplification

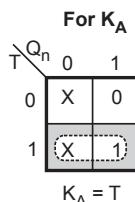
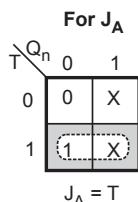


Fig. 4.6.8

Logic diagram

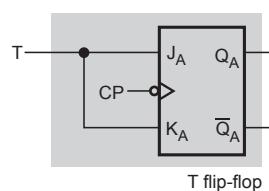


Fig. 4.6.9 JK to T flip-flop conversion

4.6.5 JK Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in the Table 4.6.6.

Input	Present state	Next state	Flip-flop inputs	
D	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

Table 4.6.6

K - map simplification

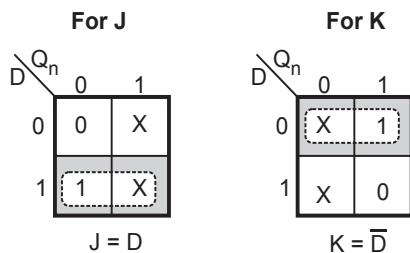


Fig. 4.6.10

Logic diagram

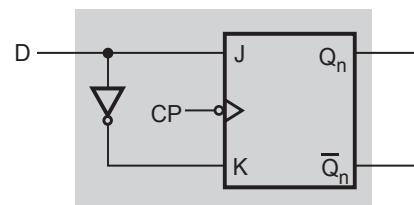


Fig. 4.6.11 JK to D flip-flop conversion

4.6.6 D Flip-Flop to T Flip-Flop

The excitation table for above conversion is as shown in the Table 4.6.7.

Input	Present state	Next state	Flip-flop input
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table 4.6.7

K - map simplification

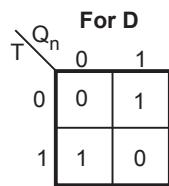


Fig. 4.6.12

Logic diagram

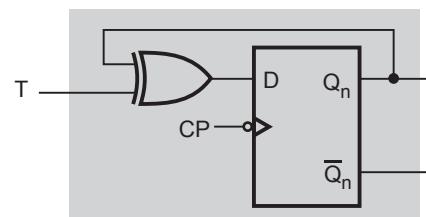


Fig. 4.6.13 D to T flip-flop conversion

$$D = \bar{T}Q_n + T\bar{Q}_n = T \oplus Q_n$$

Example 4.6.2 Analyze the circuit and prove that it is equivalent to T flip-flop.

Solution : To analyze the circuit means to derive the truth table for it.

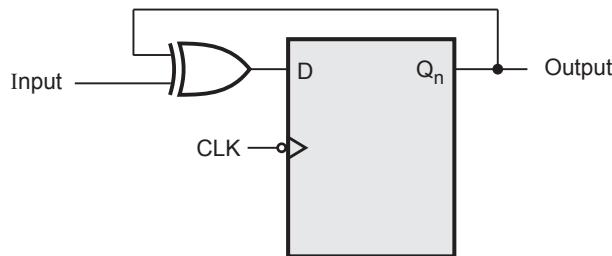


Fig. 4.6.14

$$\text{We have, } D = \text{Input} \oplus Q_n$$

CLK	Input	Q_n	$D = \text{Input} \oplus Q_n$	Q_{n+1}
↓	0	0	0	0
↓	0	1	1	1
↓	1	0	1	1
↓	1	1	0	0

When input is 0
 } output does not change
 When input is 1
 } output toggles

Table 4.6.8 Truth table for given circuit

In the above circuit, output does not change when input is 0 and it toggles when input is 1. This is the characteristics of T flip-flop. Hence, the given circuit is T flip-flop constructed using D flip-flop.

4.6.7 T Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in the Table 4.6.9.

Input	Present state	Next state	Flip-flop input
D	Q_n	Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 4.6.9

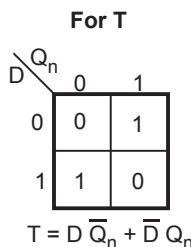
K-map simplification

Fig. 4.6.15

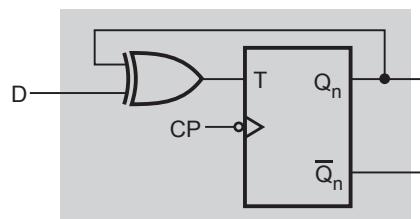
Logic diagram

Fig. 4.6.16 T to D flip-flop conversion

4.6.8 JK Flip-Flop to SR Flip-Flop

The excitation table for above conversion is as shown in Table 4.6.10.

Inputs	Present state	Next state	Flip-flop inputs	
S R	Q _n	Q _{n+1}	J	K
0 0	0	0	0	X
0 0	1	1	X	0
0 1	0	0	0	X
0 1	1	0	X	1
1 0	0	1	1	X
1 0	1	1	X	0
1 1	0	X	X	X
1 1	1	X	X	X

Table 4.6.10 Excitation table for JK to SR conversion

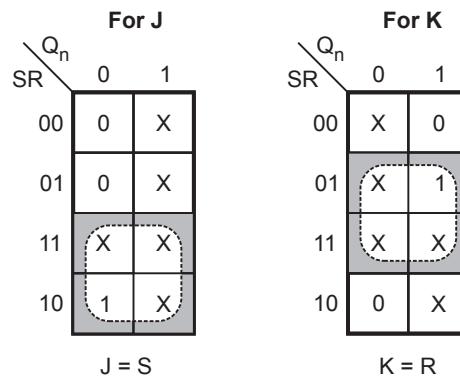
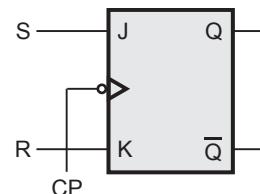
K-map simplification**Logic diagram**

Fig. 4.6.18 JK to SR

4.6.9 D Flip-Flop to SR Flip-Flop

The excitation table for above conversion is as shown in the Table 4.6.11.

Inputs	Present state	Next state	Flip-flop input
S R	Q _n	Q _{n+1}	D
0 0	0	0	0
0 0	1	1	1

0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	X	X
1	1	1	X	X

Table 4.6.11 Excitation table for D to SR conversion

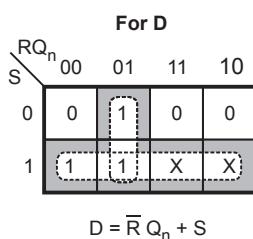
K - map simplification

Fig. 4.6.19

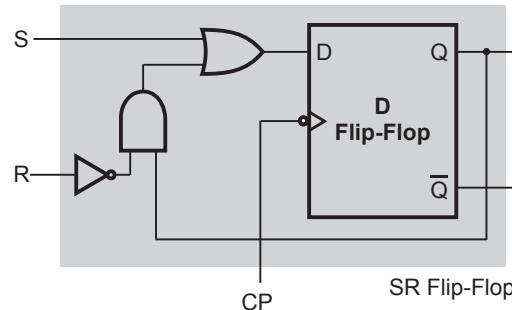
Logic diagram

Fig. 4.6.20 D to SR flip-flop conversion

4.6.10 T Flip-Flop to SR Flip-Flop

The excitation table for conversion of T FF into an SR FF is as shown in the Table 4.6.12.

Inputs	Present state		Next state		Flip-flop input
	S	R	Q _n	Q _{n+1}	
0 0	0		0	0	0
0 0		1	1	1	0
0 1	0		0	0	0
0 1		1	1	0	1
1 0	0		1	1	1
1 0		1	1	1	0
1 1	0		X	X	x
1 1		1	X	X	x

Table 4.6.12 Excitation table for T to SR conversion

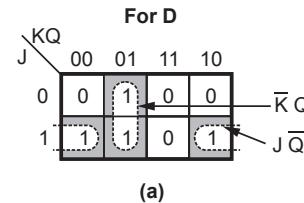
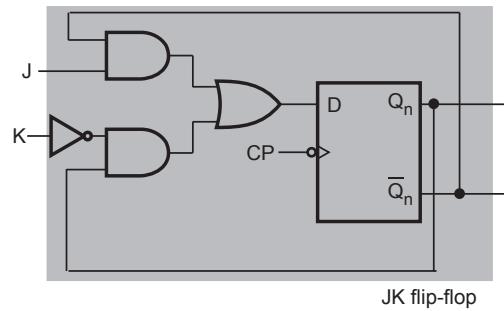
K - map simplification**Logic diagram**

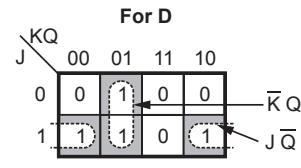
Fig. 4.6.21

4.6.11 D Flip-Flop to JK Flip-Flop

The excitation table for conversion of D flip-flop to JK flip-flop :

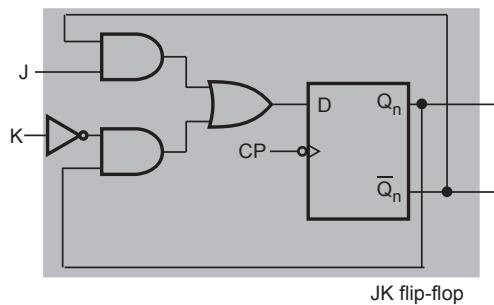
Inputs		Present state	Next state	Flip-flop input
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K - map simplification



(a)

Logic diagram



(b)

Fig. 4.6.22

Review Questions

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

SPPU : May-12, Dec.-16 Marks 2

SPPU : May-10, 12, Marks 4

SPPU : May-12, Marks 2

SPPU : May-06, Dec.-16, 19, Marks 2

SPPU : May-06, Marks 3

SPPU : Dec.-17, Marks 2

4.7 Applications of Flip-Flops

Some of the important applications of flip-flops are :

- It can be used as a memory element.
- It can be used to eliminate key debounce.

- It is used as a basic building block in sequential circuits such as counters and registers.
- It can be used as a delay element.

4.7.1 Bounce Elimination Switch

For interfacing keys to the digital systems, usually push button keys are used. These push button keys when pressed bounces a few times, closing and opening the contacts before providing a steady reading, as shown in the Fig. 4.7.1. Reading taken during bouncing period may be faulty. This problem is known as **key debounce**. The problem of key debounce is undesirable and it must be avoided.

One way to avoid key debounce problem is to use SR latch. The circuit used to avoid keybounce with SR latch is called a **switch or contact debouncer**. The Fig. 4.7.2 shows the switch debouncer circuit and its waveforms. When key is at position A, the output of SR latch is logic 1, and when key is at position B, the output of SR latch is logic 0. It is important to note that, when

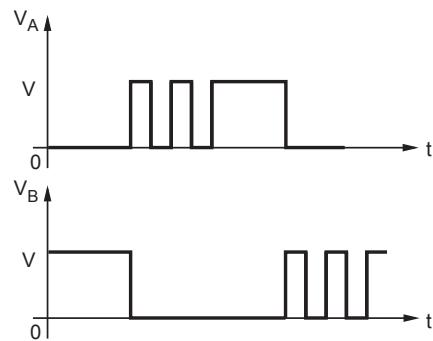
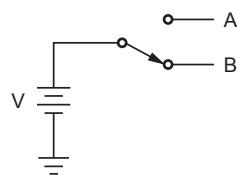


Fig. 4.7.1 Effect of key debounce

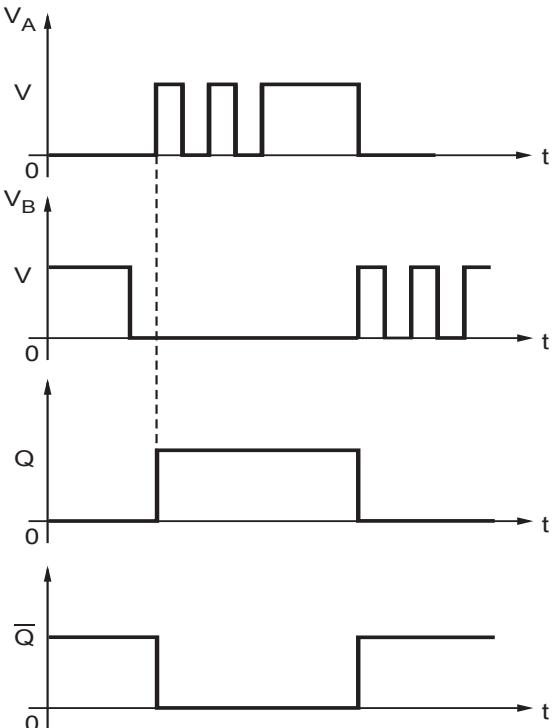


Fig. 4.7.2 Waveforms of switch debouncer

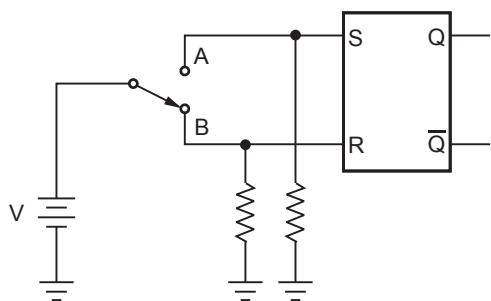


Fig. 4.7.2 (a) Switch debouncer

key is in between A and B, SR inputs are 00 and hence output does not change, preventing debouncing of key output. In other words, we can say that the output does not change during transition period, eliminating key debounce.

4.7.2 Registers

A group of flip-flops connected together forms a **register**. A register is used solely for storing and shifting data which is in the form of 1s and/or 0s, entered from an external source. Fig. 4.7.3 shows 3-bit register using three D flip-flops. Preset, reset and clock inputs are connected in parallel and 3-bit input is applied to D inputs of the flip-flop. During positive edge of the clock, D inputs of the flip-flops are stored within the flip-flop and are available at the outputs of D flip-flops.

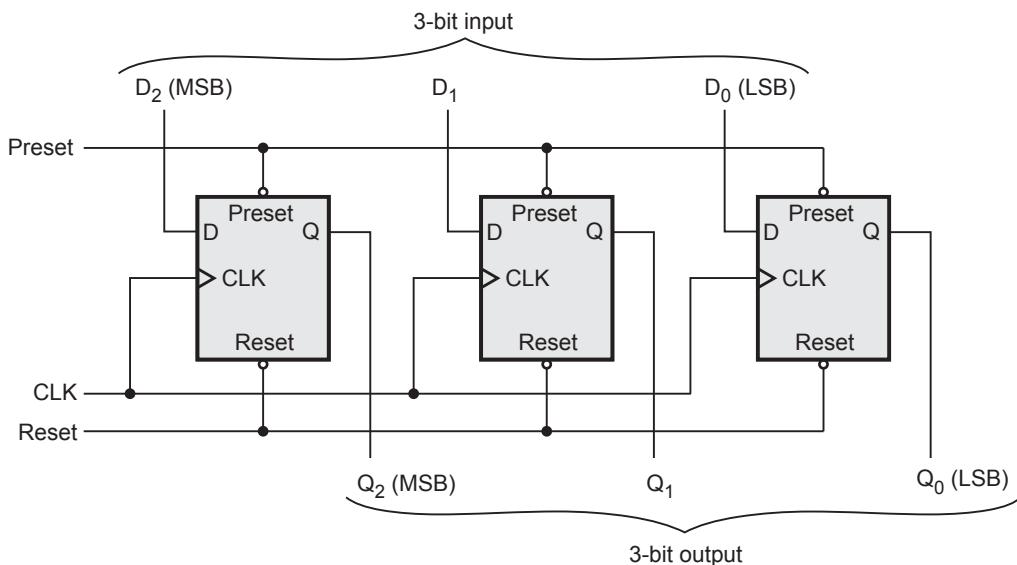


Fig. 4.7.3 Register

4.7.3 Counters

A **counter** is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. On arrival of each clock pulse, the counter is incremented by one. In case of down counter, it is decremented by one.

Fig. 4.7.4 shows 3-bit counter. It consists of three flip-flops. A counter with n flip-flops has 2^n possible states. Therefore, the 3-bit counter can count from decimal 0 to 7.

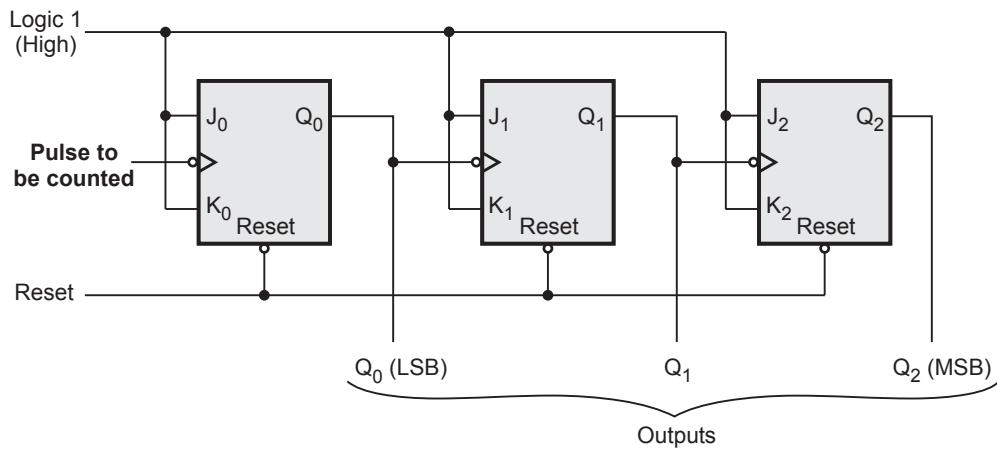


Fig. 4.7.4 Counter

Review Questions

1. Lists the applications of flip-flops.
2. How keyboard debouncing is eliminated using flip-flop ? Explain with suitable circuit diagram.



Notes

UNIT - III

5

Registers

Syllabus

SISO, SIPO, PISO, PIPO, Shift Registers, Bidirectional shift Register, Universal shift Register

Contents

5.1	<i>Introduction</i>	
5.2	<i>Shift Registers</i>	
5.3	<i>Types of Shift Registers</i>	<i>May-05,06,08,10,14, Dec.-05,08,11,12,14 , . . . Marks 10</i>
5.4	<i>Universal Shift Register</i>	<i>Dec.-06,07, May-07 , . . . Marks 10</i>
5.5	<i>Applications of Shift Registers</i>	<i>May-10, Marks 4</i>

5.1 Introduction

We have seen that a flip-flop is nothing but a binary cell capable of storing one bit information, and can be connected together to perform counting operations. Such a group of flip-flops is called **counter**. We have also seen that group of flip-flops can be used to store a word, which is called **register**.

A flip-flop can store 1-bit information. So an n-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

5.1.1 Buffer Register

Fig. 5.1.1 shows the simplest register

constructed with four D flip-flops. This register is also called **buffer register**. Each D flip-flop is triggered with a common negative edge clock pulse.

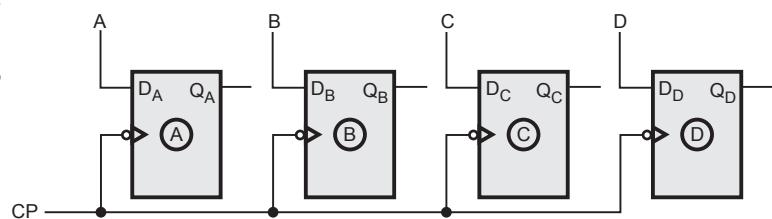


Fig. 5.1.1 Buffer register

The input bits set up the flip-flops for loading. Therefore, when the first negative clock edge arrives, the stored binary information becomes,

$$Q_A Q_B Q_C Q_D = ABCD$$

In this register, four D flip-flops are used. So it can store 4-bit binary information. Thus the number of flip-flop stages in a register determines its total storage capacity.

5.1.2 Controlled Buffer Register

We can control input and output of the register by connecting tri-state devices at the input and output sides of register as shown in Fig. 5.1.2. So this register is called '**controlled buffer register**'.

Here, tri-state switches are used to control the operation. When you want to store data in the register,

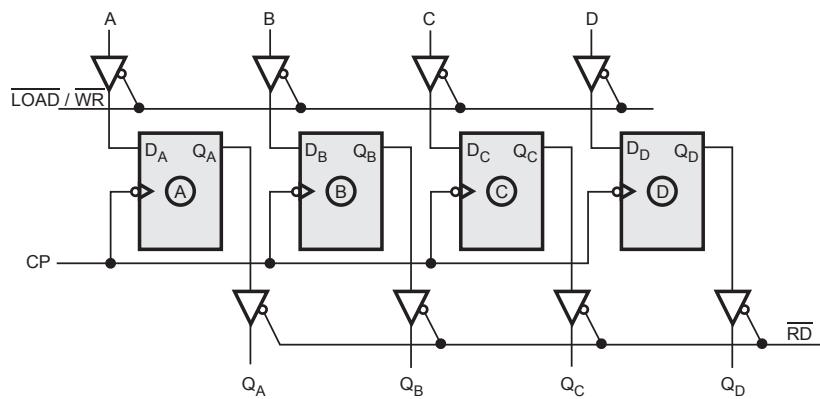


Fig. 5.1.2 Controlled buffer register

you have to make $\overline{\text{LOAD}}$ or $\overline{\text{WR}}$ signal low to activate the tri-state buffers. When you want the data at the output, you have to make $\overline{\text{RD}}$ signal low to activate the buffers. Controlled buffer registers are commonly used for temporary storage of data within a digital system.

As seen above the 4-bit register can store 4-bit binary information. In general, n-bit register can store n-bit binary information.

Example 5.1.1 Determine the number of flip-flops needed to construct a register capable of storing,

- i) A 6-bit binary number
- ii) Decimal numbers up to 32
- iii) Hexadecimal numbers up to F
- iv) Octal numbers up to 10.

Solution :

- i) A 6-bit binary number requires register with 6 flip-flops.
- ii) $(32)_{10} = (100000)_2$. The number of bits required to represent 32 in binary are six, therefore, 6 flip-flops are needed to construct a register capable of storing 32 decimal.
- iii) $(F)_{16} = (1111)_2$. The number of bits required to represent $(F)_{16}$ in binary are four, therefore four flip-flops are needed to construct a register capable of storing $(F)_{16}$.
- iv) $(10)_8 = (1000)_2$. The number of bits required to represent $(10)_8$ in binary are four, therefore, four flip-flops are needed to construct a register capable of storing $(10)_8$.

Review Questions

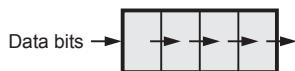
1. What is register ?
2. What is buffer register ?
3. Draw and explain 4-bit controlled buffer register.

5.2 Shift Registers

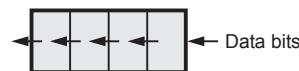
The binary information (data) in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called '**shift registers**'. They are very important in applications involving the storage and transfer of data in a digital system.

Fig. 5.2.1 gives the symbolical representation of the different types of data movement in shift register operations.

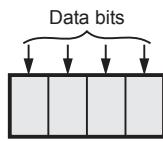
According to the data movement in a register, let us see some of the types of shift registers.



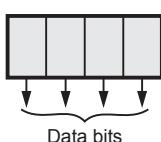
(a) Serial shift right, then out



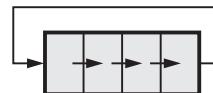
(b) Serial shift left, then out



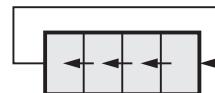
(c) Parallel shift in



(d) Parallel shift out



(e) Rotate right



(f) Rotate left

Fig. 5.2.1 Basic data movement in registers

Review Question

- List the basic types of shift registers in terms of data movements.

5.3 Types of Shift Registers

SPPU : May-05,06,08,10,14, Dec.-05,08,11,12,14

5.3.1 Serial In Serial Out (SISO) Shift Register

Shift Left Mode

Fig. 5.3.1 shows serial-in serial-out shift-left register.

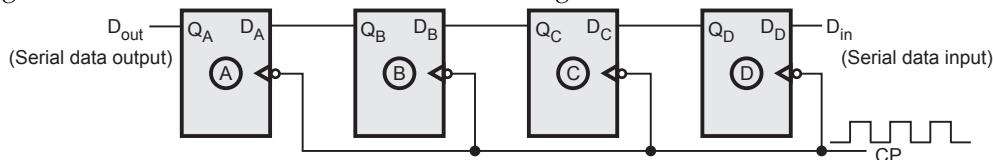
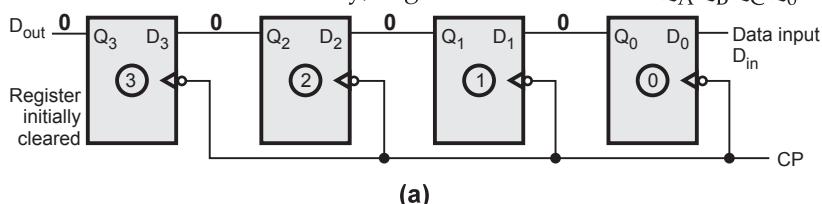
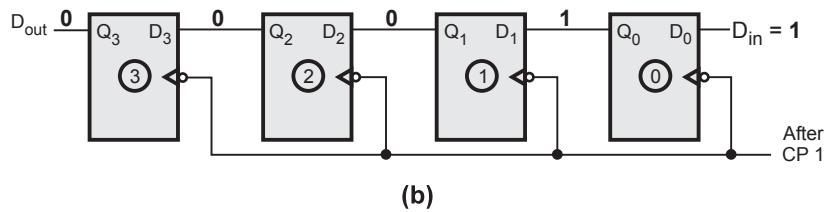


Fig. 5.3.1 Shift-left register

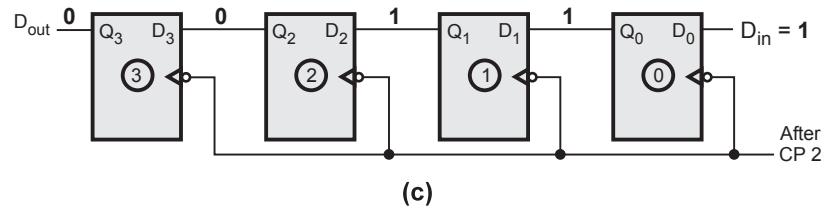
We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit. Initially, register is cleared. So $Q_A Q_B Q_C Q_D = 0 \ 0 \ 0 \ 0$



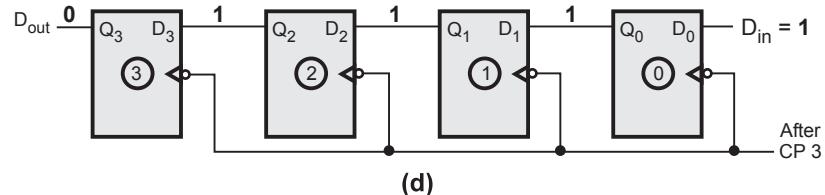
- When data 1 1 1 1 is applied serially, i.e. left-most 1 is applied as D_{in} , $D_{in} = 1$, $Q_3 Q_2 Q_1 Q_0 = 0 \ 0 \ 0 \ 0$. The arrival of the first falling clock edge sets the right-most flip-flop, and the stored word becomes, $Q_3 Q_2 Q_1 Q_0 = 0 \ 0 \ 0 \ 1$



- b) When the next negative clock edge hits, the Q₁ flip-flop sets and the register contents become, Q₃Q₂Q₁Q₀ = 0 0 1 1



- c) The third negative clock edge results in, Q₃Q₂Q₁Q₀ = 0 1 1 1



- d) The fourth falling clock edge gives, Q₃Q₂Q₁Q₀ = 1 1 1 1.

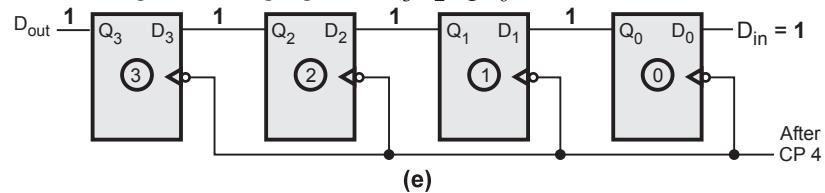


Fig. 5.3.2 Four bits 1111 being serially entered into shift-left register

The Table 5.3.1 summarizes the shift left operation.

CP	Q ₃	Q ₂	Q ₁	Q ₀	D _{in}
Initially	0	0	0	0	1
↓ 1 st	0	0	0	1	1
↓ 2 nd	0	0	1	1	1
↓ 3 rd	0	1	1	1	1
↓ 4 th	1	1	1	1	1

Table 5.3.1 Shift left operation

The Fig. 5.3.3 shows waveforms for shift left operation

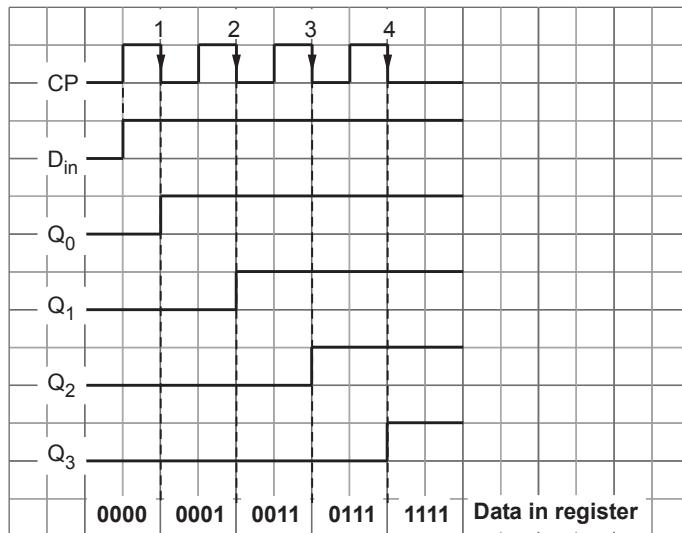


Fig. 5.3.3 Waveforms for shift left register

Shift Right Mode

Fig. 5.3.4 shows serial-in serial-out shift-left register.

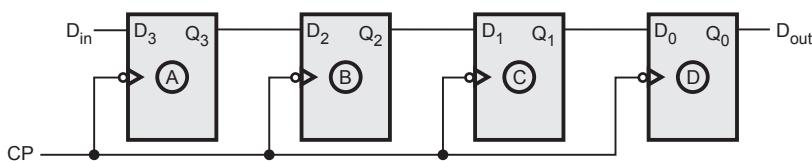
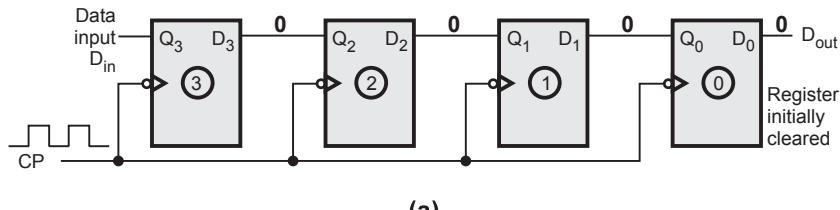


Fig. 5.3.4 Shift-right register

We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit.

Initially, register is cleared. So Q₃Q₂Q₁Q₀ = 0 0 0 0

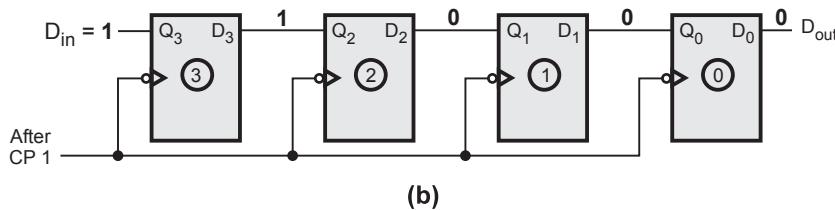


(a)

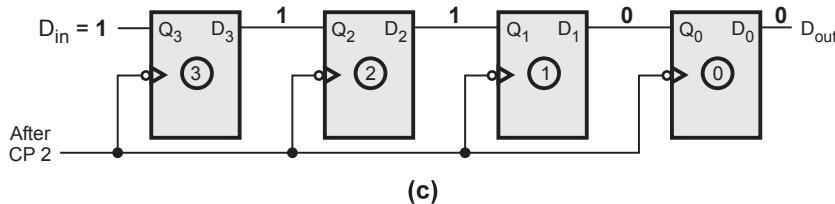
a) When data 1 1 1 1 is applied serially, i.e. left-most 1 is applied as D_{in},

$$D_{in} = 1, Q_3Q_2Q_1Q_0 = 0 0 0 0$$

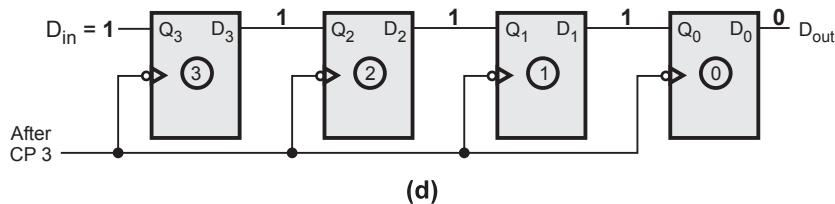
The arrival of the first falling clock edge sets the left-most flip-flop, and the stored word becomes, Q₃Q₂Q₁Q₀ = 1 0 0 0



- b) When the next falling clock edge hits, the Q₂ flip-flop sets and the register contents become, Q₃Q₂Q₁Q₀ = 1100



- c) The third falling clock edge results in, Q₃Q₂Q₁Q₀ = 1110.



- d) The fourth falling clock edge gives, Q₃Q₂Q₁Q₀ = 1111.

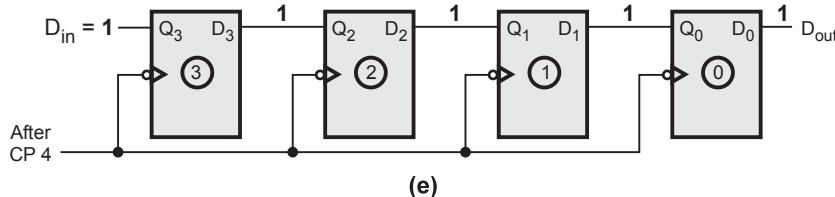


Fig. 5.3.5 Four bits 1111 being serially entered into shift-right register

Table 5.3.2 summarizes the shift right operation.

CP	D _{in}	Q ₃	Q ₂	Q ₁	Q ₀
Initially		1	0	0	0
↓ 1 st	1	1	0	0	0
↓ 2 nd	1	1	1	0	0
↓ 3 rd	1	1	1	1	0
↓ 4 th	1	1	1	1	1

Table 5.3.2 Shift right operation

The Fig. 5.3.6 shows waveforms for shift right operation.

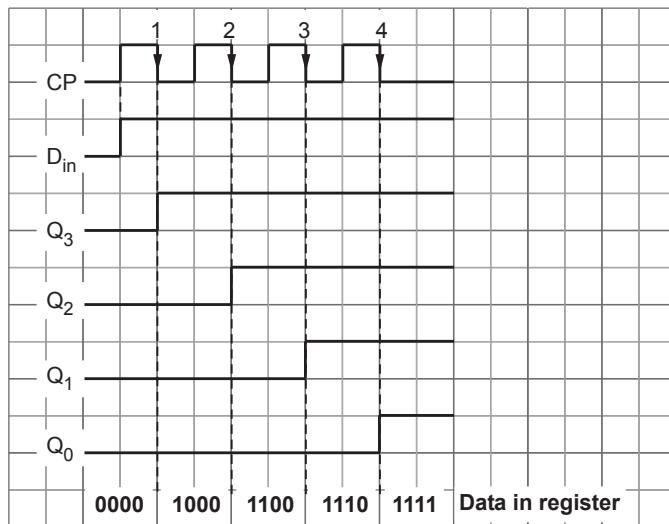


Fig. 5.3.6 Waveforms for shift right register

5.3.2 Serial In Parallel Out (SIPO) Shift Register

In this case, the data bits are entered into the register in the same manner as discussed in the last section, i.e. serially. But the output is taken in parallel. Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously, instead of on a bit-by-bit basis as with the serial output as shown in Fig. 5.3.7.

CP	Q_3	Q_2	Q_1	Q_0
-	NC	NC	NC	NC
↓	D_3	D_2	D_1	D_0

Table 5.3.3 Truth table

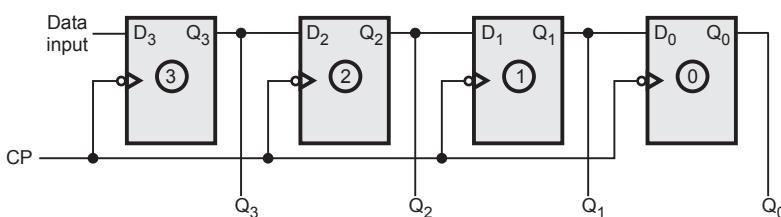


Fig. 5.3.7 A Serial In Parallel Out (SIPO) shift register

5.3.3 Parallel In Serial Out (PISO) Shift Register

SPPU : May-10, Marks 6

In this type, the bits are entered in parallel i.e simultaneously into their respective stages on parallel lines.

Fig. 5.3.8 illustrates a four-bit parallel in serial out register. There are four input lines A_3, A_2, A_1, A_0 for entering data in parallel into the register. SHIFT/LOAD is the control input which allows shift or loading data operation of the register. When SHIFT/LOAD

is low, gates G_1 , G_2 , G_3 are enabled, allowing each input data bit to be applied to D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with D = 1 will SET and those with D = 0 will RESET. Thus all four bits are stored simultaneously.

When SHIFT/LOAD is high, gates G_1 , G_2 , G_3 are disabled and gates G_4 , G_5 , G_6 are enabled. This allows the data bits to shift right from one stage to the next. The OR gates at the D-inputs of the flip-flops allow either the parallel data entry operation or shift operation, depending on which AND gates are enabled by the level on the SHIFT/LOAD input.

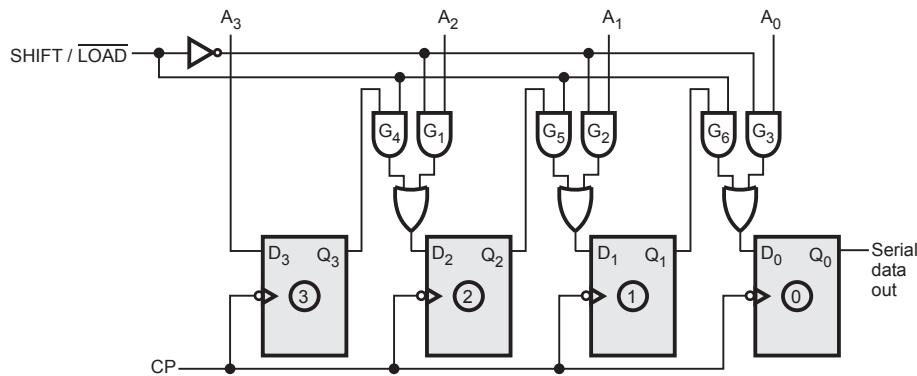


Fig. 5.3.8 Parallel In Serial Out (PISO) shift register

5.3.4 Parallel In Parallel Out (PIPO) Shift Register

From the third and second types of registers, it is cleared that how to enter the data in parallel i.e. all bits simultaneously into the register and how to take data out in parallel from the register. In 'parallel in parallel out register', there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously. Fig. 5.3.9 shows this type of register.

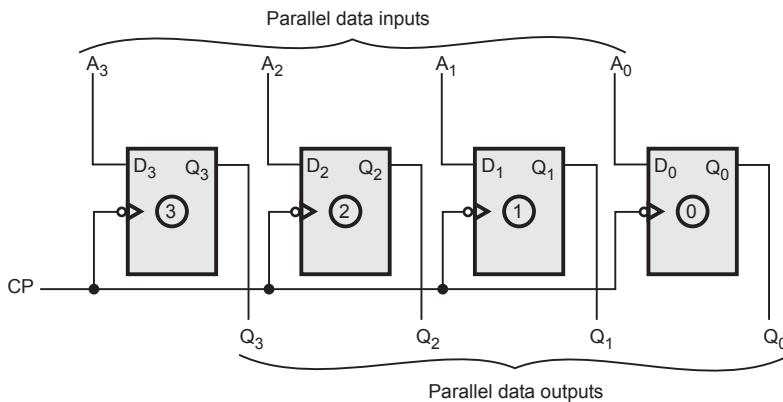


Fig. 5.3.9 Parallel In Parallel Out (PIPO) shift register

5.3.5 Bidirectional Shift Register

This type of register allows shifting of data either to the left or to the right side. It can be implemented by using logic gate circuitry that enables the transfer of data from one stage to the next stage to the right or to the left, depending on the level of a control line. Fig. 5.3.10 illustrates a four-bit bidirectional register. The RIGHT/LEFT is the control input signal which allows data shifting either towards right or towards left. A high on this line enables the shifting of data towards right and a low enables it towards left. When RIGHT/LEFT signal is high, gates G_1, G_2, G_3, G_4 are enabled. The state of the Q output of each flip-flop is passed through the D input of the following flip-flop. When a clock pulse arrives, the data are shifted one place to the right. When the RIGHT/LEFT signal is low, gates G_5, G_6, G_7, G_8 are enabled. The Q output of each flip-flop is passed through the D input of the preceding flip-flop. When clock pulse arrives, the data are shifted one place to the left.

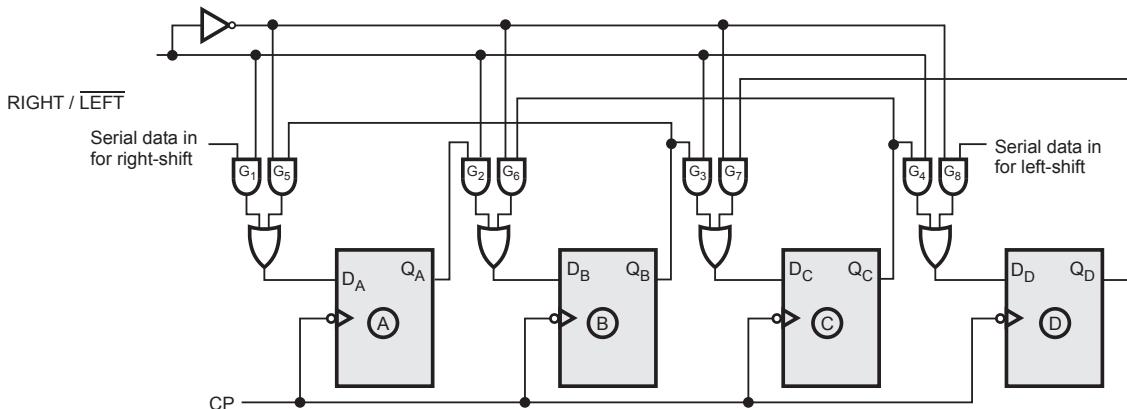


Fig. 5.3.10 4-bit bidirectional shift register

Bidirectional Shift Register with Parallel Load

We have seen that shift register can be used for converting serial data into parallel data, and vice versa. When parallel load capability is added to the shift register, the data entered in parallel can be taken out in serial fashion by shifting the data stored in the register. Such a register is called bidirectional shift register with parallel load. Fig. 5.3.11 shows bidirectional shift register with parallel load.

As shown in the Fig. 5.3.11, the D input of each flip-flop has three sources : Output of left adjacent flip-flop, output of right adjacent flip-flop and parallel input. Out of these three sources one source is selected at a time and it is done with the help of decoder. The decoder select lines (SL_1 and SL_0) select the one source out of three as shown in the Table 5.3.4.

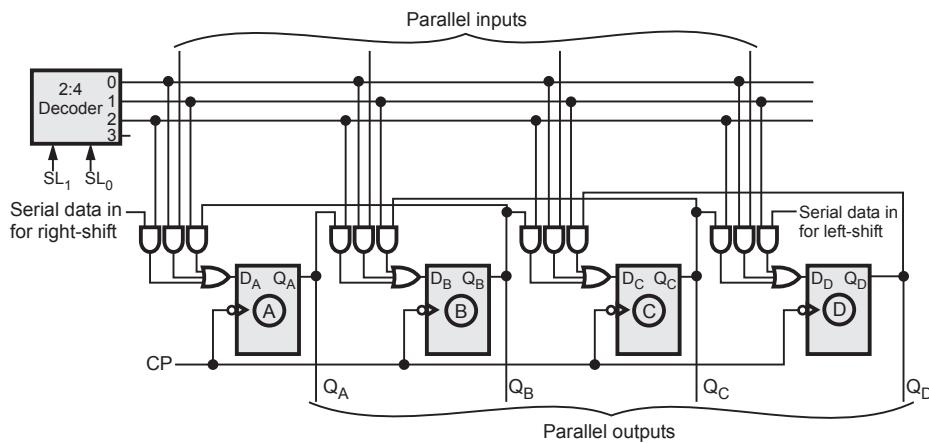


Fig. 5.3.11 4-bit bidirectional shift register with parallel load

When select lines are 00 (i.e. $SL_1 = 0$ and $SL_0 = 0$), data from the parallel inputs is loaded into the 4-bit register. When select lines are 01 (i.e. $SL_1 = 0$ and $SL_0 = 1$), data within the register is shifted 1-bit left. When select lines are 10 (i.e. $SL_1 = 1$ and $SL_0 = 0$), data within the register is shifted 1-bit right.

SL_1	SL_0	Selected source
0	0	Parallel input
0	1	Output of right adjacent FF
1	0	Output of left adjacent FF

Table 5.3.4

Review Questions

- Draw the logical diagram of a 4-bit shift register. Explain how shift left and shift right operations are performed.
- Explain modes of operation of shift register.
- Explain with a neat diagram working of parallel in serial out 4-bit shift register. Draw necessary timing diagram. **SPPU : May-10, Marks 6**
- Draw and explain the circuit diagram of 3-bit register with the following facilities :
 - Serial left shift
 - Serial right shift
 - Parallel In-Serial-Out
 - Reset.**SPPU : May-05, Marks 10**
- Draw and explain the circuit diagram of 3-bit register with the following facility
 - Parallel in serial output
 - Reset.**SPPU : Dec.-08, Marks 6**
- Explain how shift registers are used as serial to parallel converters. **SPPU : Dec.-08, Marks 3**
- Draw and explain 5-bit shift register having parallel in and serial in (left to right) facilities. Explain any one application of such register. **SPPU : May-08, Marks 8**
- Explain serial to parallel shift register with neat circuit diagram and timing diagram. **SPPU : Dec.-08, Marks 3; Dec.-12, Marks 8**

9. Explain with the help of neat diagram, the operation of 3-bit bidirectional shift register.

SPPU : Dec.-05, Marks 8

10. Draw the circuit and explain the function of 4-bit bidirectional shift register.

SPPU : May-06, Dec.-11, Marks 8

11. Explain with neat diagram working of parallel in serial out 4-bit shift register. Draw necessary timing diagram.

SPPU : May-14 , Marks 6

12. Explain with neat diagram working of serial-in serial-out 4-bit shift register. Draw necessary timing diagram.

SPPU : Dec.-14, Marks 6

5.4 Universal Shift Register

SPPU : Dec.-06,07, May-07

A register capable of shifting in one direction only is a unidirectional shift register. A register capable of shifting in both directions is a bidirectional shift register. If the register has both shifts (right shift and left shift) and parallel load capabilities, it is referred to as **universal shift register**.

The Fig. 5.4.1 shows the 4-bit universal shift register. It has all the capabilities listed above. It consists of four flip-flops and four multiplexers. The four multiplexers have two common selection inputs S_1 and S_0 , and they select appropriate input for D flip-flop. The Table 5.4.1 shows the register operation depending on the selection inputs of multiplexers. When $S_1S_0 = 00$, input 0 is selected and the present value of the register is applied to the D inputs of the flip-flops. This results no change in the register value. When $S_1S_0 = 01$, input 1 is selected and circuit connections are such that it operates as a right shift register. When $S_1S_0 = 10$, input 2 is selected and circuit connections are such that it operates as a left shift register. Finally, when $S_1S_0 = 11$, the binary information on the parallel input lines is transferred into the register simultaneously and it is a parallel load operation. (Refer Fig. 5.4.1 on next page)

Mode control		Register operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Table 5.4.1 Mode control and register operation

Example 5.4.1 Draw the logic diagram of a 4-bit shift register with four D flip-flops and four 4×1 multiplexer with mode selection inputs S_1 and S_0 . The register operates as follows.

S_1	S_0	Register operation
0	0	No change
0	1	Complement
1	0	Clear to 0
1	1	Load parallel data

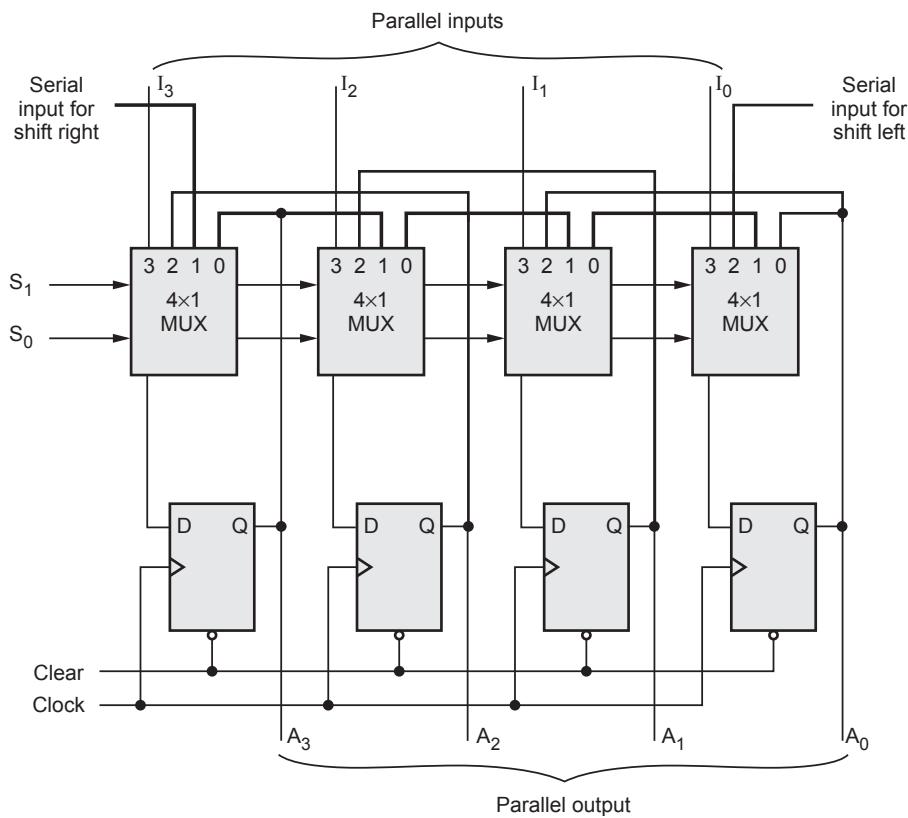


Fig. 5.4.1 4-bit universal shift register

Solution : The Fig. 5.4.2 shows the register that does the given operations.

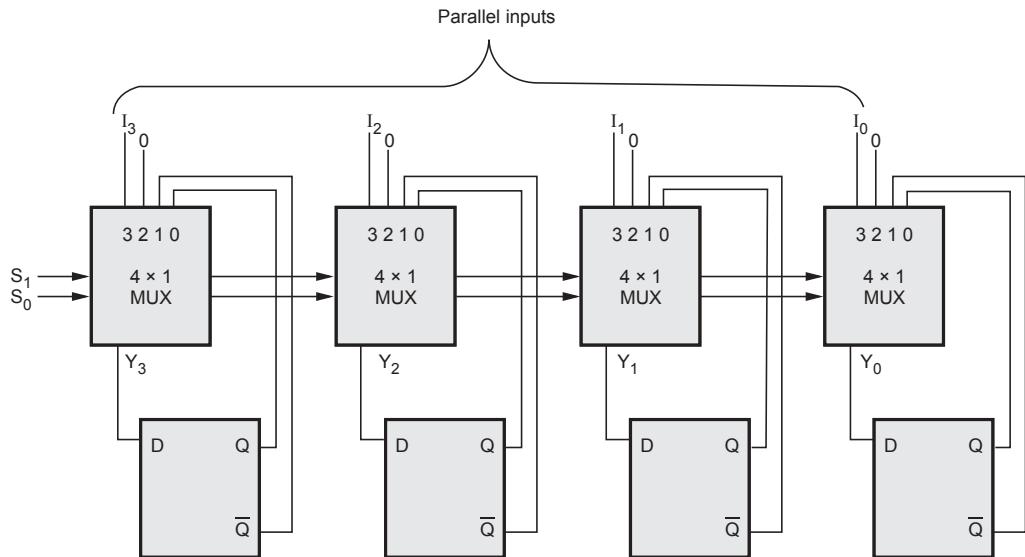


Fig. 5.4.2

Review Questions

1. Explain with the help of neat diagram the operation of 3-bit universal shift register.
2. 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.

SPPU : Dec.-06, Marks 8

3. Design 4-bit shift register with the following facilities :

SPPU : May-07, Dec.-07, Marks 10

- i) Shift-left (if $L_1 = 0$ and $L_0 = 0$)
- ii) Shift-right (if $L_1 = 0$ and $L_0 = 1$)
- iii) Parallel-in (if $L_1 = 1$ and $L_0 = 1$)

Control signals 'L1 L0' will select one of the possible operation.

5.5 Applications of Shift Registers

SPPU : May-10

We have seen that primary use of shift register is temporary data storage and bit manipulations. Some of the common applications of shift registers are as discussed below.

5.5.1 Delay Line

A Serial-In-Serial-Out (SISO) shift register can be used to introduce time delay Δt in digital signals. The time delay can be given as

$$\Delta t = N \times \frac{1}{f_c}$$

where N is the number of stages (i.e. flip-flops) and f_c is the clock frequency.

Thus, an input pulse train appears at the output delayed by Δt . The amount of delay can be controlled by the clock frequency or by the number of flip-flops in the shift register.

5.5.2 Serial-to-Parallel Converter

A Serial-In-Parallel-Out (SIPO) shift register can be used to convert data in the serial form to the parallel form.

5.5.3 Parallel-to-Serial Converter

A Parallel-In-Serial-Out (PISO) shift register can be used to convert data in the parallel form to the serial form.

5.5.4 Shift Register Counters

A shift register can also be used as a counter. A shift register with the serial output connected back to the serial input is called **shift register counter**. Because of such a

connection, special specified sequences are produced as the output. The most common shift register counters are the ring counter and the Johnson counter.

5.5.5 Pseudo-Random Binary Sequence (PRBS) Generator

Another important application of shift register is a pseudo-random binary sequence generator. Here, suitable feedback is used to generate pseudo-random sequence. The term random here means that the outputs do not cycle through a normal binary count sequence. The term pseudo here refers to the fact that the sequence is not truly random because it does cycle through all possible combinations once every $2^n - 1$ clock cycles, where n represents the number of shift register stages (number of flip-flops).

5.5.6 Sequence Generator

The shift register can be used to generate a particular bit pattern repetitively. The Fig. 5.5.1 shows the basic block diagram of a sequence generator. Here, left

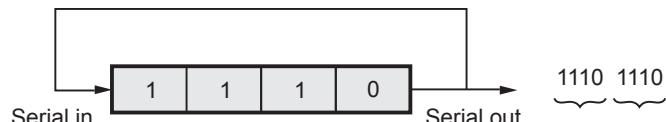


Fig. 5.5.1 4-bit sequence generator

most flip-flop input accept the serial input and the right most flip-flop gives serial data output. It is important to note that the serial data output signal is connected as a serial data in. On every clock pulse the data shift operation takes place. We get the loaded bit pattern at the serial output in a sequence. Same bit pattern is again loaded in the register since serial output is connected serial in of the register. Thus, the circuit generates a particular bit pattern repetitively.

5.5.7 Sequence Detector

The shift register can be used to detect the desired sequence. The detection process requires two registers : one register stores the bit pattern to be detected i.e. R_1 and other register accepts the input data stream i.e. R_2 . Input data stream enters a shift register as serial data in and leaves as serial out. In every clock cycle, bit-wise comparisons of these two registers are done using EX-NOR gates as shown in the Fig. 5.5.2. We know that, the two-input EX-NOR gate gives logic high output when both inputs are either low or high, i.e. when both are equal. When outputs of all the EX-NORs gates are logic high we can say that all bits are

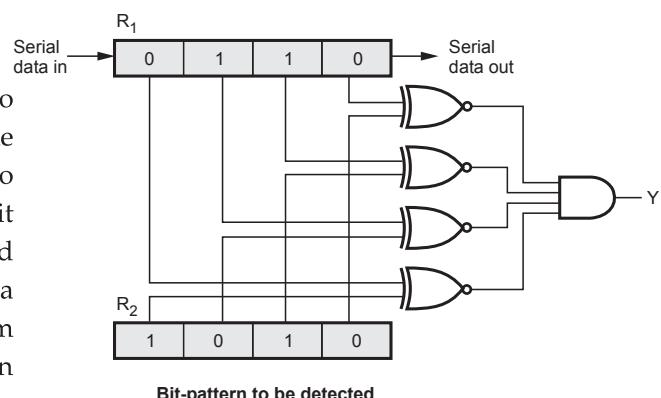


Fig. 5.5.2 4-bit sequence detector

matched and hence the desired bit pattern is detected. The final output which indicates that the pattern is detected is taken from four-input AND gate.

The 4-bit sequence detector shown in Fig. 5.5.2 can be made programmable by loading the desired 4-bit data in the register R₂.

Review Question

1. Give any four applications of shift registers.

SPPU : May-10, Marks 4

UNIT - III

6

Counters

Syllabus

Asynchronous counter, Synchronous Counter, BCD Counter, Ring Counter, Johnson Counter Modulus of the counter (IC 7490).

Contents

6.1	<i>Introduction</i>	May-07,12,13, Dec.-12,18, · Marks 4
6.2	<i>Ripple / Asynchronous Counters</i>	May-07,08,11, Dec.-05,07, Marks 8
6.3	<i>Design of Ripple (Asynchronous) Counters</i>	..	Dec.-07, May-07, Marks 4
6.4	<i>Synchronous Counters.</i>	May-07,08,10, Dec.-10,12, Marks 10
6.5	<i>Design of Synchronous Counters</i>	May-12,14,15,18, Dec.-13,19, Marks 6
6.6	<i>Lock-Out</i>	Dec.-11,15,18, May-13, · Marks 6
6.7	<i>IC 7490 (Decade Binary Counter)</i>	Dec.-04,05,06,10,11,12,17, May-07,11,12,13,15,17, .. Marks 10
6.8	<i>Ring Counter</i>	Dec.-08,10,19, May-11 · Marks 10
6.9	<i>Johnson or Twisting Ring or Switch Tail Counter</i>	Dec.-08,17,18, Marks 4

6.1 Introduction

SPPU : May-07,12,13, Dec.-12, 18

A group of flip-flops connected together forms a **register**. A register is used solely for storing and shifting data which is in the form of 1s and/or 0s, entered from an external source. It has no specific sequence of states except in certain very specialized applications. A **counter** is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. On arrival of each clock pulse, the counter is incremented by one. In case of down counter, it is decremented by one.

The Fig. 6.1.1 shows the logic symbol of a binary counter. External clock is applied to the clock input of the counter. The counter can be positive edge triggered or negative edge triggered. The

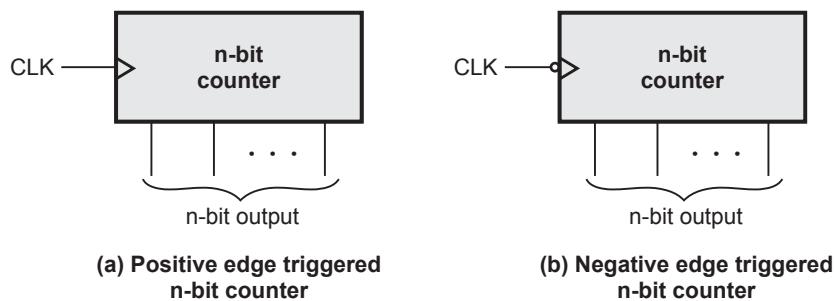


Fig. 6.1.1 Logic symbol of counter

n-bit binary counter has n flip-flops and it has 2^n distinct states of outputs. For example, 2-bit counter has 2 flip-flops and it has $4(2^2)$ distinct states : 00, 01, 10 and 11. Similarly, the 3-bit binary counter has 3 flip-flops and it has $8(2^3)$ distinct states : 000, 001, 010, 011, 100, 101 110 and 111.

The maximum count that the binary counter can count is 2^{n-1} . For example, in 2-bit binary counter, the maximum count is $2^2 - 1 = 3$ (11 in binary). After reaching the maximum count the counter resets to 0 on arrival of the next clock pulse and it starts counting again.

Synchronous Counter : When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called synchronous counter.

Asynchronous Counter / Ripple Counter : A binary asynchronous/ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming clock pulses.

The Table 6.1.1 shows the comparison between synchronous and asynchronous counters.

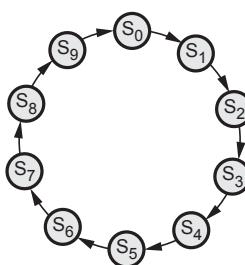
Sr. No.	Asynchronous counters	Synchronous counters
1.	In this type of counter flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
2.	All the flip-flops are not clocked simultaneously.	All the flip-flops are clocked simultaneously.
3.	Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of states increases.
4.	Main drawback of these counters is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are high speed counters and are preferred when number of flip-flops increases in the given design.

Table 6.1.1 Synchronous Vs asynchronous counters**Modulus of Counter**

The total number of counts or stable states a counter can indicate is called 'Modulus'. For instance, the modulus of a four-stage counter would be 16_{10} , since it is capable of indicating 0000_2 to 1111_2 . The term 'modulo' is used to describe the count capability of counters. For example, mod 6 counter goes through states 0 to 5 and mod 4 counter goes through states 0 - 3.

Example 6.1.1 Draw the state diagram of MOD-10 counter.

Solution :

**Fig. 6.1.2**

Example 6.1.2 Assume that the 5-bit binary counter starts in the 00000 state. What will be the count after 144 input pulses ?

Solution : $(144)_{10} = (1001\ 0000)_2$

Since counter is a 5-bit counter, it resets after $2^5 = 32$ clock pulses.

$$\begin{array}{r}
 & \quad 4 \\
 32) & 144 \\
 & - 128 \\
 & \hline
 & 16
 \end{array}$$

Therefore, counter resets four times and then it counts remaining 16 clock pulses. Thus, the count will be $(10000)_2$, i.e., 16 in decimal.

In general,

$$\text{Count} = \text{Remainder of } \left[\frac{\text{Number of input pulses}}{2^n} \right] \text{ in binary}$$

where n = Number of counter bits

Review Questions

1. What is counter ?
2. State types of counters.
3. Compare synchronous and asynchronous counter.
4. What is MOD counter ?
5. Differentiate between synchronous and asynchronous counters.

SPPU : May-13, Marks 2

SPPU : May-07,12, Dec.-12,18, Marks 4

6.2 Ripple / Asynchronous Counters

SPPU : May-07,08,11, Dec.-05,07

A binary ripple/asynchronous counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming clock pulses. A complementing flip-flops can be obtained from a JK flip-flop with the J and K inputs tied together as shown in the Fig. 6.2.1 or from a T flip-flop. A third alternative is to use a D flip-flop with the complement output connected to the D input. In this way, the D input is always the complement of the present state and the next clock pulse will cause the flip-flop to complement. Let us see the ripple counter using JK flip-flop.

Fig. 6.2.1 (a) shows 2-bit asynchronous counter using JK flip-flops. As shown in Fig. 6.2.1 (a), the clock signal is connected to the clock input of only first stage flip-flop. The clock input of the second stage flip-flop is triggered by the Q_A output of the first stage. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the Q_A output of first stage can never occur at exactly the same time. Therefore, the two

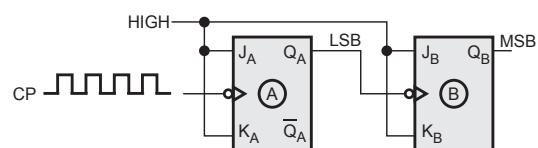


Fig. 6.2.1 (a) A two-bit asynchronous binary counter

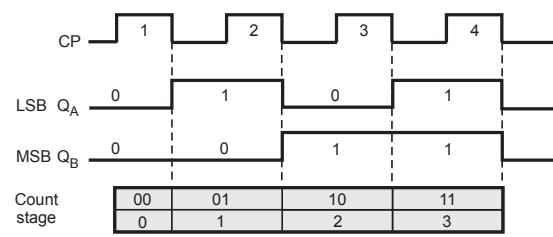


Fig. 6.2.1 (b) Timing diagram for the counter of Fig. 6.2.1 (a)

flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

Fig. 6.2.1 (b) shows the timing diagram for two-bit asynchronous counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock. J and K input of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

Example 6.2.1 Extend the counter shown in Fig. 6.2.1 (a) for 3-stages, and draw output waveforms.

Solution :

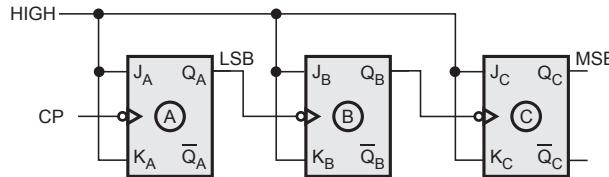


Fig. 6.2.2 (a) Logic diagram

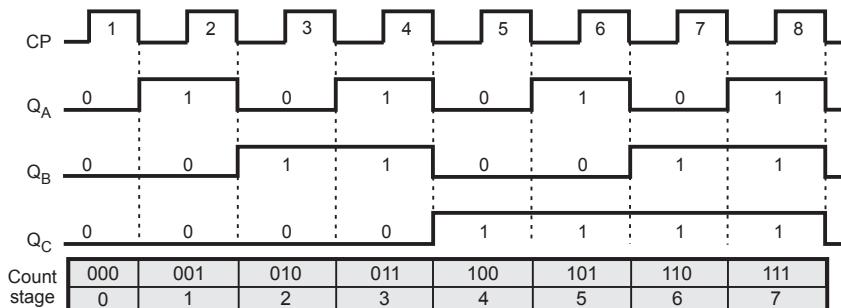


Fig. 6.2.2 (b) Output waveforms for 3-bit asynchronous counter

In Fig. 6.2.2 (b), timing diagram for 3-bit asynchronous counter we have not considered the propagation delays of flip-flops, for simplicity. If we consider the propagation delays of flip-flops we get timing diagram as shown in Fig. 6.2.3.

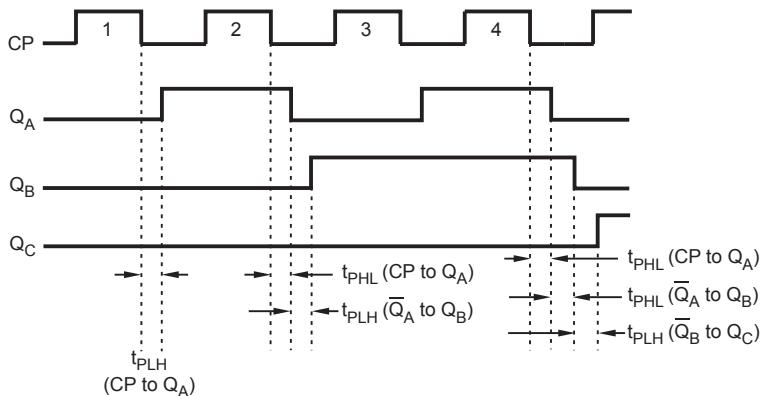


Fig. 6.2.3 Propagation delays in a ripple clocked binary counter

The timing diagram shows propagation delays. We can see that propagation delay of the first stage is added in the propagation delay of second stage to decide the transition time for third stage. This cumulative delay of an asynchronous counter is a major

disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems.

Example 6.2.2 Draw the logic diagram for 3-stage asynchronous counter with negative edge triggered flip-flops.

Solution : When flip-flops are negatively edge triggered, the Q output of previous stage is connected to the clock input of the next stage. Fig. 6.2.4 shows 3-stage asynchronous counter with negative edge triggered flip-flops.

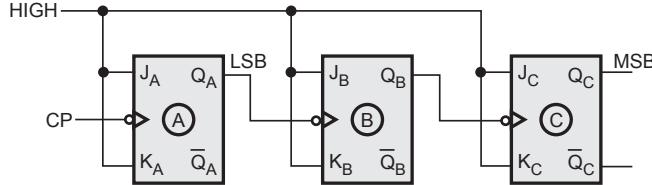


Fig. 6.2.4 Logic diagram of 3-stage negative edge triggered counter

Example 6.2.3 A counter has 14 stable states 0000 through 1101. If the input frequency is 50 kHz what will be its output frequency ?

$$\text{Solution : } \frac{50 \text{ kHz}}{14} = 3.57 \text{ kHz}$$

Example 6.2.4 The t_{pd} for each flip-flop is 50 ns, determine the maximum operating frequency for MOD-32 ripple counter.

Solution : We know that MOD-32 uses five flip-flops. With $t_{pd} = 50$ ns, the f_{\max} for ripple counter can be given as,

$$f_{\max}(\text{ripple}) = \frac{1}{5 \times 50 \text{ ns}} = 4 \text{ MHz}$$

Example 6.2.5 Draw the logic diagram for 4-stage asynchronous counter with positive edge triggered flip-flops. Also draw necessary timing diagram. Is there any frequency division concept in it.

SPPU : Dec.-07, May-08, Marks 8

Solution : When flip-flops are positive edge triggered, the \bar{Q} output of previous stage is connected to the clock input of the next stage. Fig. 6.2.5 shows 4-stage asynchronous counter with positive edge triggered flip-flops.

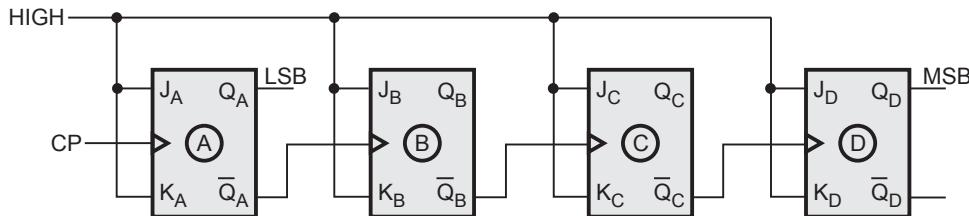


Fig. 6.2.5 Logic diagram of 4-stage positive edge triggered ripple counter

The Fig. 6.2.6 shows the timing diagram for 4-bit ripple up counter using positive edge triggered JK-FFs.

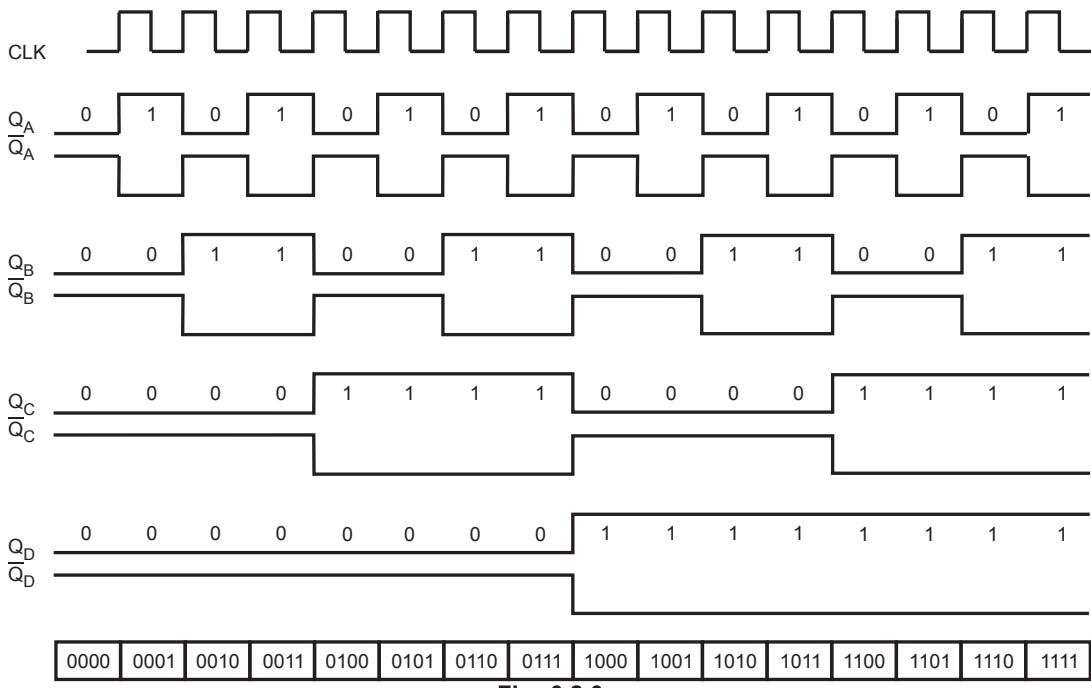


Fig. 6.2.6

From the timing diagram we can observe that

- Frequency at output $Q_A = \frac{F_{CLK}}{2}$
- Frequency at output $Q_B = \frac{Q_A}{2} = \frac{F_{CLK}}{4}$
- Frequency at output $Q_C = \frac{Q_B}{2} = \frac{Q_A}{4} = \frac{F_{CLK}}{8}$
- Frequency at output $Q_D = \frac{Q_C}{2} = \frac{Q_B}{4} = \frac{Q_A}{8} = \frac{F_{CLK}}{16}$

In general we can say that the frequency at the MSB output of counter is $\frac{F_{CLK}}{2^N}$

where N represents number of stages/bits of the counter.

6.2.1 Asynchronous / Ripple Down Counter

In the last section we have seen that the output of counter is incremented by one for each clock transition. Therefore, we call such counters as up **counters**. In this section we see the asynchronous/ripple down counter. The down counter will count downward from a maximum count to zero.

The Fig. 6.2.7 shows the 4-bit asynchronous down counter using JK flip-flops. Here, the clock signal is connected to the clock input of only first flip-flop. This connection is same as asynchronous/ripple up counter. However, the clock input of the remaining

flip-flops is triggered by the \bar{Q}_A output of the previous stage instead of Q_A output of the previous stage.

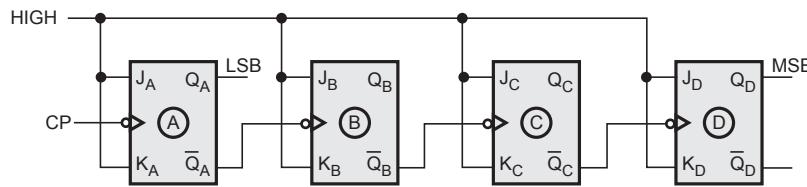


Fig. 6.2.7 4-bit asynchronous down counter

The Fig. 6.2.8

shows the timing diagram for 4-bit asynchronous down counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock. Again the J and K inputs of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

Down counters are not as widely used as up counters. They are used in situations where it must be known when a desired number of input pulses has occurred. In these situations the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state it is detected by a logic gate whose output then indicates that the preset number of pulses have occurred.

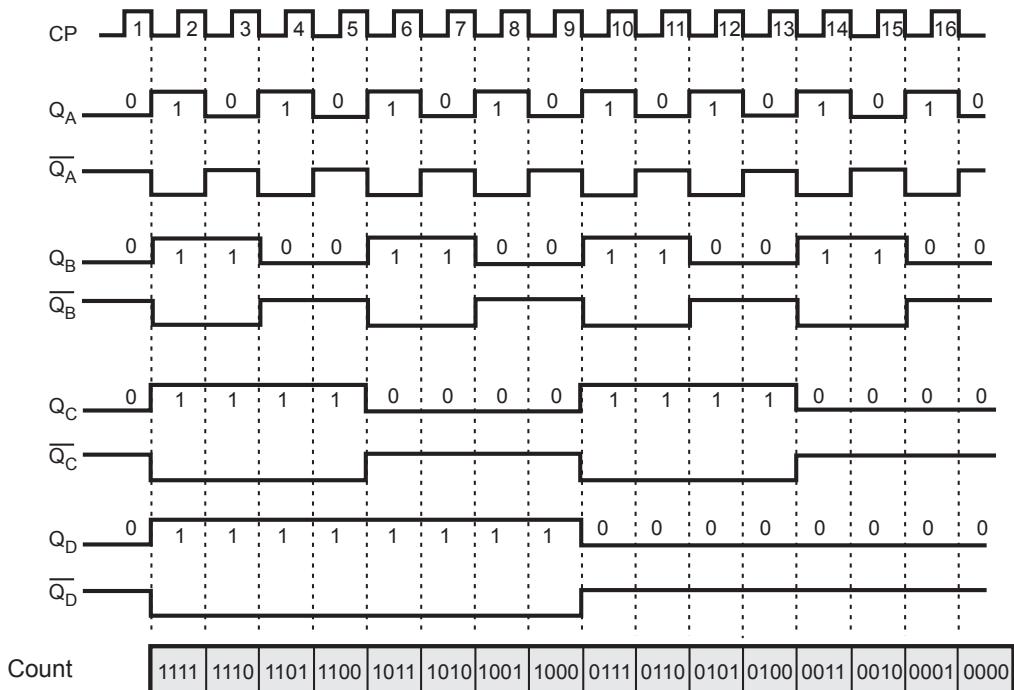
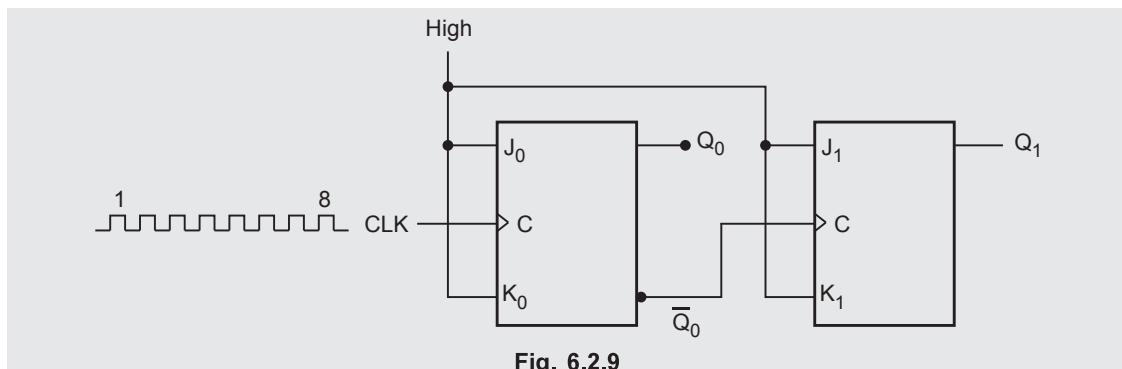
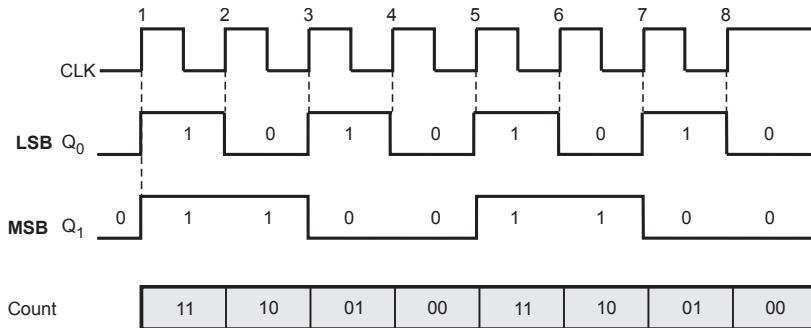


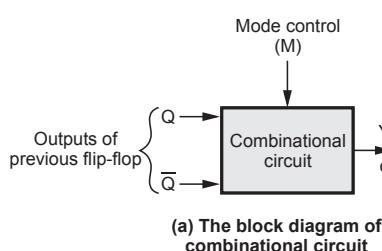
Fig. 6.2.8 Timing diagram of 4-bit asynchronous down counter

Example 6.2.6 For the ripple counter shown in Fig. 6.2.9, show the complete timing diagram for eight clock pulses, showing the clock, Q_0 and Q_1 waveforms.

**Solution :****Fig. 6.2.9 (a)**

6.2.2 Asynchronous Up / Down Counter

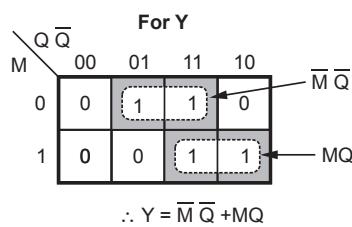
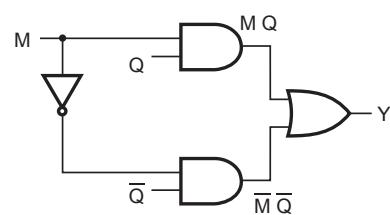
To form an asynchronous up/down counter one control input say M is necessary to control the operation of the up/down counter. When M = 0, the counter will count up and when M = 1, the counter will count down. To achieve this the M input should be used to control whether the normal flip-flop



Inputs	Output
M Q Q	Y
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

$Y = \bar{Q}$ for down counting
 $Y = Q$ for up counting

(b) Truth table

Fig. 6.2.10**a) K-map simplification****b) Logic diagram****Fig. 6.2.11**

output (Q) or the inverted flip-flop output (\bar{Q}) is fed to drive the clock signal of the

successive stage flip-flop, as shown in Fig. 6.2.10 (a). The truth table for such combinational circuit is shown in Fig. 6.2.10 (b).

The Fig. 6.2.12 shows the 3-bit up/down counter that will count from 000 up to 111 when the mode control input M is 1 and from 111 down to 000 when mode control input M is 0.

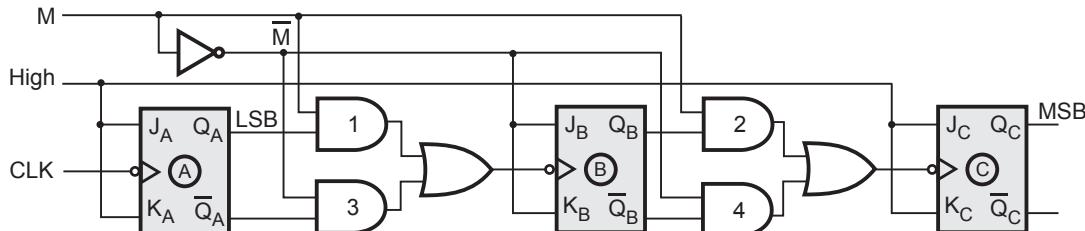


Fig. 6.2.12 3-bit asynchronous up/down counter

A logic 1 on M enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the Q_A and Q_B outputs to drive the clock inputs of their respective next stages. So that counter will count up. When M is logic 0, AND gates 1 and 2 are disabled and AND gate 3 and 4 are enabled. This allows the \bar{Q}_A and \bar{Q}_B outputs to drive the clock inputs of their respective next stages so that counter will count down. The Fig. 6.2.13 shows the timing diagram for 3-bit up/down ripple counter. (See Fig. 6.2.13 on next page).

Example 6.2.7 Design a 4-bit up/down ripple counter with a control for up/down counting.

Solution : The 4-bit counter needs four flip-flops. The circuit for 4-bit up/down ripple counter is similar to 3-bit up/down ripple counter except that 4-bit counter has one more flip-flop and its clock driving circuiting.

The Fig. 6.2.14 shows the 4-bit up/down ripple counter.

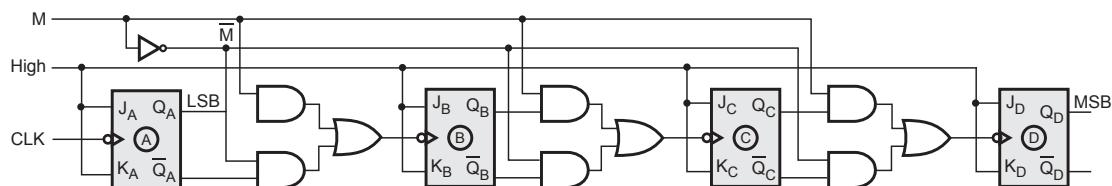


Fig. 6.2.14 4-bit asynchronous up/down counter

6.2.3 Decoding Gates

Decoding gates are used to indicate whether counter has reached to particular state. The outputs of the counter are connected to the AND gate as inputs and the output of the AND gate goes high for particular state. Let us see the Fig. 6.2.14 (a). Here, the output of decoding gate goes high when counter outputs are C = 1, B = 1 and A = 1. In

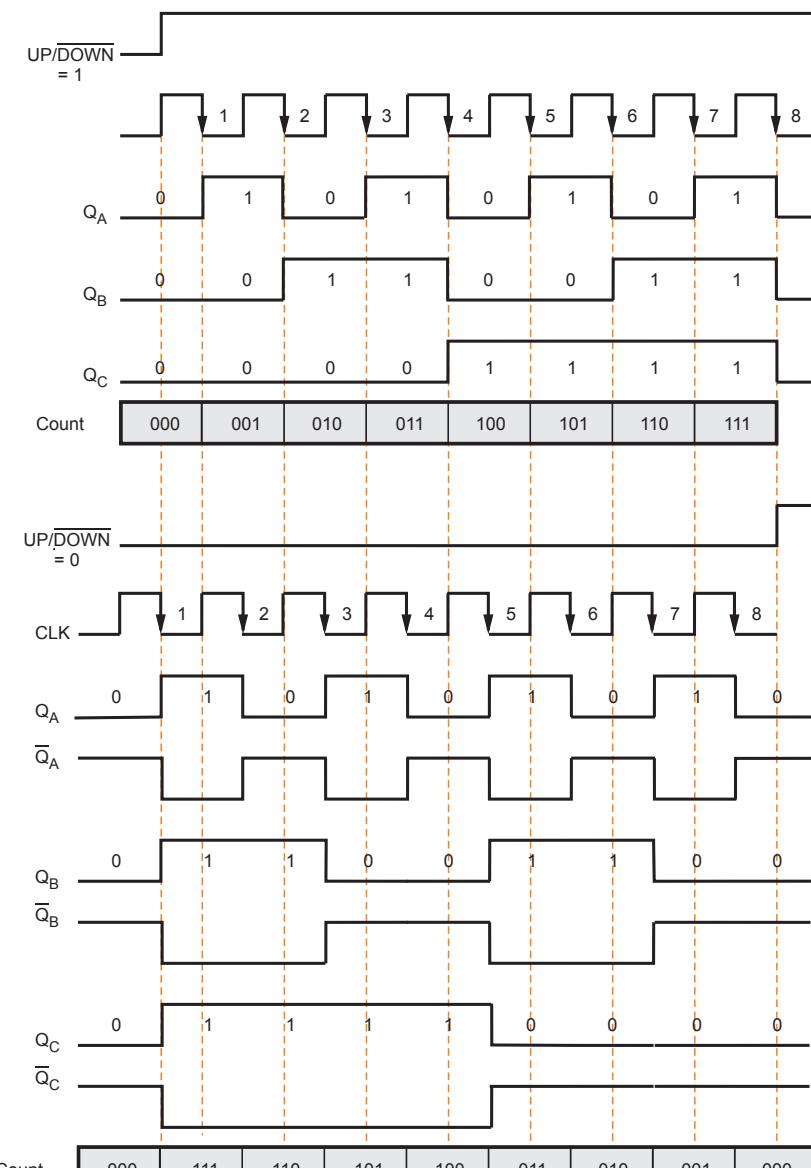


Fig. 6.2.13 Timing diagram for 3-bit UP/ DOWN ripple counter

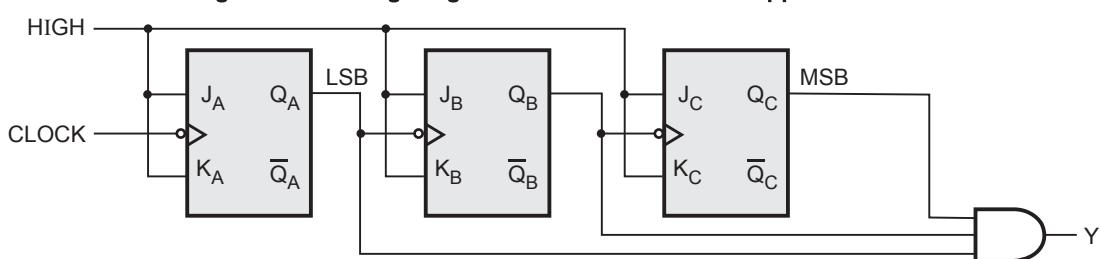


Fig. 6.2.15 (a) Decoding gate to indicate state 7 (111)

Fig. 6.2.15 (b) the output of decoding gate goes high when counter outputs are C = 1, B = 0 and A = 0.

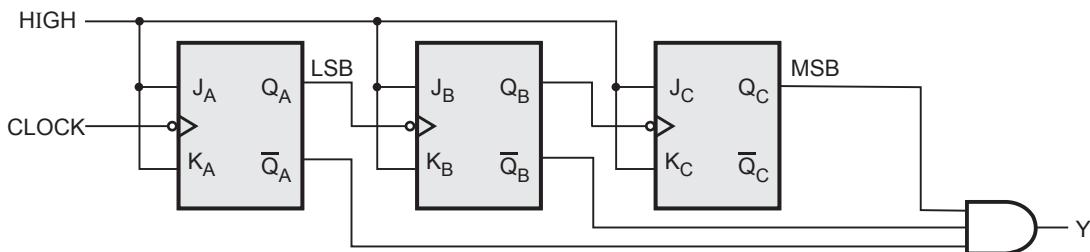


Fig. 6.2.15 (b) Decoding gate to indicate state 4 (100)

Similarly, we can connect corresponding outputs to decoding gate inputs to indicate desired state. The Fig. 6.2.16 gives these connections for all possible state detection for 3-bit counter.

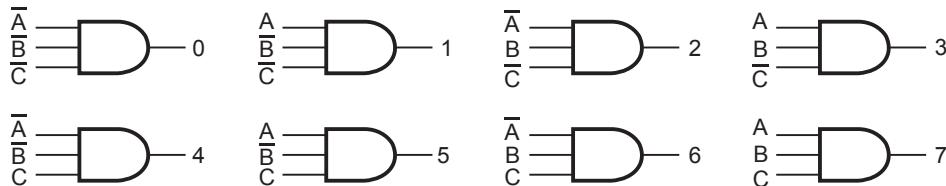


Fig. 6.2.16 Decoding gate connections

6.2.4 Problem Faced by Ripple Counters (Glitch Problem)

We know that, due to the propagation delay the output flip-flop is delayed by time t_p . This illustrated in the waveform shown in Fig. 6.2.17 shows the waveform of the circuit which decodes state 7. Here, the output of flip-flop A triggers the flip-flop B,

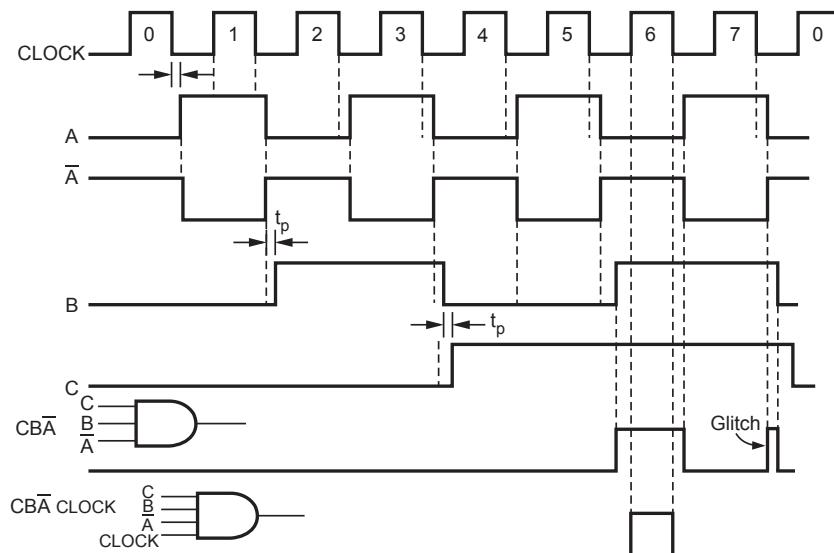


Fig. 6.2.17 Elimination of glitch

hence the B waveform is delayed by one flip-flop delay time (t_p) from the negative transition of A. Similarly, the C waveform is delayed by t_p from each negative transition of B.

Let us observe the output waveform when the counter progresses from state 7 to state 0. At point X, A goes low (\bar{A} goes high); however, because of flip-flop delay time, B does not go low until point Y. Thus between points X and Y we have the condition C = 1, B = 1 and A = 1. As a result, the output is high between points X and Y. This undesirable output is known as **glitch**. We can avoid the glitch on the output waveform by connecting clock as a fourth input to the decoding gate along with inputs A, B and C. This is illustrated in the Fig. 6.2.16.

Review Questions

1. What do you mean by ripple counter ? SPPU : May-08, Marks 2
2. Draw and explain 3-bit asynchronous up counter using flip-flops and explain with output waveforms. SPPU : May-07, Marks 6
3. Draw 4-bit asynchronous counter. Also explain timing diagram for the same. SPPU : May-11, Marks 8
4. Draw and explain 3-bit up/down asynchronous counter.
5. What do you mean by decoding gates ?
6. Explain the problem of glitches in asynchronous counter circuits and solution for the same. SPPU : Dec.-07, Marks 4
7. Explain using suitable waveforms : Problems faced by ripple counter. SPPU : Dec.-05, Marks 6

6.3 Design of Ripple (Asynchronous) Counters

SPPU : Dec.-07, May-07

The steps involved in the design of asynchronous counter are :

1. Determine the number of flip-flops needed.
2. Choose the type of flip-flops to be used : T or JK. If T flip-flops are used connect T input of all flip-flops to logic 1. If JK flip-flops are used connect both J and K inputs of all flip flops to logic 1.
Such connection toggles the flip-flop output on each clock transition.
3. Write the truth table for the counter.
4. Derive the reset logic by K-map simplification.
5. Draw the logic diagram.

Example 6.3.1 Design BCD ripple counter using JK flip-flop.

SPPU : Dec.-07

Solution : **Step 1 :** Determine the number of flip-flops needed. The BCD counter goes through states 0-9, i.e. total 10 states. Thus, N = 10 and for $2^n \geq N$, we need n = 4, i.e. 4 flip-flops required.

Step 2 : Type of flip-flops to be used : JK

Step 3 : Write the truth table for the counter

CLK	D	C	B	A	Output of reset logic Y
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

Note : The reset input (CLR) of each Flip-Flop is active-low input. By making CLR input of all Flip-Flops logic 0, we can reset the counter. Thus reset logic is designed such a way that for invalid states, $Y = 0$ and counter resets.

Valid states

Invalid states

Table 6.3.1 Truth table for BCD counter

Step 4 : Derive reset logic

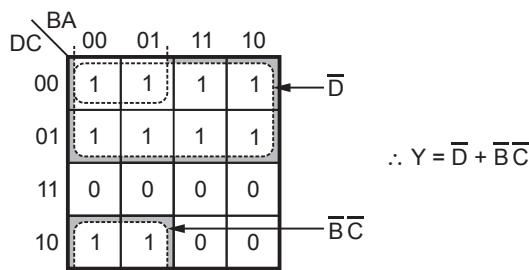


Fig. 6.3.1

Step 5 : Draw logic diagram.

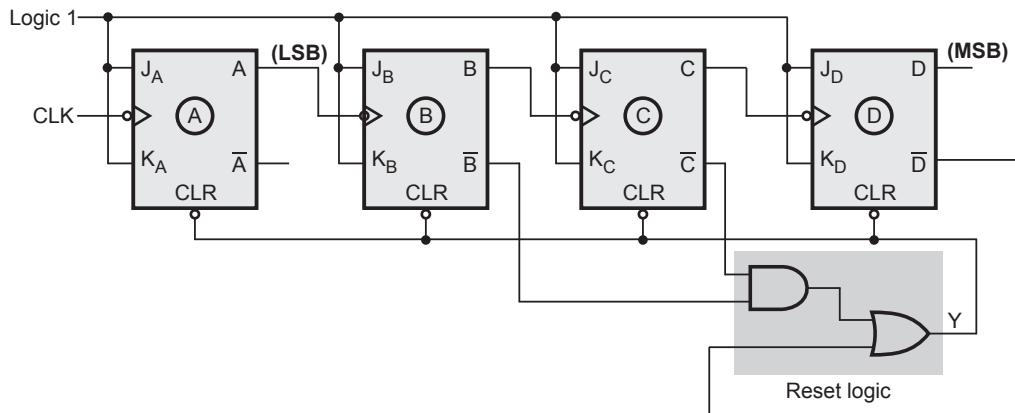


Fig. 6.3.2 Logic diagram of BCD ripple counter

Example 6.3.2 Design a 3-bit asynchronous ripple counter using T flip-flops and explain its operation.

Solution : The Fig. 6.3.3 shows a 3-bit asynchronous ripple counter using T flip-flops. As shown in Fig. 6.3.3, the clock input of only first stage flip-flop. The clock input of the second stage flip-flop is triggered by the Q_A output of the first stage and third stage flip-flop is triggered by the Q_A output of second stage. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the Q_A output of previous stage can never occur at exactly the same time. Therefore, flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

Since, T input is connected to logic 1 each flip-flop toggles at clock input. The Fig. 6.3.4 shows the timing diagram for 3-bit asynchronous counter.

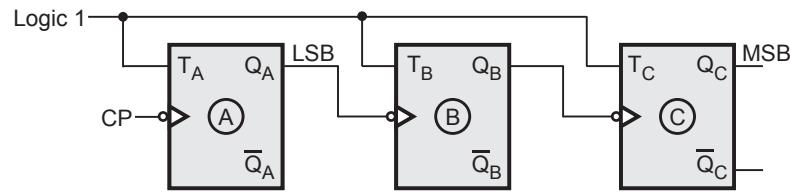


Fig. 6.3.3 3-bit asynchronous counter using T flip-flops

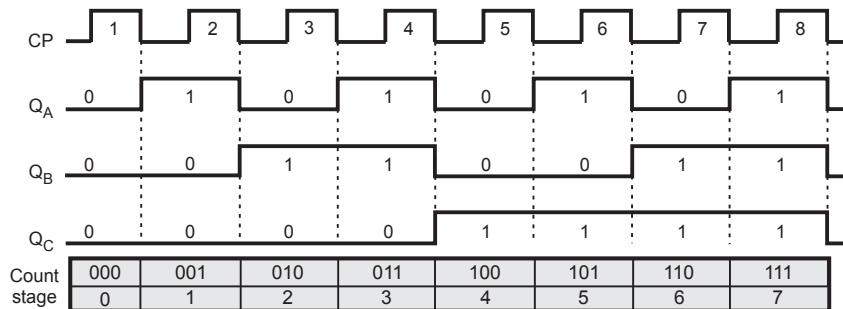


Fig. 6.3.4 Output waveforms for 3-bit asynchronous counter

Example 6.3.3 Design mod 6 ripple counter using T flip-flops.

Solution :

Step 1 : Determine the number of flip-flop required. Here, counter goes through 0 - 5 states, i.e., total 6 states. Thus $N = 6$ and for $2^n \geq N$ we need $n = 3$, i.e. 3 flip-flops.

Step 2 : Type of flip-flops to be used : T

Step 3 : Write the truth table for counter

CLK	C	B	A	Output of reset logic Y
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	1
-	1	1	0	0
-	1	1	1	0

Fig. 6.3.5

Step 4 : Derive reset logic

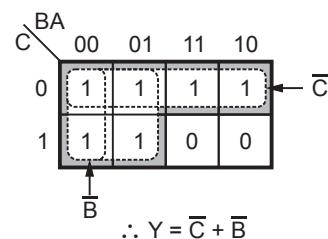


Fig. 6.3.6

Step 5 : Draw logic diagram

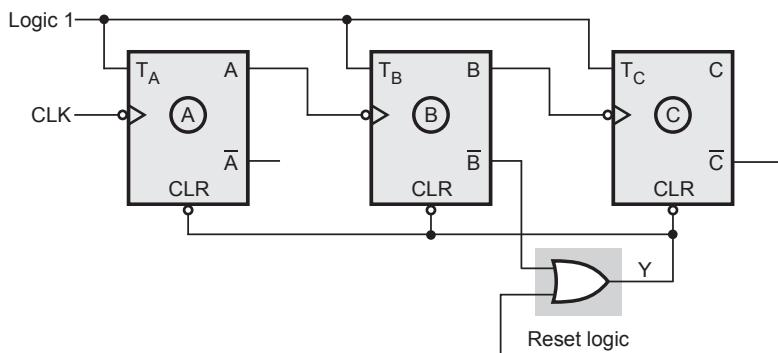


Fig. 6.3.7

Review Question

1. Design 3-bit down ripple counter. Draw waveforms.

SPPU : May-07, Marks 4

6.4 Synchronous Counters

SPPU : May-07,08,10, Dec.-10,12

When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called as synchronous counter.

6.4.1 2-bit Synchronous Binary Up Counter

Fig. 6.4.1 shows two stage synchronous counter.

Here, clock signal is connected in parallel to clock inputs of both the flip-flops. But the Q_A output of first stage is used to drive the J and K inputs of the second stage. Let us see the operation of the circuit. Initially, we assume

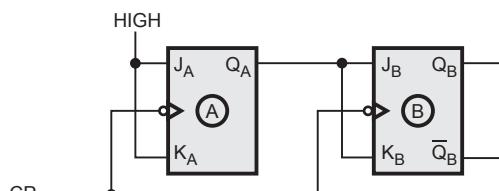


Fig. 6.4.1 A two-bit synchronous binary counter

that the $Q_A = Q_B = 0$. When positive edge of the first clock pulse is applied, flip-flop A will toggle because $J_A = K_A = 1$, whereas flip-flop B output will remain zero because $J_B = K_B = 0$. After first clock pulse $Q_A = 1$ and $Q_B = 0$. At negative going edge of the second clock pulse both flip-flops will toggle because they both have a toggle condition on their J and K inputs ($J_A = K_A = J_B = K_B = 1$). Thus after second clock pulse, $Q_A = 0$ and $Q_B = 1$. At negative going edge of the third clock pulse flip-flop A toggles making $Q_A = 1$, but flip-flop B remains set i.e. $Q_B = 1$. Finally, at the leading edge of the fourth clock pulse both flip-flops toggle as their JK inputs are at logic 1. This results $Q_A = Q_B = 0$ and counter recycled back to its original state. The timing details of above operation is shown in Fig. 6.4.2.

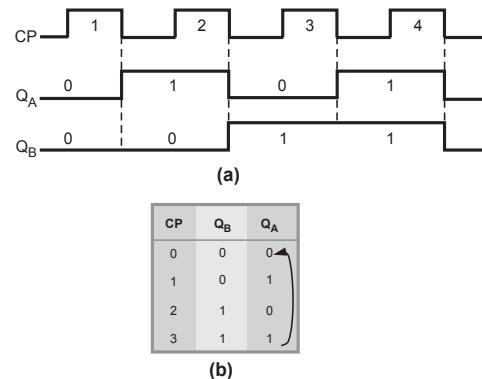


Fig. 6.4.2 Timing diagram and state sequence for the 2-bit synchronous counter

6.4.2 3-bit Synchronous Binary Up Counter

Fig. 6.4.3 (a) shows 3-bit synchronous binary counter and its timing diagram. The state sequence for this counter is shown in Table 6.4.1.

Looking at Fig. 6.4.3 (b), we can see that Q_A changes on each clock pulse as we progress from its original state to its final state and then back to its original state. To produce this operation, flip-flop A is held in the toggle mode by connecting J and K inputs to HIGH. Now let us see what flip-flop B does. Flip-flop B toggles, when Q_A is 1. When Q_A is a 0, flip-flop B is in the no-change mode and remains in its present state. Looking at the Table 6.4.1 we can notice that flip-flop C has to change its state only when

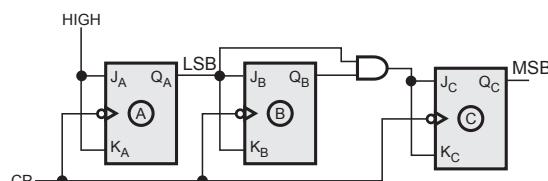


Fig. 6.4.3 (a) A three-bit synchronous binary counter

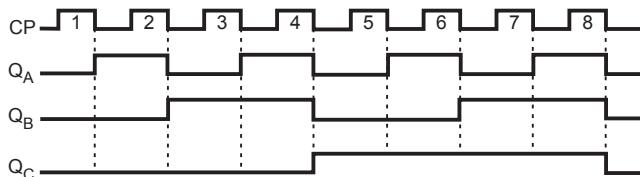


Fig. 6.4.3 (b) Timing diagram for 3-bit synchronous binary counter

CP	Q_C	Q_B	Q_A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table 6.4.1 State sequence for 3-bit binary counter

Q_B and Q_A both are at logic 1. This condition is detected by AND gate and applied to the J and K inputs of flip-flop C. Whenever both Q_A and Q_B are HIGH, the output of the AND gate makes the J and K inputs of flip-flop C HIGH and flip-flop C toggles on the following clock pulse. At all other times, the J and K inputs of flip-flop C are held LOW by the AND gate output and flip-flop does not change state.

6.4.3 4-bit Synchronous Binary Up Counter

Fig. 6.4.4 (a) shows logic diagram and timing diagram for 4-bit synchronous binary counter.

As counter is implemented with negative edge triggered flip-flops, the transitions occur at the negative edge of the clock pulse.

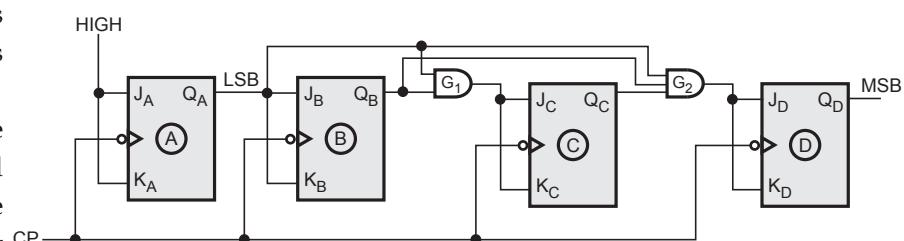


Fig. 6.4.4 (a)

In this circuit, first three flip-flops work same as 3-bit counter discussed previously.

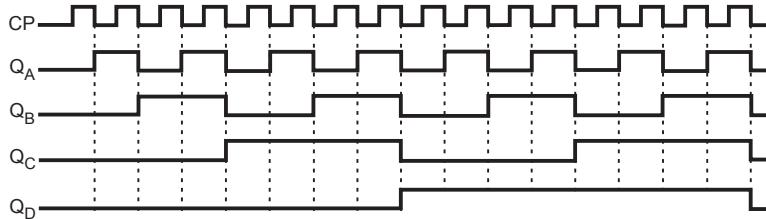


Fig. 6.4.4 (b) A four-bit synchronous binary counter and timing diagram

For the fourth stage, flip-flop has to change the state when $Q_A = Q_B = Q_C = 1$. This condition is decoded by 3-input AND gate G_2 . Therefore, when $Q_A = Q_B = Q_C = 1$, flip-flop D toggles and for all other times it is in no change condition.

Example 6.4.1 Determine f_{max} for the 4-bit synchronous counter if t_{pd} for each flip-flop is 50 ns and t_{pd} for each AND gate is 20 ns. Compare this with f_{max} for a MOD-16 ripple counter.

Solution : For a synchronous counter the total delay that must be allowed between input clock pulses is equal to flip-flop $t_{pd} + \text{AND gate } t_{pd}$. Thus $T_{clock} \geq 50 + 20 = 70$ ns and so the counter has

$$f_{max} = \frac{1}{70\text{ ns}} = 14.3 \text{ MHz}$$

We know that MOD-16 ripple counter used four flip-flops. With flip-flop $t_{pd} = 50$ ns, the f_{max} for ripple counter can be given as,

$$f_{max}(\text{ripple}) = \frac{1}{4 \times 50 \text{ ns}} = 5 \text{ MHz}$$

6.4.4 Synchronous Down and Up/Down Counters

SPPU : May-10, Marks 10

We have seen that a ripple counter could be made to count down by using the inverted output of each flip-flop to drive the next flip-flops in the counter. A parallel/synchronous down counter can be constructed in a similar manner that is, by using the inverted FF outputs to drive the following JK inputs. For example, the parallel up counter of Fig. 6.4.4 (a) can be converted to a down counter by connecting the \bar{Q}_A , \bar{Q}_B , \bar{Q}_C and \bar{Q}_D outputs in place of Q_A , Q_B , Q_C and Q_D respectively. The counter will then proceed through the following sequence as input pulses are applied :

To form a parallel up/down counter the control input ($\text{UP}/\overline{\text{DOWN}}$) is used to control whether the normal flip-flop outputs or the inverted flip-flop outputs are fed to the J and K inputs of the following flip-flops. The Fig. 6.4.5 shows 3-bit up/down counter that will count from 000 up to 111 when the $\text{Up}/\overline{\text{Down}}$ control input is 1 and from 111 down to 000 when the $\text{Up}/\overline{\text{Down}}$ control input is 0.

A logic 1 on the $\text{Up}/\overline{\text{Down}}$ enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the Q_A and Q_B outputs through to the J and K inputs of the next flip-flops so that the counter will count up as pulses are applied. When $\text{Up}/\overline{\text{Down}}$ line is logic 0, AND gates 1 and 2 are disabled and AND gates 3 and 4 are enabled. This allows the \bar{Q}_A and \bar{Q}_B outputs through to the J and K inputs of the next flip-flops so that the counter will count down as pulses are applied.

1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
.	.	.	.
.	.	.	.
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

Fig. 6.4.5

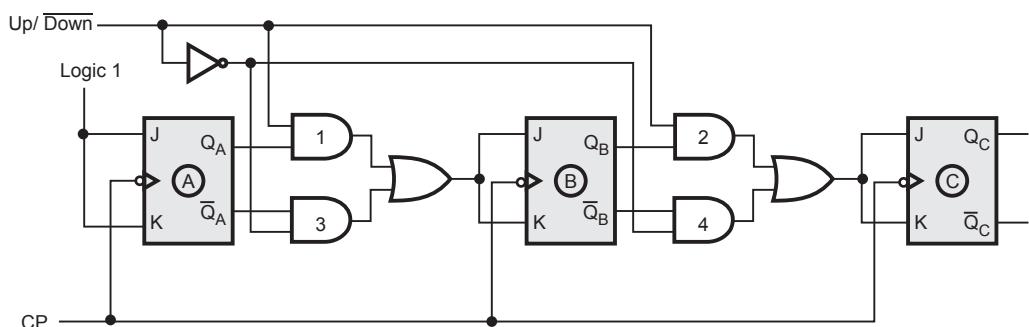


Fig. 6.4.6 3-bit synchronous/parallel up/down counter

Fig. 6.4.7 shows the timing diagram for 3-bit up-down counter.

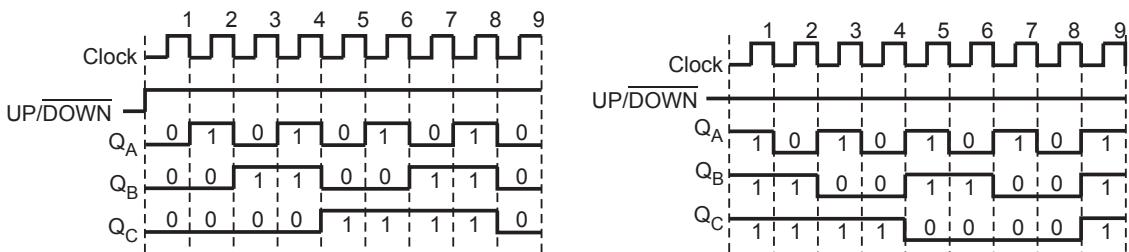


Fig. 6.4.7 Timing diagram

Review Questions

- What do you mean by ripple counter ? SPPU : May-08, Marks 2
- Draw and explain 3-bit asynchronous up counter using flip-flops and explain with output waveforms. SPPU : May-07, Dec.-12, Marks 6
- Draw and explain the working of 3-bit UP/DOWN synchronous counter.
- Draw a 4-bit synchronous counter. Also explain timing diagram for the same. SPPU : Dec.-10, Marks 10

6.5 Design of Synchronous Counters

SPPU : May-12,14,15,18, Dec.-13, 19

- Determine the number of flip-flops needed. If n represents number of flip-flops $2^n \geq$ number of states in the counter.
- Choose the type of flip-flops to be used.
- Using excitation table for selected flip-flop determine the excitation table for the counter.
- Use K-map or any other simplification method to derive the flip-flop input functions.
- Draw the logic diagram.

Example 6.5.1 Design a MOD-5 synchronous counter using JK flip-flops and implement it.
Also draw the timing diagram. SPPU : May-14, 15, Marks 6

Solution : **Step 1 :** Determine the number of flip-flop needed

Flip-flops required are

$$2^n \geq N$$

Here $N = 5 \therefore n = 3$ i.e. three flip-flops are required.

Step 2 : Type of flip-flop to be used : JK

Step 3 : Determine the excitation table for the counter.

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1

Table 6.5.1 Excitation table for JK flip-flop

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	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

Table 6.5.2 Excitation table for counter

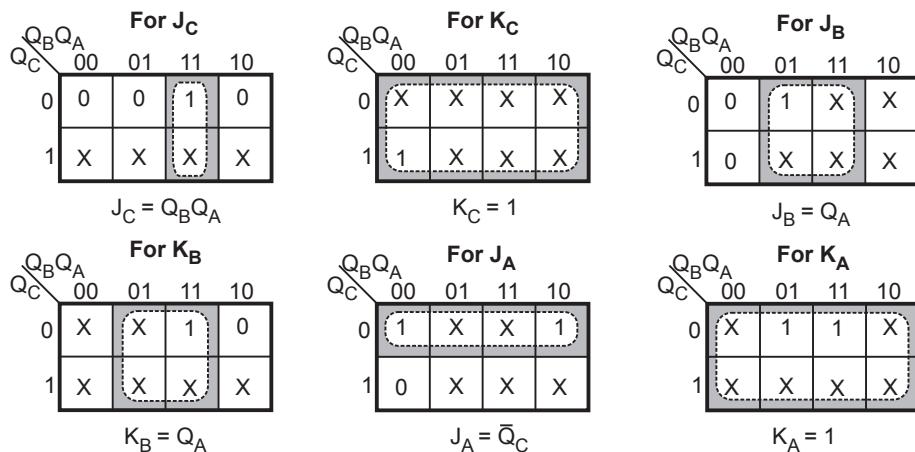
Step 4 : K-map simplification

Fig. 6.5.1

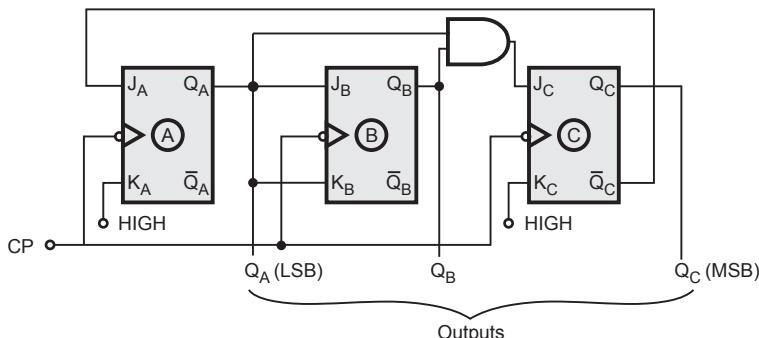
Step 5 : Draw the logic diagram

Fig. 6.5.2 (a) MOD-5 synchronous counter

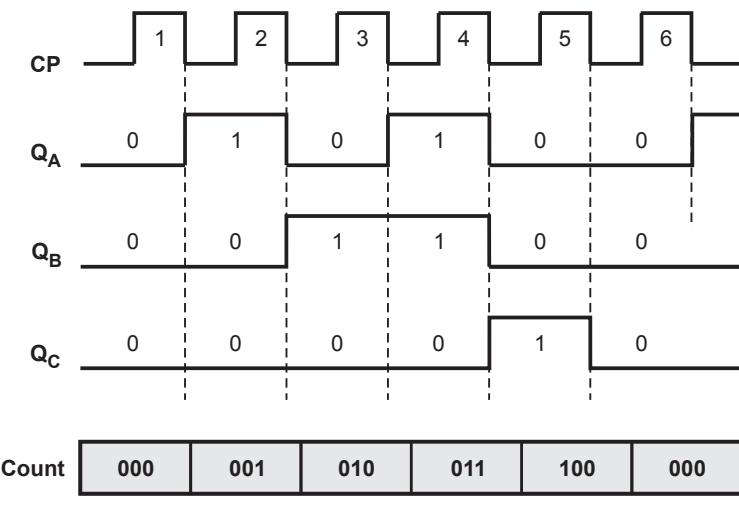
Timing Diagram

Fig. 6.5.2 (b)

Example 6.5.2 Design divide by 6 counter using T-flip-flops. Write state table and reduce the expression using K-map.

Solution :

Step 1 : Determine the number of flip-flops needed.

For designing mod 6 counter using the formula

$$2^n \geq N$$

Here $N = 6 \therefore n = 3$ i.e. 3 flip-flops are required.

Step 2 : Type of flip-flops to be used : T

Step 3 : Determine the excitation table for counter.

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 6.5.3 Excitation table for T-flip-flop

CP	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
1	0	0	1	0	1	0	0	1	1
2	0	1	0	0	1	1	0	0	1
3	0	1	1	1	0	0	1	1	1
4	1	0	0	1	0	1	0	0	1
5	1	0	1	0	0	0	1	0	1

Table 6.5.4 Excitation table for counter

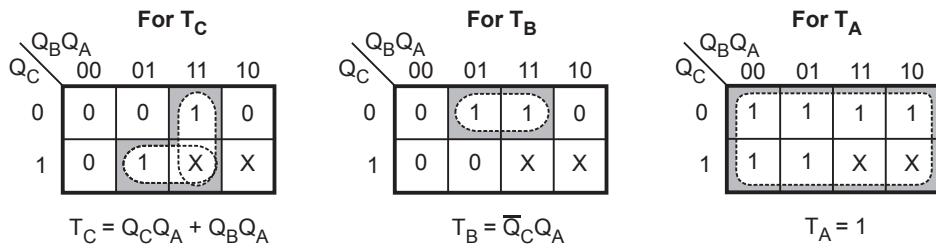
Step 4 : K-map simplification.

Fig. 6.5.3

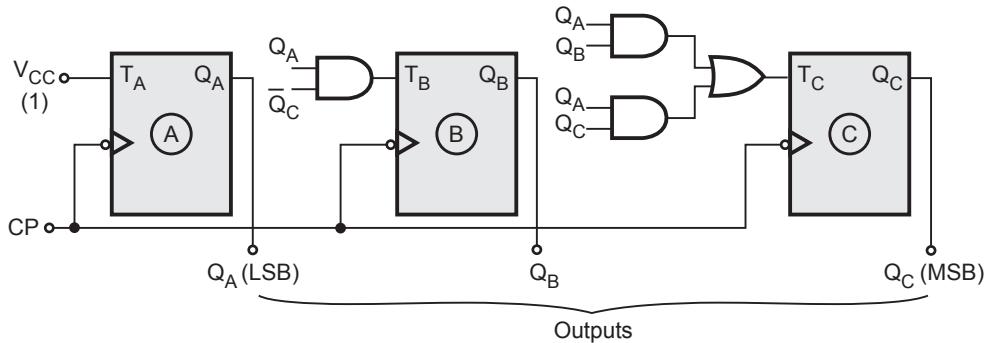
Step 5 : Draw the logic diagram.

Fig. 6.5.4 Logic diagram

Example 6.5.3 Design a synchronous decade counter using D flip-flop.

Solution : The decade counter is a mod-10 counter. It has ten states : 0 - 9.

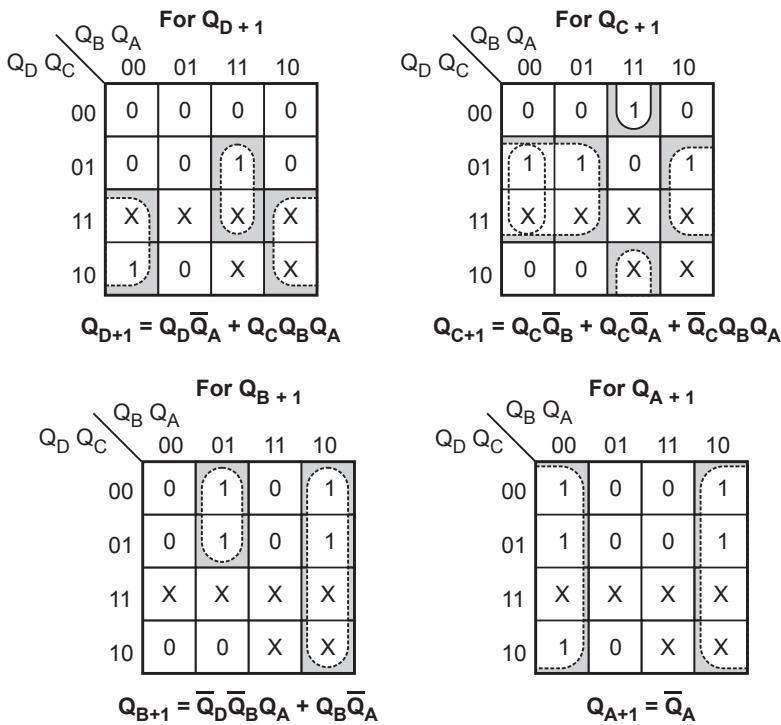
Step 1 : Determine the number of flip-flops needed.
We know that $2^n \geq N$.
Here, N = 10 $\therefore n = 4$
i.e. 4 flip-flops needed.

Step 2 : Types of flip-flops to be used : D

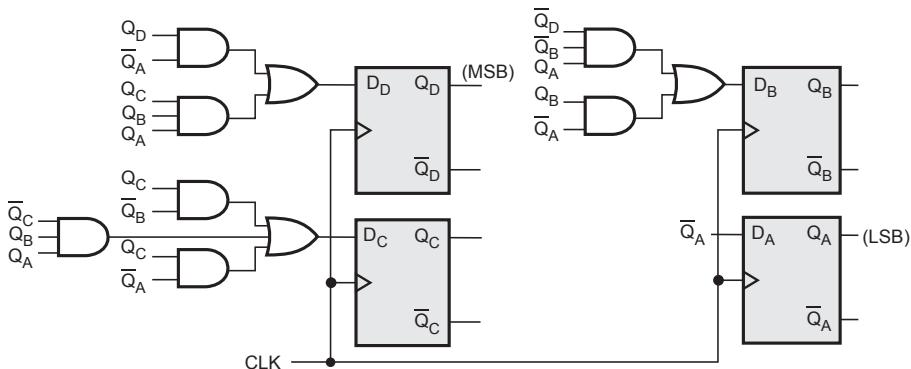
Step 3 : Determine the excitation table for counter.

Present state				Next state			
Q _D	Q _C	Q _B	Q _A	Q _D + 1	Q _C + 1	Q _B + 1	Q _A + 1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

Table 6.5.5 Excitation table for counter

Step 4 : K-map simplification**Fig. 6.5.5**

$$\begin{aligned}
 Q_{D+1} &= Q_D \bar{Q}_A + Q_C Q_B Q_A \\
 Q_{D+1} &= Q_D \bar{Q}_A + Q_C Q_B Q_A \\
 Q_{C+1} &= Q_C \bar{Q}_B + Q_C \bar{Q}_A + \bar{Q}_C Q_B Q_A \\
 Q_{B+1} &= \bar{Q}_D \bar{Q}_B Q_A + Q_B \bar{Q}_A \\
 Q_{A+1} &= \bar{Q}_A
 \end{aligned}$$

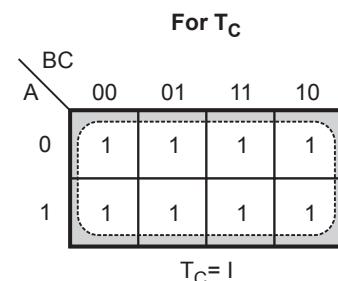
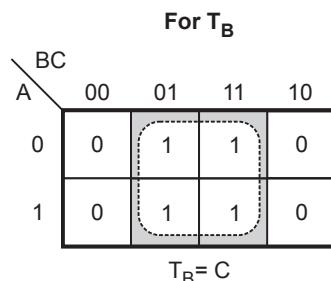
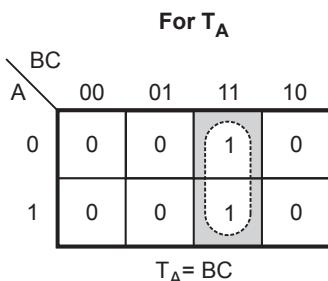
Step 5 : Draw the logic diagram.**Fig. 6.5.6 Logic diagram**

Example 6.5.4 Design 3-bit synchronous counter using T flip-flop. **SPPU : May-18, Marks 4**

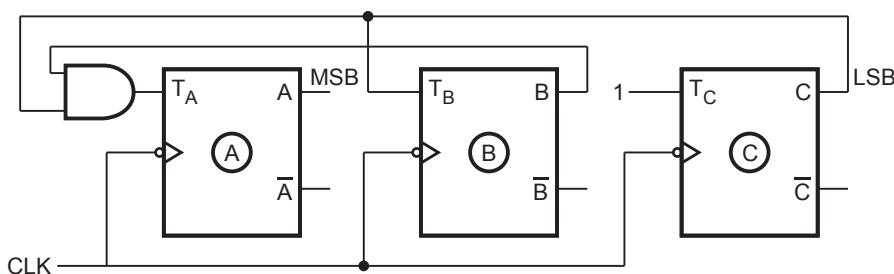
Solution :

Present state			Next state			Flip-flop inputs		
A	B	C	A^+	B^+	C^+	T_A	T_B	T_C
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

K-map simplification



Logic diagram



Example 6.5.5 Design 2 bit synchronous UP counter using MS-JK flip-flop. Draw the circuit diagram.

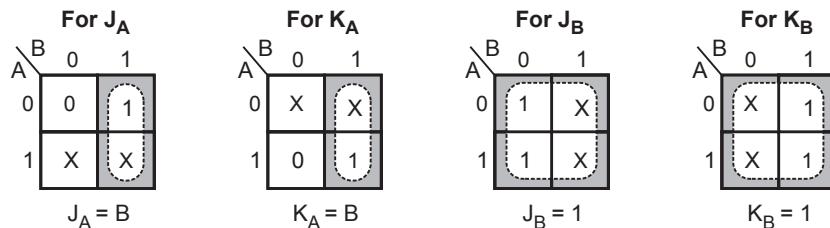
SPPU : Dec.-19, Marks 4

Solution :

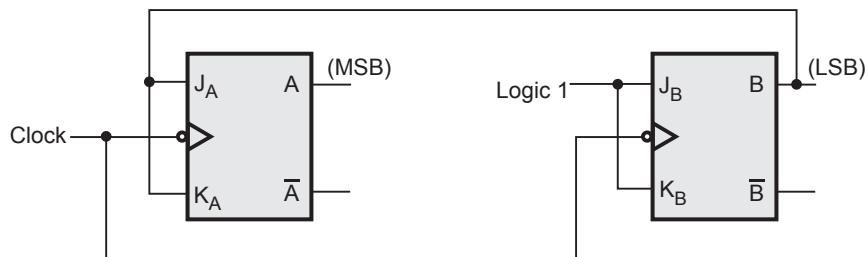
Excitation table :

Present state		Next state		Flip-flop inputs			
A	B	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

K-map simplification :



Logic diagram :



Examples for Practice

Example 6.5.7 : Design a 3 bit synchronous gray code counter using T flip-flop.

Example 6.5.8 : The following sequence is to be realized by a counter consisting of 3 JK FF's.

A ₁	0	0	0	0	1	1	0
A ₂	0	1	1	0	0	1	0
A ₃	0	1	0	1	1	0	0

Design the counter.

Example 6.5.9 : Design and explain the working of a mod-11 counter.

Example 6.5.10 : Design mod-6 synchronous counter using JK flip-flops and implement it.

Review Questions

1. Explain the steps in design of synchronous counter with the help of example.

2. Design 3-bit synchronous up-counter using MS JK flip-flops.

SPPU : May-12, Marks 6

3. Design Mod-5 synchronous counter using JK FFs.

SPPU : Dec.-13, Marks 4

4. Design a MOD-5 synchronous counter using JK FF and implement it. Also draw timing diagram.

SPPU : May-14, Marks 6

6.6 Lock-Out

SPPU : Dec.-11,15,18, May-13

In a counter if the next state of some unused state is again an unused state and if by chance the counter happens to find itself in the unused states and never arrived at a used state then the counter is said to be in the lockout conditions. This is illustrated in the Fig. 6.6.1. The counter which never goes in lockout condition is called **self starting counter**.

The circuit that goes in lockout condition is called **bushless circuit**.

To make sure that the

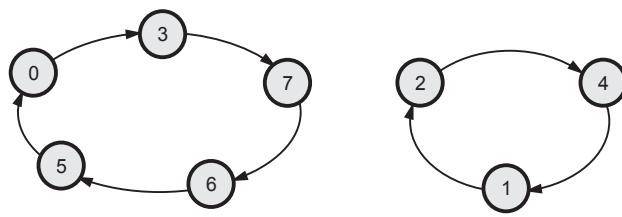


Fig. 6.6.1

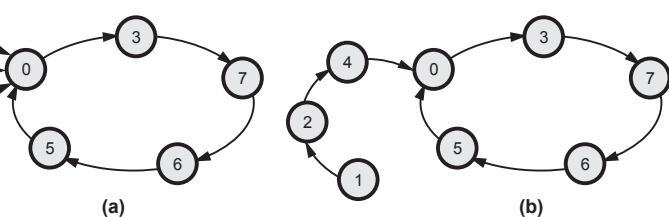


Fig. 6.6.2 State diagram for removing lockout

counter will come to the initial state from any unused state, the additional logic circuit is

necessary. To ensure that the lock out does not occur, the counter should be designed by forcing the next state to be the initial state from the unused states as shown in Fig. 6.6.2.

For example, as shown in Fig. 6.6.2, actually it is not necessary to force all unused states into initial state. Forcing any one state is sufficient. Because if counter initially goes to unused state which is not forced, it will go to another unused state. This will continue until it reaches the forced unused state. Once forced unused state is reached next state is used state, and circuit is lock free circuit. This is illustrated in Fig. 6.6.2 (b).

Example 6.6.1 Design a synchronous counter for

$$4 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 4 \dots$$

Avoid lockout condition. Use JK type design.

SPPU : Dec.-15, Marks 6

Solution : Step 1 : State diagram

Here, states 5, 2 and 0 are forced to go into 6, 3 and 1 state, respectively to avoid lockout condition.

Step 2 : Excitation table

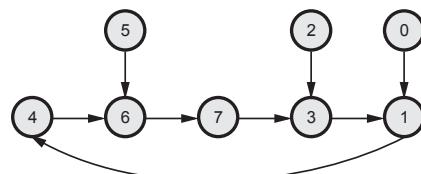


Fig. 6.6.3

Present states			Next states			Flip-flop inputs					
A	B	C	A_{+1}	B_{+1}	C_{+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	1	0	0	1	X	0	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	0	0	1	0	X	X	1	X	0
1	0	0	1	1	0	X	0	1	X	0	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	1	1	X	1	X	0	X	0

Table 6.6.1

Step 3 : K-map simplification

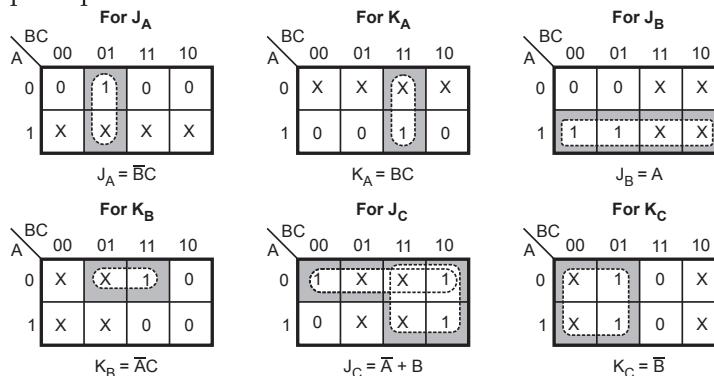


Fig. 6.6.4

Step 4 : Logic diagram

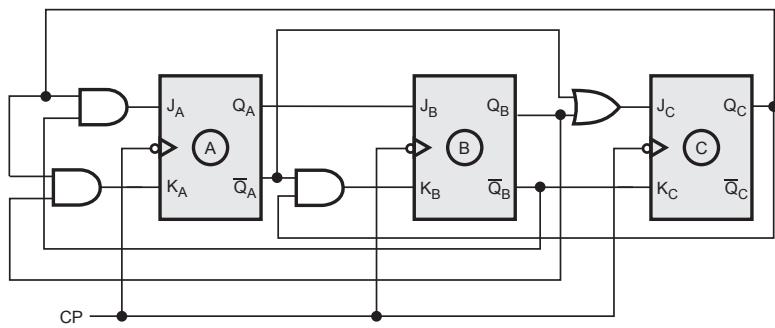


Fig. 6.6.5

Example 6.6.2 Design synchronous counter which will go through the following step, using JK flip-flop. (Avoid lock out condition.).

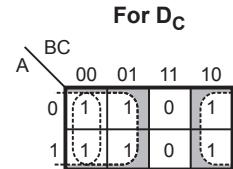
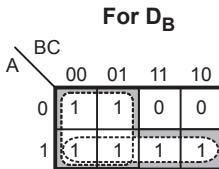
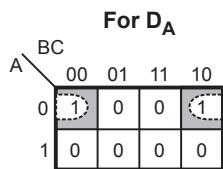
Solution : The Fig. 6.6.6 shows the state diagram for the given counter. To avoid lock-out condition states 1, 4 and 6 are forced to enter into state 3.

Flip-flop excitation table is as shown below.

Present state			Next state		
A	B	C	A_{+1}	B_{+1}	C_{+1}
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	1	0

Table 6.6.2

K-map simplification



$$D_A = \overline{A} \cdot \overline{C}$$

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

$$D_C = \overline{B} + \overline{C}$$

Fig. 6.6.7

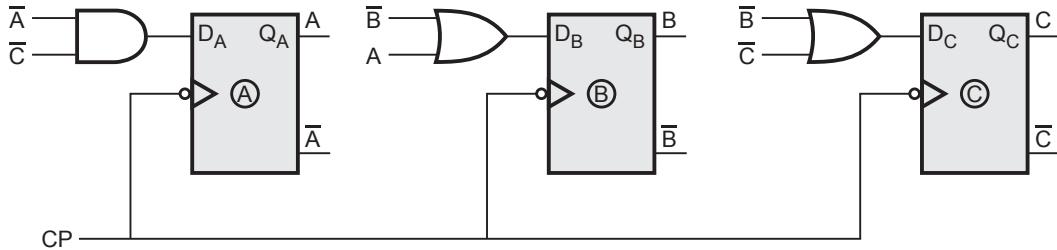
Logic diagram

Fig. 6.6.8

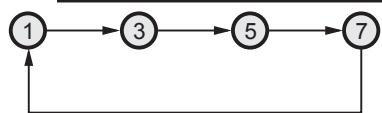
Review Questions

1. What is lockout condition ?
2. What is a self starting counter ?
3. How to avoid lockout condition.
4. Design sequence generator using JK FFs. Avoid lockout condition.

SPPU : May-13, Marks 6

SPPU : May-13, Marks 2

SPPU : Dec.-11,18, Marks 4

**6.7 IC 7490 (Decade Binary Counter)**

SPPU : Dec.-04,05,06,10,11,12,17, May-07,11,12,13,15,17

IC 7490 is a decade binary counter. It consists of four master-slave flip-flops and additional gating to provide a divide-by-two counter and a three stage binary counter for which the count cycle length is divide by five.

Since the output from the divide-by-two section is not internally connected to the succeeding stages, the devices may be operated in various counting modes.

1. BCD Decade (8421) Counter : The B input must be externally connected to the Q_A output and A input receives the incoming count and a BCD count sequence is produced.

2. Symmetrical Bi-quinary Divide-by-Ten Counter : The Q_D output must be externally connected to the A input. The input count is then applied to the B input and a divide-by-ten square wave is obtained at output Q_A .

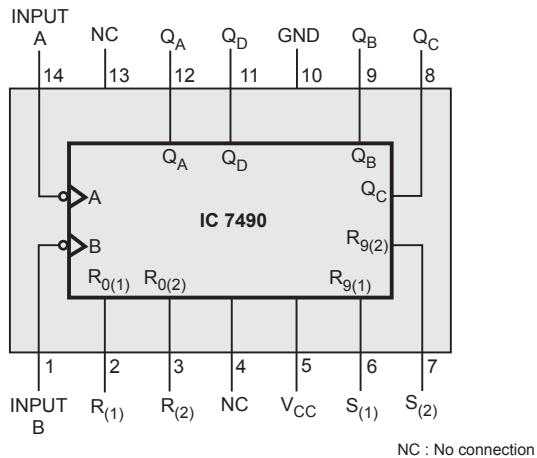


Fig. 6.7.1 Connection diagram for 7490

3. Divide-by-Two and Divide-by-Five Counter : No external interconnections are required. The first flip-flop is used as a binary element for the divide-by-two function (A as the input and Q_A as the output). The B input is used to obtain binary divide-by-five operation at the Q_D output.

Table 6.7.1 shows function tables and Fig. 6.7.2 shows logic diagram for IC7490.

Count	Outputs			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H

Table 6.7.1 (a) BCD count sequences
(Note 1)

Count	Outputs			
	Q_A	Q_D	Q_C	Q_B
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

Table 6.7.1 (b) BCD Bi-quinary (5-2) (Note 2)

Reset Inputs				Outputs			
$R_{0(1)}$	$R_{0(2)}$	$R_{9(1)}$	$R_{9(2)}$	Q_D	Q_C	Q_B	Q_A
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	COUNT			
L	X	L	X	COUNT			
L	X	X	L	COUNT			
X	L	L	X	COUNT			

Table 6.7.1 (c) Reset/Count function table

H : HIGH Level

L : LOW Level

X : Don't Care

Note 1 : Output Q_A is connected to input B for BCD count.

Note 2 : Output Q_D is connected to input A for bi-quinary count.

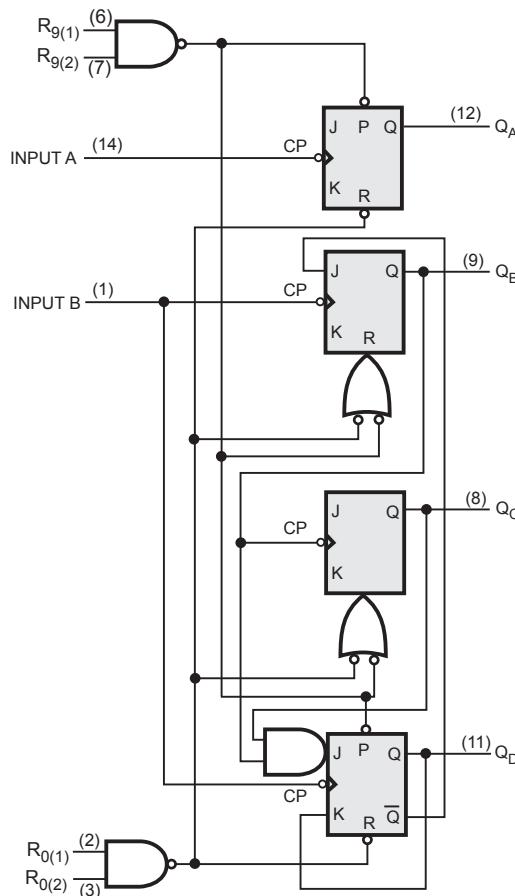


Fig. 6.7.2 Internal block diagram of 7490

Example 6.7.1 Draw basic internal architecture of IC 7490. Design a divide-by-20 counter using IC 7490.

SPPU : May-15, Dec.-11, Marks 8

Solution : Internal structure of 7490 ripple counter IC is as shown in Fig. 6.7.3.

We know that, one IC can work as mod-10 (BCD) counter. Therefore we need two ICs. The counter will go through states 0-19 and should be reset on state 20. i.e.

Q _D	Q _C	Q _B	Q _A	Q _D	Q _C	Q _B	Q _A
0	0	1	0	0	0	0	0

$\underbrace{7490 \text{ (2)}}_{\text{7490 (1)}}$

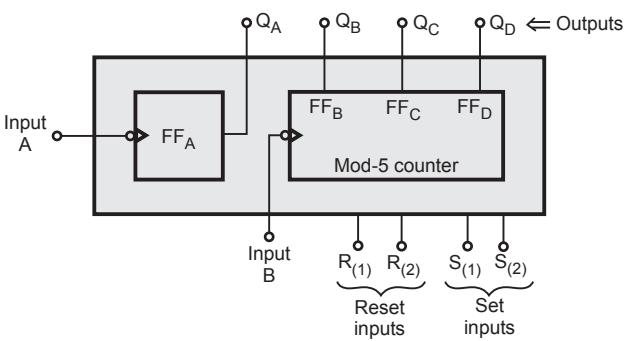


Fig. 6.7.3 Basic internal structure of IC 7490

The diagram of divide-by-20 counter using IC 7490 is as shown in Fig. 6.7.4.

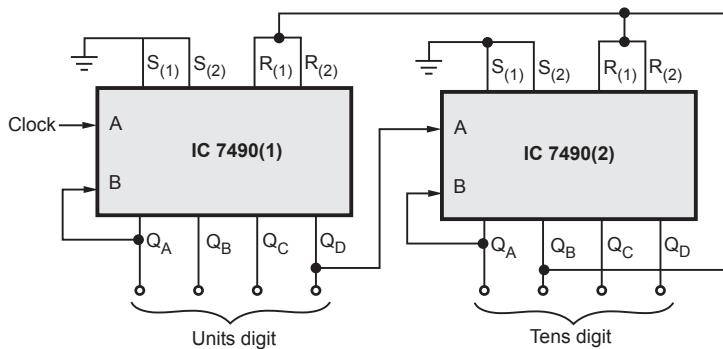


Fig. 6.7.4 Divide-by-20 counter using IC 7490

Example 6.7.2 Design a divide-by-96 counter using 7490 ICs.

SPPU : Dec.-05, Marks 6

Solution : IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide by 100 counter. To get a divide-by-96 counter, the counter is reset as soon as it becomes 1001 0110. The diagram is shown in Fig. 6.7.5.

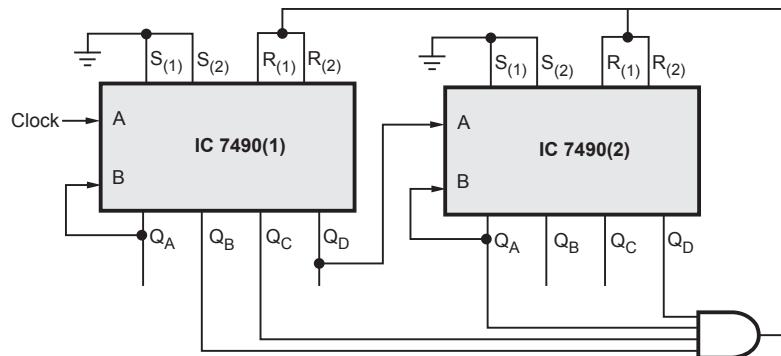


Fig. 6.7.5 Divide-by-96 counter

Example 6.7.3 Design divide-by-93 counter using the same IC.

Solution : IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide-by-100 counter. To get a divide-by-93 counter, the counter is reset as soon as it becomes 1001 0011. The diagram is shown in the Fig. 6.7.6.

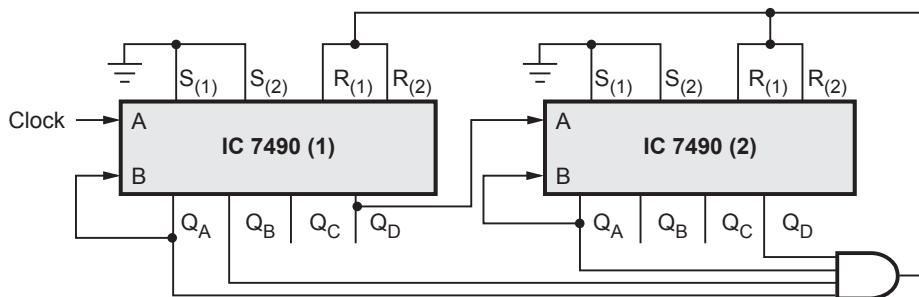


Fig. 6.7.6 Divide-by-93 counter

Example 6.7.4 Design MOD-78 counter using IC 7490.

SPPU : Dec.-06, Marks 6

Solution : IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide-by-100 counter. To get a divide by 78 or MOD-78 counter, the counter is reset as soon as it becomes 0111 1000 as shown in Fig. 6.7.7.

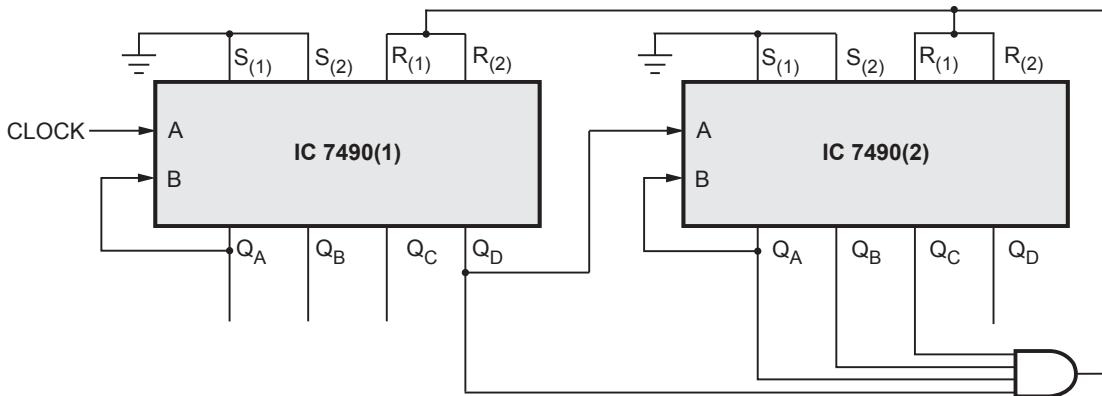


Fig. 6.7.7

Example 6.7.5 Design Mod 8 counter using 7490.

SPPU : May-07, Marks 6

Solution : MOD 8 ripple counter using 7490 :

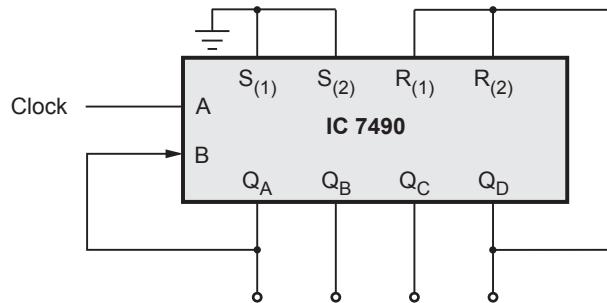


Fig. 6.7.8

Example 6.7.6 In 7490, if Q_D output is connected to A input and pulses at B input. Find count sequence and waveform of output Q_A.

Solution : If the Q_D output is connected to A input of 7490 IC and, input clock is applied to B input divide by ten square wave is obtained at output Q_A.

Clock	Outputs			
	Q _A	Q _D	Q _C	Q _B
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H

4	L	H	L	L
5	H	L	L	L
6	H	L	L	H
7	H	L	H	L
8	H	L	H	H
9	H	H	L	L

Table 6.7.2

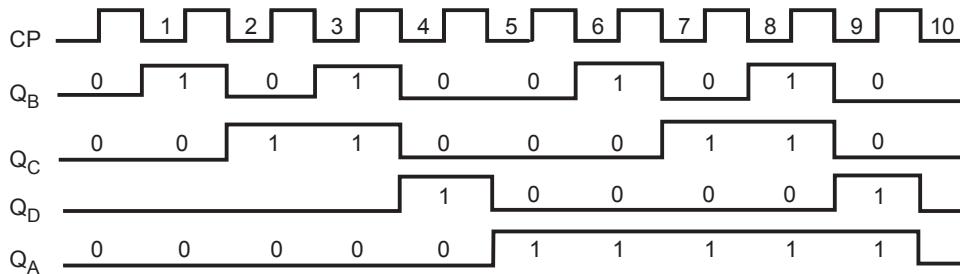


Fig. 6.7.9 Waveforms

Example 6.7.7 Design a mod 25 counter using IC 7490.

Solution : IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide-by-100 counter. To get a divide-by-25 counter, the counter is reset as soon as it becomes 0010 0101. The diagram is shown in Fig. 6.7.10.

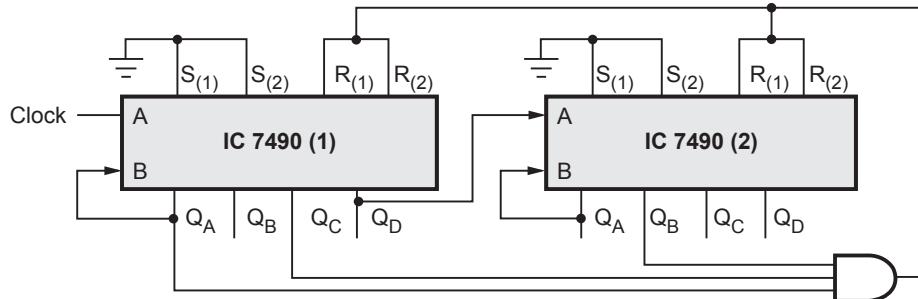


Fig. 6.7.10

Example 6.7.8 Design Mod - 24 counter using 7490.

SPPU : May-17, Marks 2

Solution : IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide-by-100 counter. To get a divide-by-24 counter, the counter is reset as soon as it becomes 0010 0100. The diagram is shown in Fig. 6.7.11.

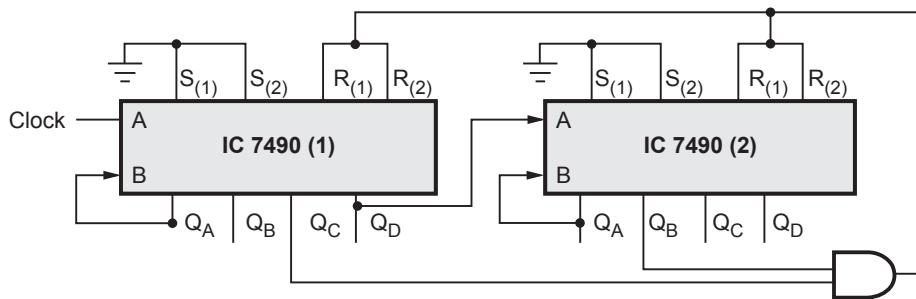


Fig. 6.7.11

Review Questions

1. Draw basic internal architecture of IC 7490 and explain its operation.

SPPU : Dec.-04,11,17, May-13 Marks 4

2. Draw internal block diagram of 7490. Mention how to use it for designing decade-counter.

SPPU : May-07, Marks 4

3. What is advantage of MOD counter ? Explain working of MOD-17 and MOD-24 counter with detail diagram using IC-7490.

SPPU : Dec.-10, Marks 8

4. What is advantage of MOD counter ? Explain working of MOD-27 and MOD-13 counter with detail diagram.

SPPU : May-11, Marks 10

5. Explain the internal diagram of IC 7490. Design MOD 7 and MOD 98 counter using 7490.

SPPU : May-12, Marks 8

6. Design the following using IC7490 :

- i) MOD 97 counter ii) MOD 45 counter.

SPPU : Dec.-12, Marks 8

7. Design MOD 56 counter using IC 7490 & necessary logic gates.

SPPU : May-13, Marks 4

6.8 Ring Counter

SPPU : Dec.-08,10,19, May-11

Fig. 6.8.1 shows the logic diagram for four-bit ring counter. As shown in the Fig. 6.8.1, the Q output of each stage is connected to the D input of the next stage and

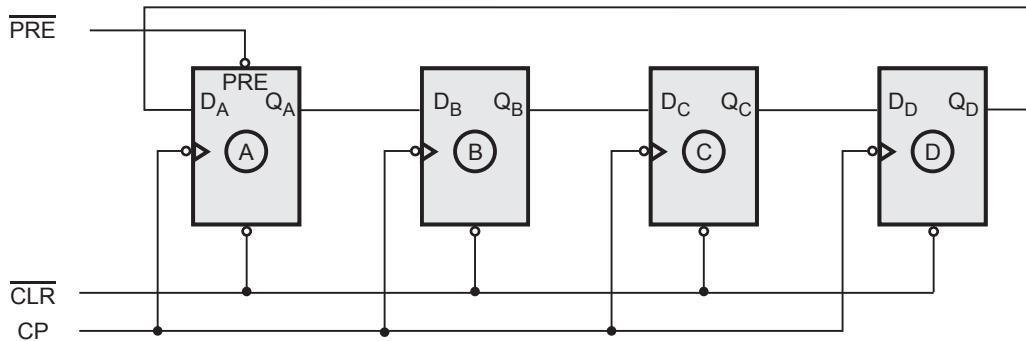


Fig. 6.8.1 Four-bit ring counter

the output of last stage is fed back to the input of first stage. The $\overline{\text{CLR}}$ followed by $\overline{\text{PRE}}$ makes the output of first stage to '1' and remaining outputs are zero, i.e. Q_A is one and Q_B, Q_C, Q_D are zero.

The first clock pulse produces $Q_B = 1$ and remaining outputs are zero. According to the clock pulses applied at the clock input CP, a sequence of four states is produced. These states are listed in Table 6.8.1.

As shown in Table 6.8.1, 1 is always retained in the counter and simply shifted 'around the ring', advancing one stage for each clock pulse. In this case four stages of flip-flops are used. So a sequence of four states is produced and repeated. Fig. 6.8.2 gives the timing sequence for a four-bit ring counter.

The ring counter can be used for counting the number of pulses. The number of pulses counted is read by noting which flip-flop is in state 1. No decoding circuitry is required. Since there is one pulse at the output for each of the N clock pulses, this circuit is also referred to as a divide-by-N-counter or an $N : 1$ scalar. Ring counters can be instructed for any desired MOD number, that is MOD-N ring counter requires N flip-flops.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Table 6.8.1 Ring counter sequence 4-bits

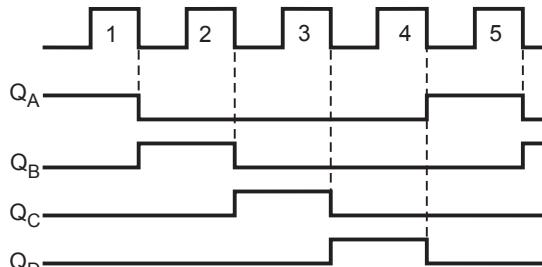


Fig. 6.8.2 Timing sequence for a four-bit ring counter

Review Questions

1. Draw and explain the working of 4-bit ring counter using D flip-flop.

SPPU : Dec.-19, Marks 6

2. Write a short note on ring counter.

SPPU : Dec.-08, Marks 3

3. Explain ring counter with design having initial state '01011', from initial state explain all possible states in that ring.

SPPU : Dec.-10, Marks 10

4. Explain ring counter design for the initial condition '10110' also explain twisted ring counter in brief.

SPPU : May-11, Marks 8

6.9 Johnson or Twisting Ring or Switch Tail Counter

SPPU : Dec.-08,17,18

In a Johnson counter, the Q output of each stage of flip-flop is connected to the D input of the next stage. The single exception is that the complement output of the last flip-flop is connected back to the D-input of the first flip-flop as shown in Fig. 6.9.1.

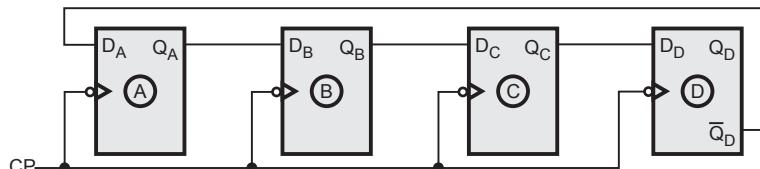


Fig. 6.9.1 Four-bit Johnson counter

Note Johnson counter can be implemented with SR or JK flip-flops as well.

As shown in Fig. 6.9.1 there is a feedback from the rightmost flip-flop complement output to the leftmost flip-flop input. This arrangement produces a unique sequence of states.

Initially, the register (all flip-flops) is cleared. So all the outputs, Q_A, Q_B, Q_C, Q_D are zero. The output of last stage, Q_D is zero. Therefore complement output of last stage, \bar{Q}_D is one. This is connected back to the D input of first stage. So D_A is one. The first falling clock edge produces $Q_A = 1$ and $Q_B = 0, Q_C = 0, Q_D = 0$ since D_B, D_C, D_D are zero. The next clock pulse produces $Q_A = 1, Q_B = 1, Q_C = 0, Q_D = 0$. The sequence of states is summarized in Table 6.9.1. After 8 states the same sequence is repeated.

In this case, four-bit register is used. So the four-bit sequence has a total of eight states. Fig. 6.9.2 gives the timing sequence for a four-bit Johnson counter.

If we design a counter of five-bit sequence, it has a total of ten states, as shown in Table 6.9.2.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 6.9.1 Four-bit Johnson sequence

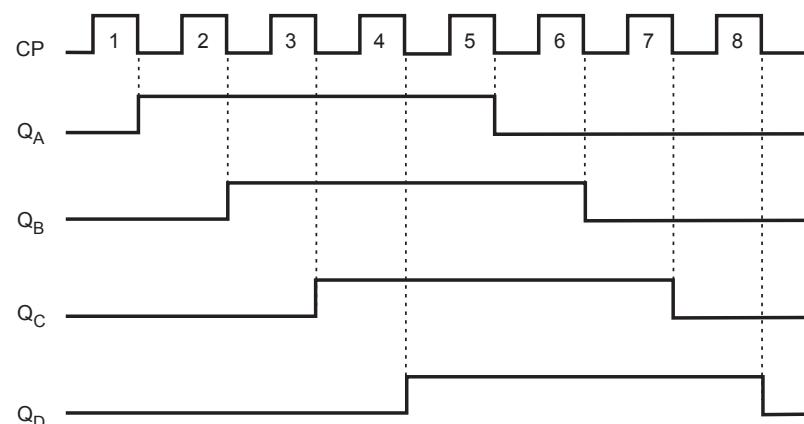


Fig. 6.9.2 Timing sequence for a four-bit Johnson counter

So in general we can say that, an n-stage Johnson counter will produce a modulus of $2 \times n$, where n is the number of stages (i.e. flip-flops) in the counter. Thus, Johnson counter requires only half the number of flip-flops compared to the standard ring counter. However, it requires more flip-flop than binary counter. As shown in tables, the counter will 'fill up' with 1s from left to right and then it will 'fill up' with 0s again. Another advantage of this type of sequence is that it is readily decoded with two input AND gates. Table 6.9.3 gives the count sequence and required decoding.

Clock pulse	Q_A	Q_B	Q_C	Q_D	Q_E
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

Table 6.9.2 Five-bit Johnson sequence

Clock pulse	Q_A	Q_B	Q_C	Q_D	AND Gate required for output
0	0	0	0	0	$\bar{Q}_A \bar{Q}_D$
1	1	0	0	0	$Q_A \bar{Q}_B$
2	1	1	1	0	$Q_B \bar{Q}_C$
3	1	1	1	1	$Q_C \bar{Q}_D$
4	1	1	1	1	$Q_A Q_D$
5	0	1	1	1	$\bar{Q}_A Q_B$
6	0	0	1	1	$\bar{Q}_B Q_C$
7	0	0	0	1	$\bar{Q}_C Q_D$

Table 6.9.3 Count sequence and required decoding

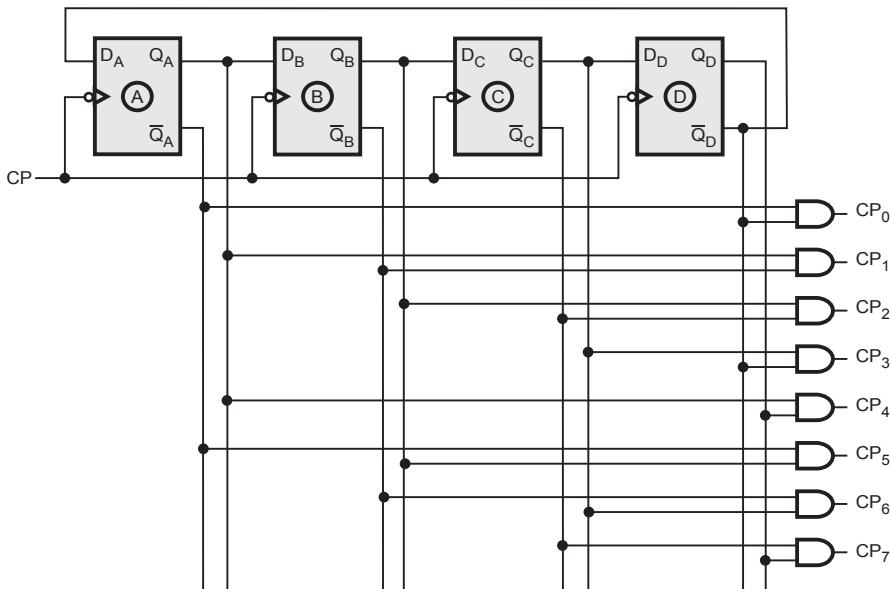


Fig. 6.9.3 Johnson counter with decoder

Example 6.9.1 Draw and explain ring counter using JK flip-flop (timing diagram is expected)

SPPU : Dec.-18, Marks 4

Solution : Fig. 6.9.4 shows the logic diagram for four bit ring counter. As shown in the Fig. 6.9.4, the Q output of each stage is connected to the J input of the next stage and the same signal after inversion connected to the K-input of the next stage. The output of the last stage is fed back to inputs of the first stage. The $\overline{\text{CLR}}$ followed by $\overline{\text{PRE}}$ makes the output of first stage to '1' and remaining outputs are zero, i.e. Q_A is one and Q_B, Q_C, Q_D are zero.

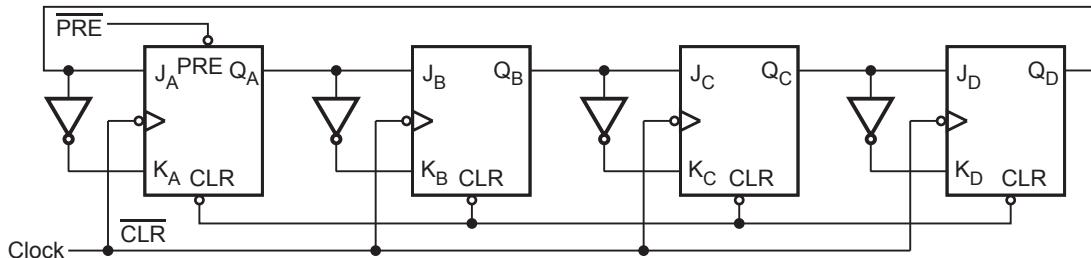


Fig. 6.9.4 Four-bit ring counter

The first clock pulse produces $Q_B = 1$ and remaining outputs are zero. According to the clock pulses applied at the clock input CP, a sequence of four states is produced. These states are listed in Table 6.9.4.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Table 6.9.4

As shown in Table 6.9.4, 1 is always retained in the counter and simply shifted 'around the ring', advancing one stage for each clock pulse. In this case four stages of flip-flops are used. So a sequence of four states is produced and repeated. Fig. 6.9.5 gives the timing sequence for a four-bit ring counter.

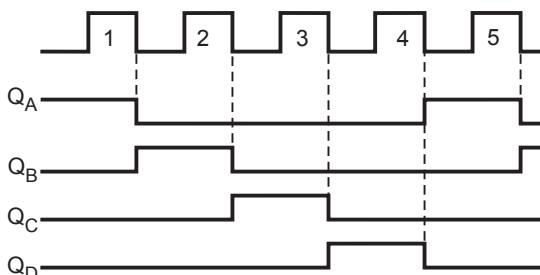


Fig. 6.9.5 Timing sequence for a four-bit ring counter

Review Questions

1. What is the difference between ring counter and Johnson's counter ?
2. Draw the neat circuit diagram of 3-bit Johnson's counter. Draw the relevant output waveforms.
3. Write a short note on Johnson's counter.

SPPU : Dec.-08,17, Marks 3

Notes

UNIT - III

7

Synchronous Sequential Circuit Design

Syllabus

Models-Moore and Mealy, State diagram and State Table, Design Procedure, Sequence generator and detector.

Contents

7.1	<i>Clocked Sequential Circuits</i>	<i>May-10,12,13,</i>
		<i>Dec.-10,12,16</i> Marks 4
7.2	<i>Design of Clocked Sequential Circuits</i>	<i>Dec.-05,07,08,10,12,13,14,</i>
		<i>May-05,06,11,12,13,</i> Marks 10
7.3	<i>Sequence Generator</i>	<i>May-2000,02,06,08,12,18,</i>
		<i>Dec.-06,07,08,12,13</i> Marks 8
7.4	<i>Sequence Detector</i>	<i>May-06,07,10,12,13,14,</i>
		<i>Dec.-05,06,07,08,10,12,13,</i>
		<i>15,16,</i> Marks 12

7.1 Clocked Sequential Circuits

SPPU : May-10,12,13, Dec.-10,12,16

In synchronous or clocked sequential circuits, clocked flip-flops are used as memory elements, which change their individual states in synchronism with the periodic clock signal. Therefore, the change in states of flip-flops and change in state of the entire circuit occurs at the transition of the clock signal. The states of the output of the flip-flop in the sequential circuit gives the state of the sequential circuit.

Present state : The status of all state variables, at some time t , before the next clock edge, represents condition called **present state**.

Next state : The status of all state variables, at some time, $t + 1$, represents a condition called **next state**.

The synchronous or clocked sequential circuits are represented by two models.

- **Moore model :** The output depends only on the present state of the flip-flops.
- **Mealy model :** The output depends on both the present state of the flip-flop(s) and on the input (s).

7.1.1 Moore Model

As mentioned earlier, when the output of the sequential circuit depends only on the present state of the flip-flop, the sequential circuit is referred to as **Moore model**. Let us see one example of Moore model.

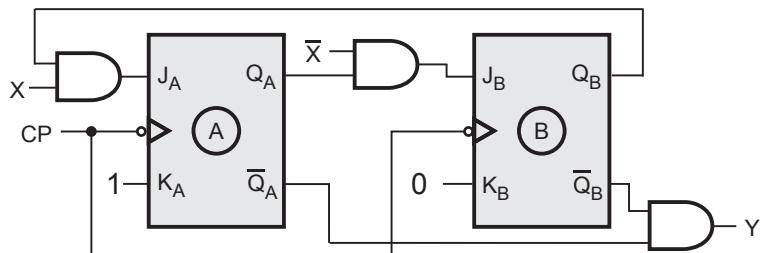


Fig. 7.1.1 Example of Moore model

Fig. 7.1.1 shows a sequential circuit which consists of two JK flip-flops and AND gate. The circuit has one input X and one output Y.

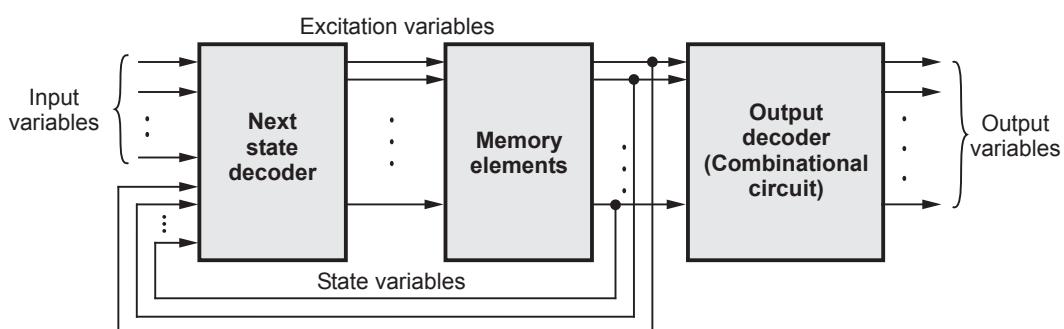


Fig. 7.1.2 Moore circuit model with an output decoder

As shown in the Fig. 7.1.1, input is used to determine the inputs of the flip-flops. It is not used to determine the output. The output is derived using only present states of the flip-flops or combination of it (in this case $Y = Q_A Q_B$).

In general form the Moore model can be represented with its block schematic as shown in Fig. 7.1.2.

7.1.2 Mealy Model

When the output of the sequential circuit depends on both the present state of flip-flop(s) and on the input(s), the sequential circuit is referred to as **Mealy model**. Fig. 7.1.3 shows the sample Mealy model. As shown in the Fig. 7.1.3, the output of the circuit is derived from the combination of present state of flip-flops and input (s) of the circuit.

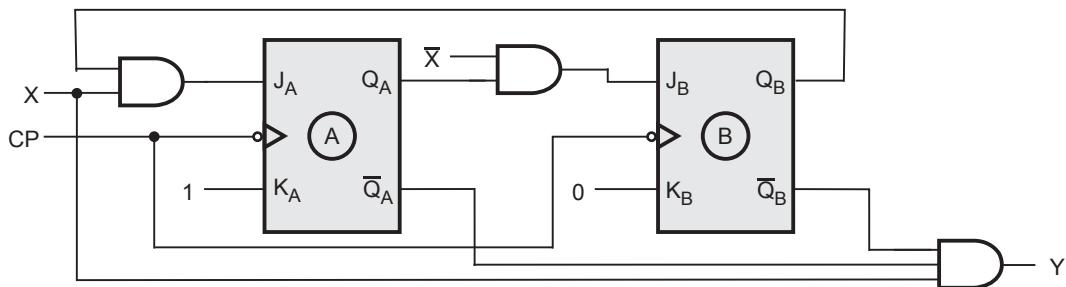


Fig. 7.1.3 Example of Mealy model

Looking at Fig. 7.1.3, we can easily realize that changes in the input within the clock pulses can not affect the state of the flip-flop. However, they can affect the output of the circuit. Due to this, if the input variations are not synchronized with the clock, the derived output will also not be synchronized with the clock and we get false output (as it is a synchronous sequential circuit). The false outputs can be eliminated by allowing input to change only at the active transition of the clock (in our example HIGH-to-LOW). In general form the Mealy model can be represented with its block schematic as shown in Fig. 7.1.4.

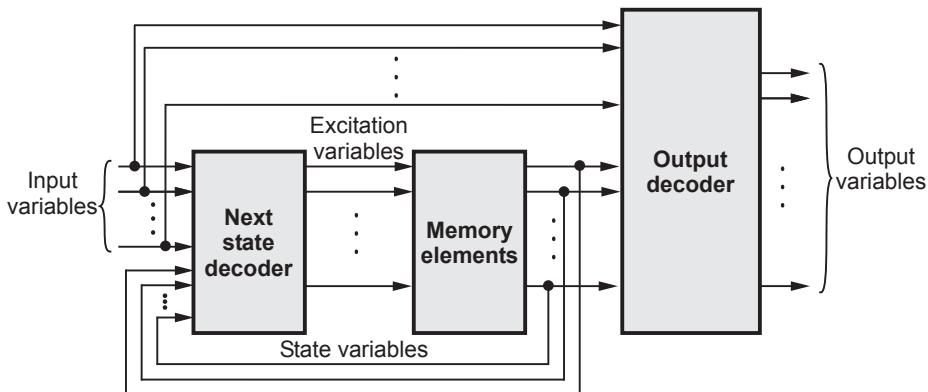


Fig. 7.1.4 Mealy circuit model

7.1.3 Moore vs Mealy Circuit Models

Sr. No.	Moore model	Mealy model
1.	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2.	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3.	Moore model requires more number of states for implementing same function.	It requires less number of states for implementing same function.

Table 7.1.1

7.1.4 Representations of Sequential Circuits

7.1.4.1 State Diagram

State diagram is a pictorial representation of a behaviour of a sequential circuit. Fig. 7.1.5 shows a state diagram. The state is represented by the circle, and the transition between states is indicated by directed lines connecting the circles. A directed line connecting a circle with itself indicates that next state is same as present state. The binary number inside each circle identifies the state represented by the circle. The directed lines are labelled with two binary numbers separated by a symbol '/'. The input value that causes the state transition is labelled first and the output value during the present state is labelled after the symbol '/'.

In case of Moore circuit, the directed lines are labelled with only one binary number representing the state of the input that causes the state transition. The output state is indicated within the circle, below the present state because output state depends only on present state and not on the input. Fig. 7.1.6 shows the state diagram for Moore circuit.

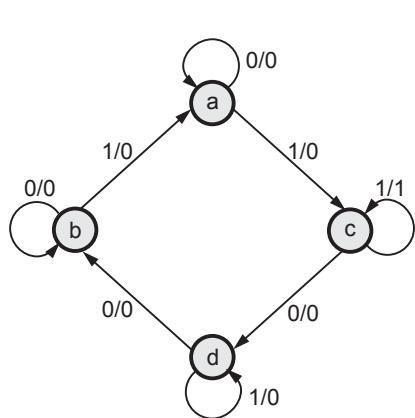


Fig. 7.1.5 State diagram for Mealy circuit

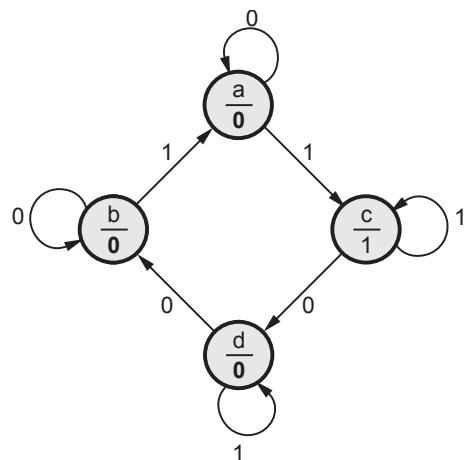


Fig. 7.1.6 State diagram for Moore circuit

7.1.4.2 State Table

Although the state diagram provides a description of the behaviour of a sequential circuit that is easy to understand, to proceed with the implementation of the circuit, it is convenient to translate the information contained in the state diagram into a tabular form called **state synthesis table** or simply **state table**. Table 7.1.2 (a) shows the state table for the state diagram shown in Fig. 7.1.5. It represents relationship between input, output and flip-flop states. It consists of three sections labeled present state, next state, and output. The present state designates the state of flip-flops before the occurrence of a clock pulse. The next state is state of the flip-flop after the application of a clock pulse, and the output section gives the values of the output variables during the present state. Both the next state and output sections have two columns representing two possible input conditions : $X = 0$ and $X = 1$.

Present state	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
AB	AB	AB	Y	Y
a	a	c	0	0
b	b	a	0	0
c	d	c	0	1
d	b	d	0	0

Table 7.1.2 (a)

Present state	Next state		Output
	$X = 0$	$X = 1$	
AB	AB	AB	
a	a	c	0
b	b	a	0
c	d	c	1

Table 7.1.2 (b)

In case of Moore circuit the output section has only one column since output does not depend on input. The Table 7.2.2 (b) shows the state table for Moore circuit whose state diagram is shown in Fig. 7.1.6.

7.1.4.3 Transition Table

A transition table takes the state table one step further. The state diagram and state table represent state using symbols or names. In the transition table specific state variable values are assigned to each state. Assignment of values to state variables is called **State assignment**. Like state table transition table also represents relationship between input, output and flip-flop states. The Fig. 7.1.7 shows the transition table.

Present state	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
A B	AB	AB	Y	Y
0 0	0 0	1 0	0	1
0 1	1 1	0 0	0	0
1 0	1 0	0 1	1	0
1 1	0 0	1 0	1	0

Fig. 7.1.7

Here, AB are the state variables. The AB = 00 represents one state, AB = 01 represents second state and so on.

Example 7.1.1 What does the word 'Finite' signify in the terms finite state machine ? State advantages and disadvantages of a finite state machine. SPPU : May-10, Marks 4

Solution : The word 'Finite' signifies a finite or limited number of possible states in the terms finite state machine (FSM).

Advantages of finite state machine

- Simple to understand
- Predictable for given a set of inputs and a known current state, the state transition can be predicted, allowing for easy testing.
- Due to their simplicity, FSMs are quick to design, quick to implement and quick in execution.
- FSMs are relatively flexible.
- Easy to transfer from a meaningful abstract representation to a coded implementation.

Disadvantages of FSM

- The predictable nature of deterministic FSMs can be unwanted in some domains such as computer games.
- Larger systems implemented using a FSM can be difficult to manage and maintain.
- Not suited to all problem domains, should only be used when a systems behavior can be decomposed into separate states, transitions and conditions need to be known up front and be well defined.
- The conditions for state transitions are ridged, meaning they are fixed.

Review Questions

1. What is a state ?
2. Define present state and next state.
3. Draw and explain the block diagram of Moore model.
4. What is Moore machine ?
5. Draw and explain the block diagram of Mealy model.
6. What is a Mealy machine ? Give an example.
7. Compare Moore and Mealy circuits.
8. Explain state table.
9. Explain state diagram.
10. What is transition table ?

SPPU : Dec.-10, Marks 4

SPPU : May-12, Dec.-16, Marks 4

SPPU : Dec.-10, 12, May-12, Marks 4

SPPU : Dec.-12, May-13, Marks 2

SPPU : Dec.-12, May-13, Marks 2

7.2 Design of Clocked Sequential Circuits

SPPU : Dec.-05,07,08,10,12,13,14, May-05,06,11,12,13

The recommended steps for the design of a clocked synchronous sequential circuit are as follows :

1. It is necessary to first obtain the state table from the given circuit information such as a state diagram, a timing-diagram, or other pertinent information.
2. The number of states may be reduced by state reduction technique if the sequential circuit can be categorized by input-output relationships independent of the number of states.
3. Assign binary values to each state in the state table.
4. Determine the number of flip-flops needed and assign a letter symbol to each.
5. Choose the type of flip-flop to be used.
6. From the state table, derive the circuit excitation and output tables.
7. Using the K-map or any other simplification method, derive the circuit output functions and the flip-flop input functions.
8. Draw the logic diagram.

Let us see the details of various steps involved in the design of clocked sequential circuits.

7.2.1 State Reduction

The state reduction technique basically avoids the introduction of redundant states. The reduction in redundant states reduce the number of required flip-flops and logic gates, reducing the cost of the final circuit. The two states are said to be redundant or equivalent, if every possible set of inputs generate exactly same output and same next state. When two states are equivalent, one of them can be removed without altering the input-output relationship.

Let us illustrate the state reduction technique with an example. We start with a sequential circuit whose specification is given in the state diagram of Fig. 7.2.1. As shown in the diagram, the states are denoted by letter symbols instead of their binary values, because in state reduction technique internal states are not important; but only input-output sequences are important.

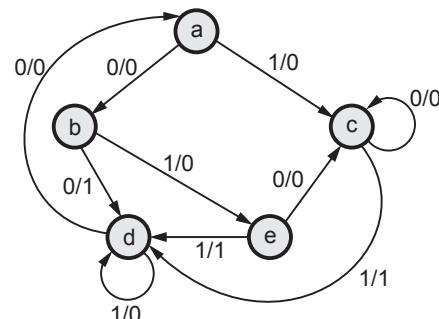


Fig. 7.2.1 State diagram

Step 1 : Determine the state table for given state diagram

Table 7.2.1 shows the state table for given state diagram

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	b	c	0	0
b	d	e	1	0
c	c	d	0	1
d	a	d	0	0
e	c	d	0	1

Table 7.2.1 State table

Step 2 : Find equivalent states

As mentioned earlier, in equivalent states every possible set of inputs generate exactly same output and same next state. In the given circuit there are two input combinations : X = 0 and X = 1. Looking at the state table for two present states that go to the same next state and have the same output for both input combinations, we can easily find that states c and e are equivalent. This is because, c and e both states go to states c and d and have outputs of 0 and 1 for X = 0 and X = 1, respectively. Therefore, state e can be removed and replaced by c. The final reduced table is shown in Table 7.2.2. The state diagram for the reduced table consists of only four states and is shown in Fig. 7.2.2.

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	b	c	0	0
b	d	c	1	0
c	c	d	0	1
d	a	d	0	0

Table 7.2.2 Reduced state table

Examples for Understanding

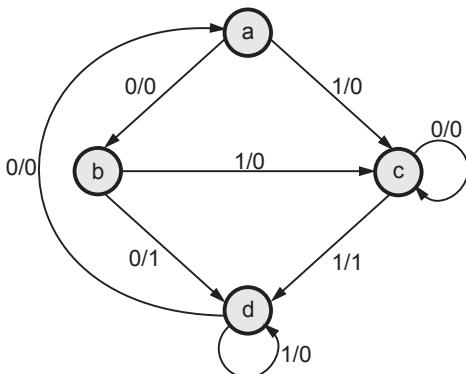


Fig. 7.2.2 Reduced state diagram

Example 7.2.1 Minimize the state table shown given below.

Present state	Next state, Z (Output)	
	X (Input)	
	0	1
A	B,0	C,0

<i>B</i>	<i>B,0</i>	<i>D,0</i>
<i>C</i>	<i>B,0</i>	<i>C,0</i>
<i>D</i>	<i>E,1</i>	<i>C,0</i>
<i>E</i>	<i>B,0</i>	<i>D,0</i>

Solution : Step 1 : Find equivalent states. The Table 7.2.3 shows the equivalent states in the given state table. The states A and C, and states B and E generate exactly same output and same next state. Hence, state A and C are equivalent and similarly states B and E are equivalent.

Present state	Next state, Z(output)	
	X = 0	X = 1
A	B, 0	C, 0
B	B, 0	D, 0
C	B, 0	C, 0
D	E, 1	C, 0
E	B, 0	D, 0

Table 7.2.3 Equivalent states

Step 2 : Replace redundant states with equivalent states.

∴ Replace C by A and replace E by B, and remove states C and E.

Now, there are no equivalent states and hence Table 7.2.3 (a) shows the minimized state table.

Present state	Next state, Z (output)	
	X = 0	X = 1
A	B, 0	A, 0
B	B, 0	D, 0
D	B, 1	A, 0

Table 7.2.3 (a) Minimized state table

Example 7.2.2 Minimize the state table shown given below :

Present state	Next state, Z (Output)	
	X (Input)	
	0	1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	G, 0	F, 1
G	A, 0	F, 1

Solution : Step 1 : Find equivalent states

Step 2 : Replace redundant state with equivalent state.

Therefore, replace G by E and remove state G. Looking at Table 7.2.4 (b), we find that states D and F are equivalent. Therefore, replace F by D and remove state F. Now, there are no equivalent states and hence Table 7.2.4 (c) shows the minimized state table.

Present state	Next state, Z(output)	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	G, 0	F, 1
G	A, 0	F, 1

Equivalent states

Table 7.2.4 (a) Equivalent states

Present state	Next state, Z(output)	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	E, 0	F, 1

Equivalent states

Table 7.2.4 (b) Equivalent states

Present state	Next state, Z (Output)	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	D, 1
E	A, 0	D, 1

Table 7.2.4 (c) Minimized state table

7.2.2 State Assignment

SPPU : Dec.-13

In sequential circuits we know that the behaviour of the circuit is defined in terms of its inputs, present states, next states and outputs. To generate desired next state at particular present state and inputs, it is necessary to have specific flip-flop inputs. These flip-flop inputs are described by a set of Boolean functions called flip-flop input functions. To determine the flip-flop input functions, it is necessary to represent states in the state diagram using binary values instead of alphabets. This procedure is known as state assignment. We must assign binary values to the states in such a way that it is possible to implement flip-flop input

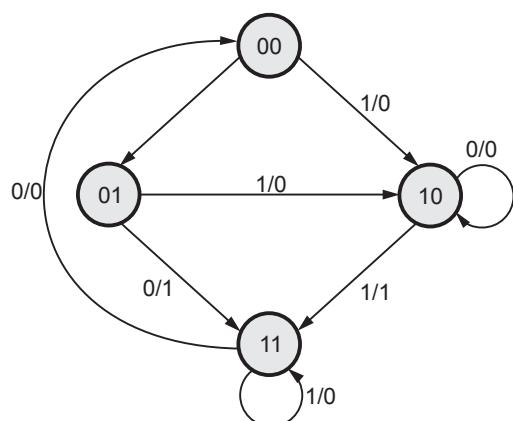


Fig. 7.2.3 State diagram with binary states

functions using minimum logic gates. Fig. 7.2.3 shows the state diagram shown in Fig. 7.2.3 after state assignments.

Rules for state assignments

There are two basic rules for making state assignments.

Rule 1 : States having the **same** NEXT STATES for a given input condition should have assignments which can be grouped into logically adjacent cells in a K-map.

Fig. 7.2.3 shows the example for Rule 1. As shown in the Fig. 7.2.4, there are four states whose next state is same. Thus states assignments for these states are 100, 101, 110 and 111 which can be grouped into logically adjacent cells in a K-map.

Rule 2 : States that are the NEXT STATES of a single state should have assignment which can be grouped into logically adjacent cells in a K-map.

Fig. 7.2.5 shows the example for Rule 2. As shown in the Fig. 7.2.5, for state 000, there are four next states. These states are assigned as 100, 101, 110 and 111 so that they can be grouped into logically adjacent cells in a K-map and table shows the state table with assigned states.

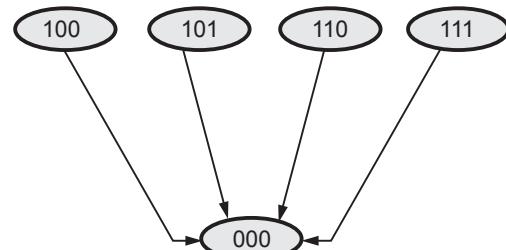


Fig. 7.2.4 Example of using Rule 1

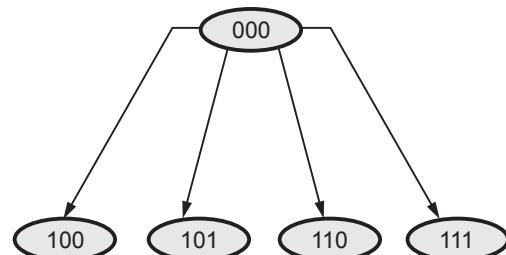


Fig. 7.2.5 Example of using Rule 2

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
00	01	10	0	0
01	11	10	1	0
10	10	11	0	1
11	00	11	0	0

Table 7.2.5 State table with assigned states

7.2.3 Choice of Flip-Flops and Derivation of Next State and Output

From the state assigned state table shown in Table 7.2.5, we can derive the logic expression for the next state and output functions. But first we have to decide on the type of flip-flops that will be used in the circuit. The most straightforward choice is to use D flip-flops, because in this case the values of next state are simply clocked into the flip-flops to become the new values of present state. For other types of flip-flops, such as JK, T and RS the relationship between the next-state variable and inputs to a flip-flop is not as straightforward as D flip-flop. For other types of flip-flops we have to refer excitation table of flip-flop to find flip-flop inputs. This is illustrated in the following example.

Example 7.2.3 A sequential circuit has one input and one output. The state diagram is shown in Fig. 7.2.6. Design the sequential circuit with a) D flip-flops b) T flip-flops c) RS flip-flops and d) JK- flip-flops.

Solution : The state table for the state diagram shown in Fig. 7.2.6 is as given in Table 7.2.6.

As seen from the state table there are no equivalent states. Therefore, no reduction is the state diagram. The state table shows that circuit goes through four states, therefore we require 2 flip-flops (number of states = 2^m , where m = number of flip-flops). Since two flip-flops are required first is denoted as A and second is denoted as B.

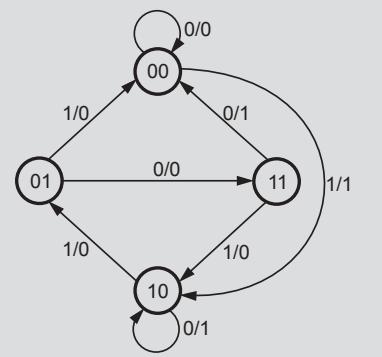


Fig. 7.2.6

1. Design using D flip-flops

As mentioned earlier, for D flip-flops next states are nothing but the new present states. Thus, we can directly use next states to determine the flip-flop input with the help of K-map simplification.

Present state		Next state		Output	
		X = 0	X = 1	X = 0	X = 1
A	B	AB	AB	Y	Y
0	0	0 0	1 0	0	1
0	1	1 1	0 0	0	0
1	0	1 0	0 1	1	0
1	1	0 0	1 0	1	0

Table 7.2.6

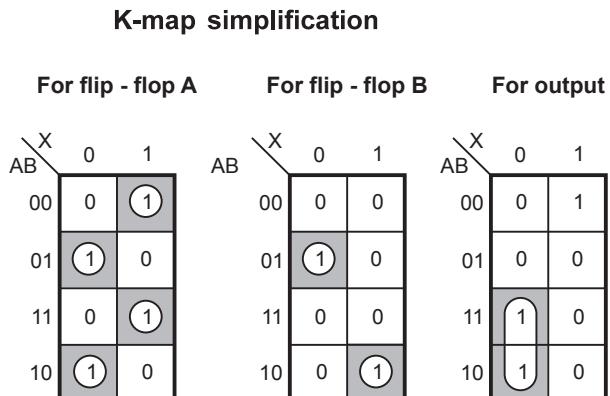


Fig. 7.2.7

$$D_A = \overline{A} \overline{B} X + \overline{A} \overline{B} \overline{X} + A B X + A \overline{B} \overline{X} \quad \text{and}$$

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

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

With these flip-flop input functions and circuit output function we can draw the logic diagram. (See Fig. 7.2.8 on next page)

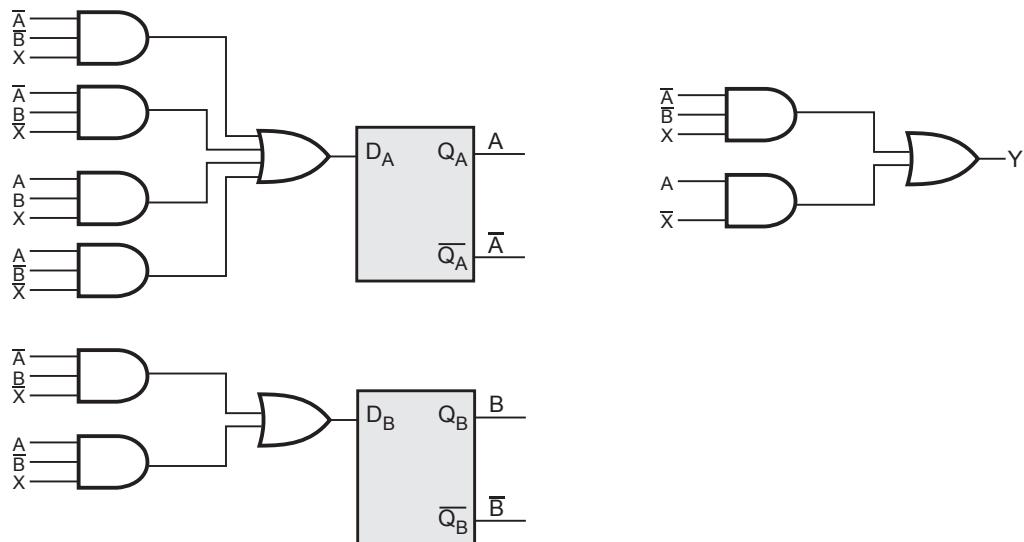


Fig. 7.2.8 Logic diagram of given sequential circuit using D flip-flop

2. Design using T flip-flops

Using the excitation table for T flip-flop shown in Table 7.2.7 we can determine the excitation table for the given circuit as shown in Table 7.2.8.

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 7.2.7 Excitation table for T flip-flop

The first row of circuit excitation table shows that there is no change in the state for both flip-flops. The transition from $0 \rightarrow 0$ for T flip-flop requires input T to be at logic 0. The second row shows that flip-flop A has transition $0 \rightarrow 1$. It requires the input T_A to be at logic 1. It requires the input T_A to be at logic 1. Similarly, we can find inputs for each flip-flop for each row in the table by referring present state, next state and excitation table.

Let us use K-map simplification to determine the flip-flop input functions and circuit output functions.

Present state	Input		Next state		Flip-flop inputs		Output	
	A	B	X	A	B	T_A	T_B	
0 0	0	0	0	0	0	0	0	0
0 0	0	0	1	1	0	1	0	1
0 1	0	1	0	1	1	1	0	0
0 1	0	1	1	0	0	0	1	0
1 0	1	0	0	1	0	0	0	1
1 0	1	0	1	0	1	1	1	0
1 1	1	1	0	0	0	1	1	1

Table 7.2.8 Circuit excitation table

Therefore, input function for

$$T_A = B\bar{X} + \bar{B}X,$$

$$T_B = AB + BX + AX, \text{ and}$$

$$\text{Circuit output function} = A\bar{X} + \bar{A}\bar{B}X$$

With these flip-flop input functions and circuit output function we can draw the logic diagram as follows :

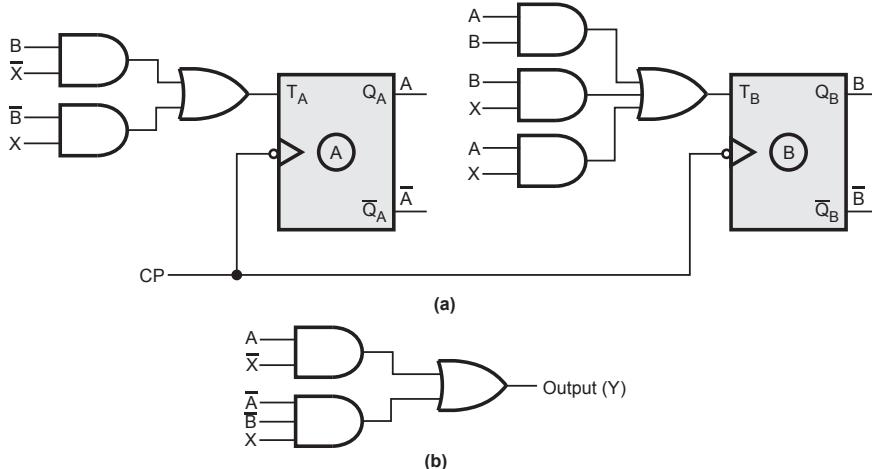


Fig. 7.2.10 Logic diagram of given sequential circuit using T flip-flops

3. Design using RS flip-flops

Using the excitation table for RS flip-flop shown in Table 7.2.9 we can determine the excitation table for the given circuit as shown in Table 7.2.10.

Q_n	Q_{n+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

Table 7.2.9 Excitation table for RS flip-flop

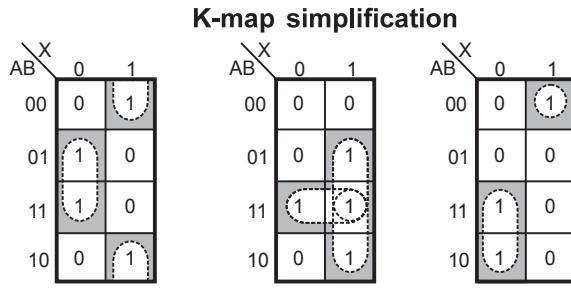


Fig. 7.2.9

Present state	Input	Next state	Flip-flop inputs				Output
			A	B	R _A	S _A	
0 0	0	0 0	X	0	X	0	0
0 0	1	1 0	0	1	X	0	1
0 1	0	1 1	0	1	0	X	0
0 1	1	0 0	X	0	1	0	0
1 0	0	1 0	0	X	X	0	1
1 0	1	0 1	1	0	0	1	0
1 1	0	0 0	1	0	1	0	1
1 1	1	1 0	0	X	1	0	0

Table 7.2.10

The first row of circuit excitation table shows that there is no change in the state for both flip-flops. The transition from $0 \rightarrow 0$ for RS flip-flop requires inputs R and S to be X and 0, respectively. Similarly, we can determine inputs for each flip-flop for each row in the table by referring present state, next state and excitation table. Let us use K-map simplification to determine the flip-flop input functions and circuit output functions.

Therefore input function for

$$R_A = AB\bar{X} + A\bar{B}X$$

$$S_A = \bar{A}B\bar{X} + \bar{A}\bar{B}X$$

$$R_B = AB + BX$$

$$S_B = \bar{A}\bar{B}X \text{ and,}$$

$$\text{Circuit output function} = A\bar{X} + \bar{A}\bar{B}X$$

With these flip-flop input functions and circuit output function we can draw the logic diagram as follows :

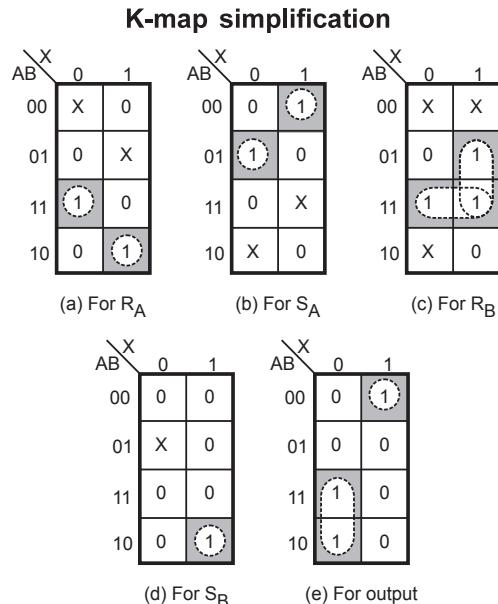


Fig. 7.2.11

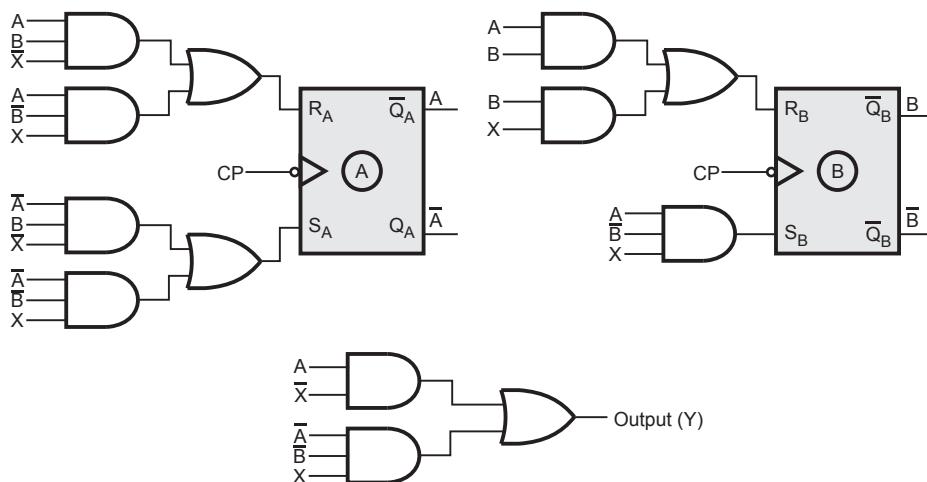


Fig. 7.2.12 Logic diagram of given sequential circuit using RS flip-flop

4. Design using JK flip-flops

Using the excitation table for JK flop-flop shown in Table 7.2.11 we can determine the excitation table for the given circuit as shown in Table 7.2.12.

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 7.2.11 Excitation table for JK flip-flop

Present state		Input	Next state		Flip-flop inputs				Output
A	B	X	A	B	J_A	K_A	J_B	K_B	Y
0	0	0	0	0	0	X	0	X	0
0	0	1	1	0	1	X	0	X	1
0	1	0	1	1	1	X	X	0	0
0	1	1	0	0	0	X	X	1	0
1	0	0	1	0	X	0	0	X	1
1	0	1	0	1	X	1	1	X	0
1	1	0	0	0	X	1	X	1	1
1	1	1	1	0	X	0	X	1	0

Table 7.2.12

The first row of circuit excitation table shows that there is no change in the state for both flip-flops. The transition from 0 → 0 for JK flip-flop requires inputs J and K to be 0 and X, respectively. Similarly, we can determine inputs for each flip-flop for each row in the table by referring present state, next state and excitation table. Let us use K-map simplification to determine the flip-flop input functions and circuit output functions.

K-map simplification

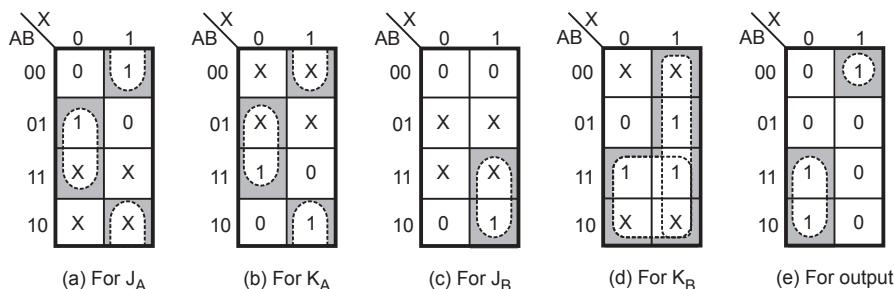


Fig. 7.2.13

Therefore, input function for

$$J_A = B\bar{X} + \bar{B}X$$

$$K_A = B\bar{X} + \bar{B}X$$

$$J_B = AX$$

$$K_B = A + X$$

$$\text{Circuit output function} = A\bar{X} + \bar{A}\bar{B}X$$

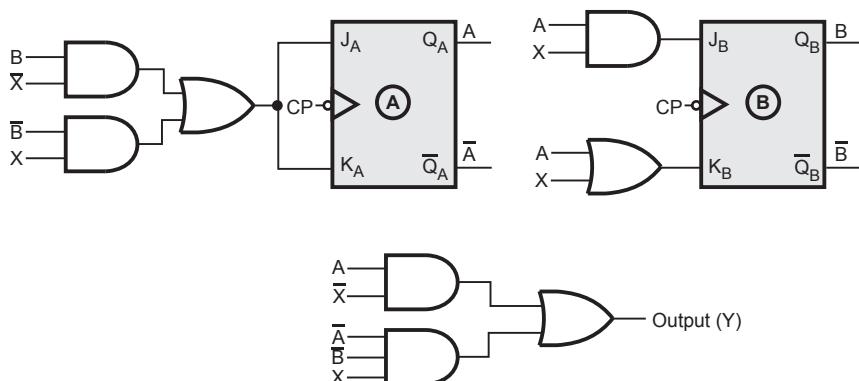


Fig. 7.2.14 Logic diagram of given sequential circuit using JK flip-flop

7.2.4 State Assignment Problem

Example 7.2.4 Design a sequential circuit for a state diagram shown in Fig. 7.2.15.

Solution :

Step 1 : Assign binary values to each state. Using random state assignment we get

$a = 000$, $b = 001$, $c = 010$, $d = 011$ and $e = 100$. The excitation table with these assignments is as given in Table 7.2.13.

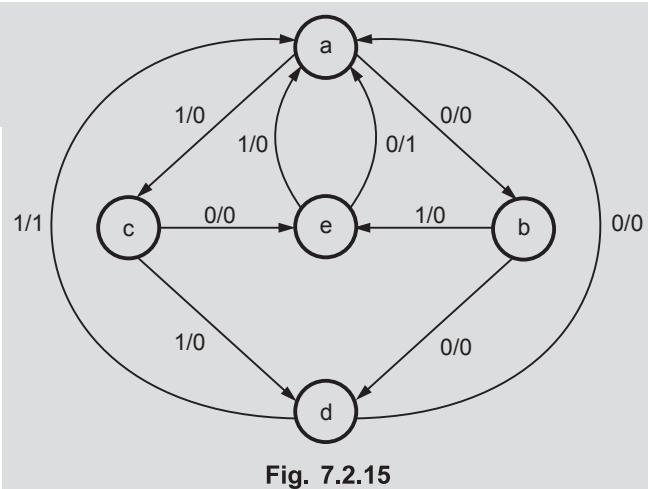


Fig. 7.2.15

Step 2 : Find number of flip-flops needed. Since there are five states we require $2^n \geq 5$
 $\therefore n = 3$ flip-flops.

Let us use D flip-flops

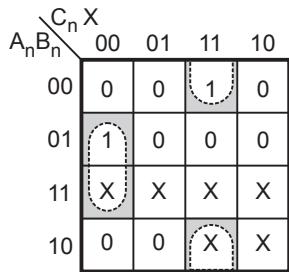
Step 3 : Derive the excitation table.

Present state			Input	Next state			Output
A_n	B_n	C_n	X	A_{n+1}	B_{n+1}	C_{n+1}	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0

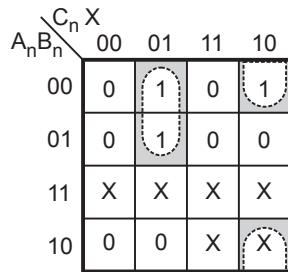
0	1	0	0	1	0	0	0
0	1	0	1	0	1	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Table 7.2.13

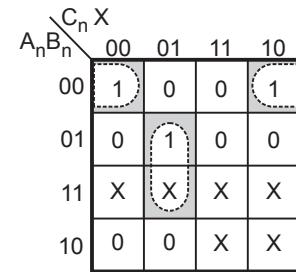
Step 4 : Derive circuit output and flip-flop input functions using K-map simplification



$$D_A = B_n \bar{C}_n \bar{X} + \bar{B}_n C_n X$$



$$D_B = \bar{A}_n \bar{C}_n X + \bar{B}_n C_n \bar{X}$$



$$D_C = \bar{A}_n \bar{B}_n \bar{X} + B_n \bar{C}_n X$$

Fig. 7.2.16 (a)

The random assignments require:

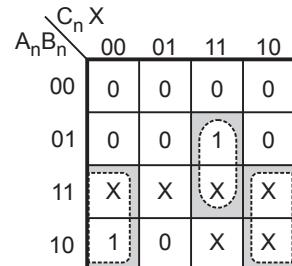
- 7 three input AND functions
 - 1 two input AND function
 - 4 two input OR functions
-
- 12 gates with 31 inputs

Step 5 : State assignment using rules.

Now we will apply the state assignment rules and compare the results.

Looking at Fig. 7.2.16 (a) and (b).

Rule 1 says that : e and d must be adjacent, and



$$Z = B_n C_n X + A_n \bar{X}$$

Fig. 7.2.16 (b)

b and c must be adjacent

Rule 2 says that : b and c must be adjacent, and
e and d must be adjacent

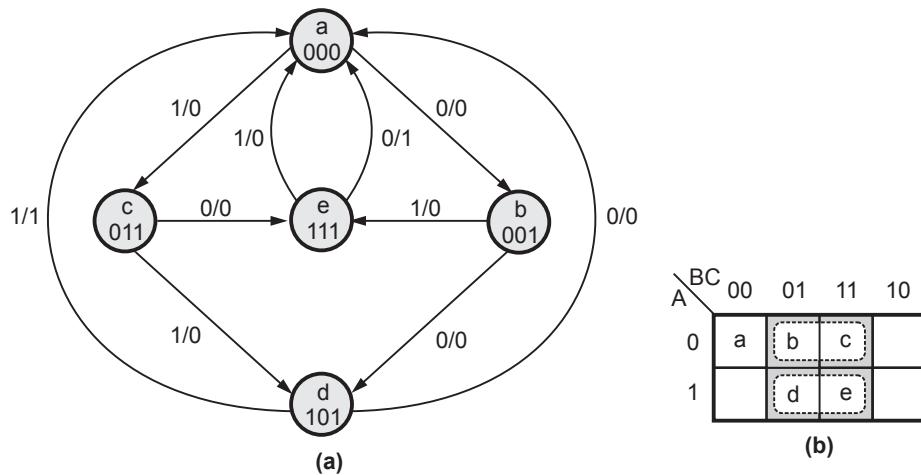


Fig. 7.2.17 (a) State diagram after applying Rules 1 and 2

(b) K-map showing b-c d-e adjacent to each other

Step 6 : Derive excitation table

Applying Rule 1 Rule 2 to the state diagram we get the excitation table as shown in the Table 7.2.14.

Present state			Input	Next state			Output
A _n	B _n	C _n	X	A _{n+1}	B _{n+1}	C _{n+1}	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	0	X	X	X	X
0	1	0	1	X	X	X	X
0	1	1	0	1	1	1	0
0	1	1	1	1	0	1	0
1	0	0	0	X	X	X	X
1	0	0	1	X	X	X	X

1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

Table 7.2.14

Step 7 : Derive circuit output and flip-flop input functions using K-map simplification.

		C _n X	00	01	11	10
		A _n B _n	00	01	11	10
00	00	0	0	1	1	
01	01	X	X	1	1	
11	11	X	X	0	0	
10	10	X	X	0	0	

$$A_{n+1} = D_A = \bar{A}_n C_n$$

		C _n X	00	01	11	10
		A _n B _n	00	01	11	10
00	00	0	1	1	0	
01	01	X	X	0	1	
11	11	X	X	0	0	
10	10	X	X	0	0	

$$B_{n+1} = D_B = \bar{A}_n \bar{B}_n X + \bar{A}_n \bar{B}_n \bar{X}$$

		C _n X	00	01	11	10
		A _n B _n	00	01	11	10
00	00	1	1	1	1	
01	01	X	X	1	1	
11	11	X	X	0	0	
10	10	X	X	0	0	

$$C_{n+1} = D_C = \bar{A}_n$$

Fig. 7.2.18 (a)

The state assignments using Rule 1 and 2 require :

- | | |
|-------|---------------------------|
| 4 | three input AND functions |
| 1 | two input AND functions |
| 2 | two input OR functions |
| <hr/> | |
| 7 | gates with 18 inputs |

Thus by simply applying Rules 1 and 2 good results have been achieved.

		C _n X	00	01	11	10
		A _n B _n	00	01	11	10
00	00	0	0	0	0	0
01	01	X	X	0	0	0
11	11	X	X	0	1	
10	10	X	X	1	0	0

$$Z = A_n B_n \bar{X} + A_n \bar{B}_n \bar{X}$$

Fig. 7.2.18 (b)

7.2.5 Design with Unused States

There are occasions when a sequential circuit may use less than the available these maximum number of states. We can consider the unused states as don't care conditions and can be used to simplify the input functions to flip-flops.

Let us consider one example. First we will design the given sequential circuit without using unused states and then we will design the given sequential circuit using unused states.

Example 7.2.5 Design the sequential circuit for the state diagram shown in Fig. 7.2.19 use JK flip-flops.

Solution : Step 1 : Derive excitation table

The excitation table for given state diagram is as follows

Present state	Input	Next state			Flip-flop inputs						Output			
		A	B	C	X	A	B	C	J _A	K _A	J _B	K _B	J _C	K _C
0 0 1	0	0	1	0	0	X	1	X	X	X	1			1
0 0 1	1	0	1	1	0	X	1	X	X	X	0			1
0 1 0	0	0	0	1	0	X	X	1	1	1	X			0
0 1 0	1	0	1	0	0	X	X	0	0	0	X			1
0 1 1	0	1	0	0	1	X	X	1	X	1				1
0 1 1	1	1	1	0	1	X	X	0	X	1				0
1 0 0	0	1	1	0	X	0	1	X	0	X	0			0
1 0 0	1	0	0	1	X	1	0	X	1	X				1
1 1 0	0	1	1	0	X	0	X	0	0	X				1
1 1 0	1	0	1	0	X	1	X	0	0	X				0

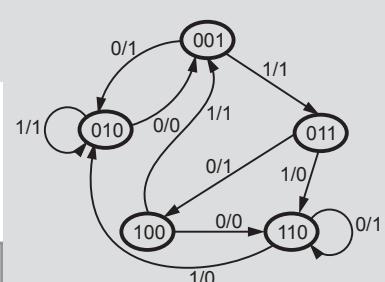


Fig. 7.2.19

Table 7.2.15

Step 2 : K-map simplification for JK inputs and circuit output

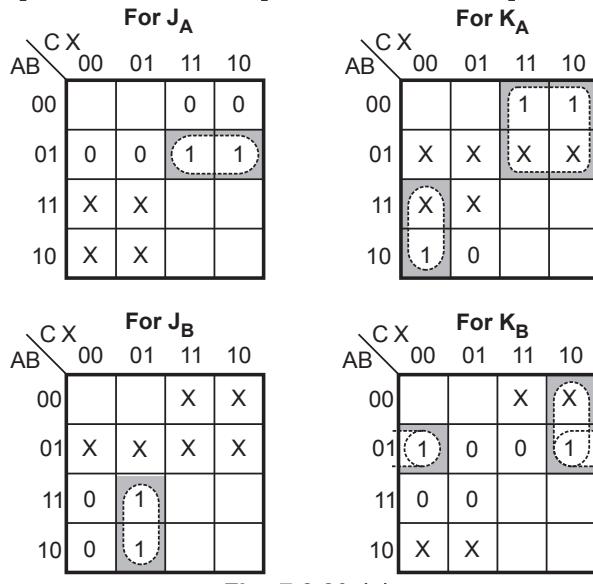


Fig. 7.2.20 (a)

		For J_C				For K_C				For Output						
		AB	00	01	11	10	AB	00	01	11	10	AB	00	01	11	10
C	X	00		X	X		00		1		0	00		1		1
0	1	01	1	0	X	X	01	X	X	1	1	01	0	1	0	1
1	0	11	0	0			11	X	X			11	1	0		
0	1	10	0	1			10	X	X			10	0	1		

Fig. 7.2.20 (b)

Therefore, input functions for

$$J_A = \overline{ABC}$$

$$K_A = A\overline{C}\overline{X} + \overline{AC}$$

$$J_B = A\overline{CX}$$

$$K_B = \overline{AB}\overline{X} + \overline{AC}\overline{X}$$

$$J_C = \overline{AB}\overline{X} + A\overline{B}\overline{CX}$$

$$K_C = \overline{AB} + \overline{ACX} \text{ and}$$

Circuit output function, $Y = AB\overline{C}\overline{X} + \overline{ABC} + A\overline{B}\overline{CX} + \overline{A}\overline{BC} + \overline{ACX}$

Step 3 : Draw logic diagram.

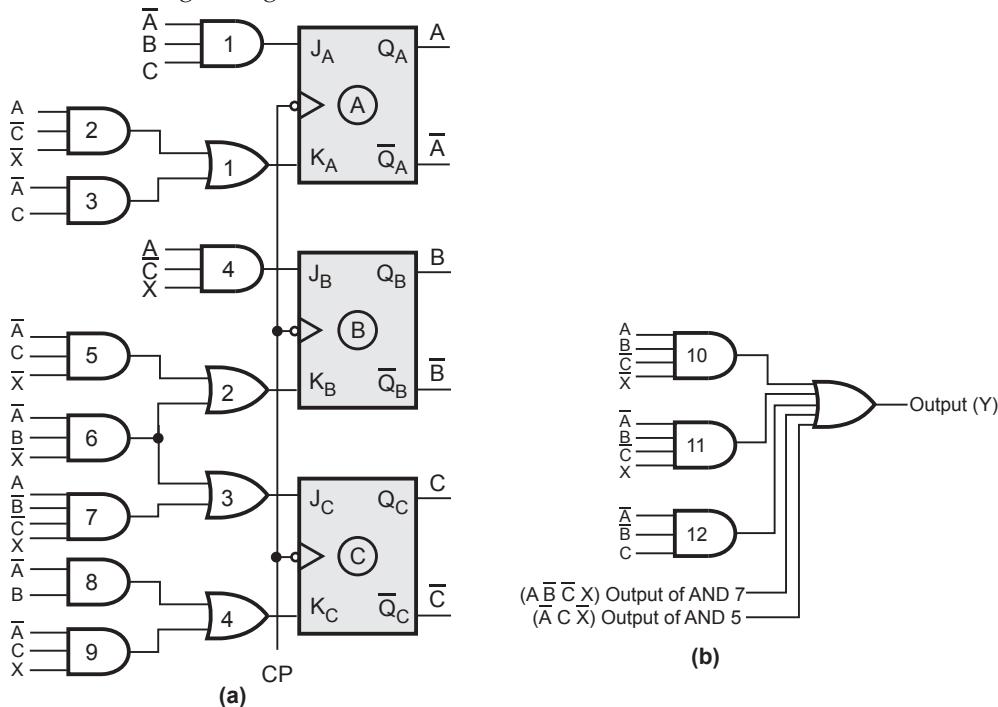


Fig. 7.2.21

Step 4 : Derive circuit output and flip-flop inputs considering unused states.

Let us see the circuit design with the use of unused states. These unused states 000, 101 and 111 are considered as a don't cares and are used to simplify the K-maps as follows :

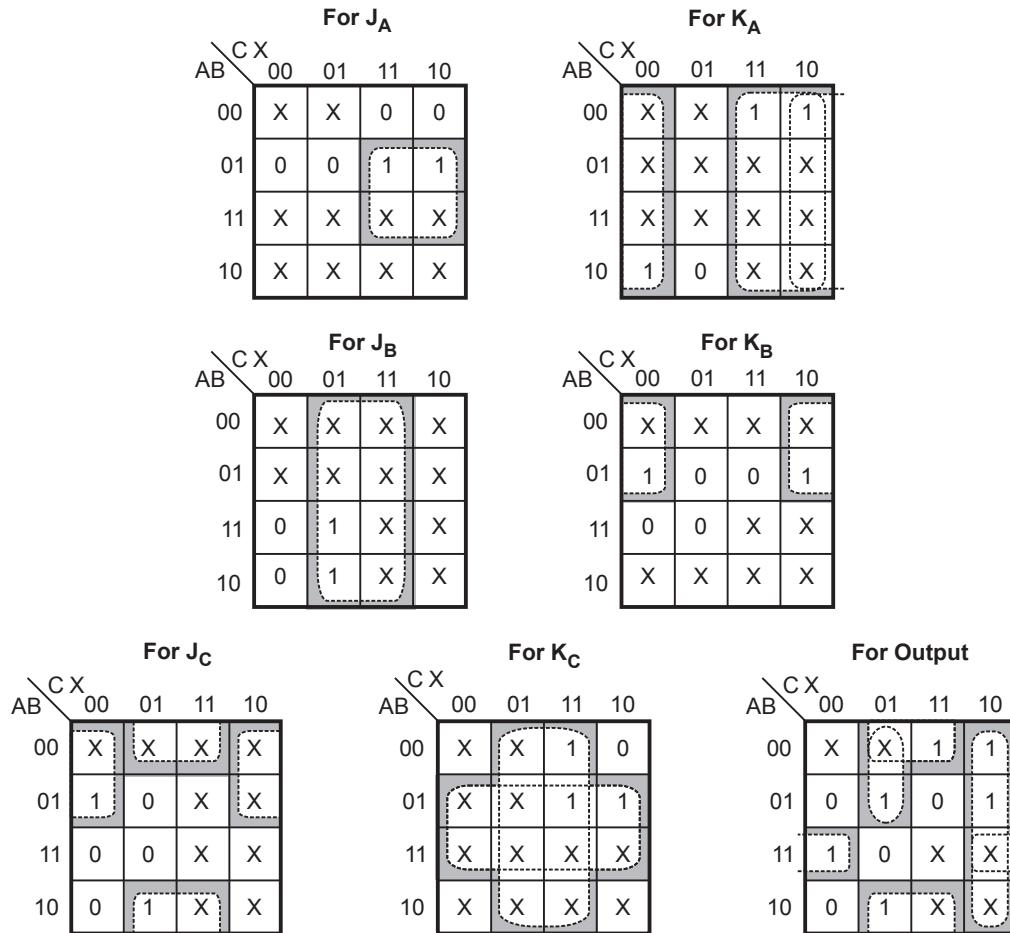


Fig. 7.2.22

Therefore, input functions for

$$J_A = BC$$

$$K_A = \bar{X} + C$$

$$J_B = X$$

$$K_B = \bar{A} \bar{X}$$

$$J_C = \bar{A} \bar{X} + \bar{B}X \quad \text{and} \quad K_C = B + X$$

The circuit output function $Y = AB\bar{X} + A\bar{C}X + \bar{B}X + C\bar{X}$

Step 5 : Draw logic diagram.

From the above logic diagram it can be realized that using unused state as don't cares we can further simplify the flip-flop input functions and circuit output function.

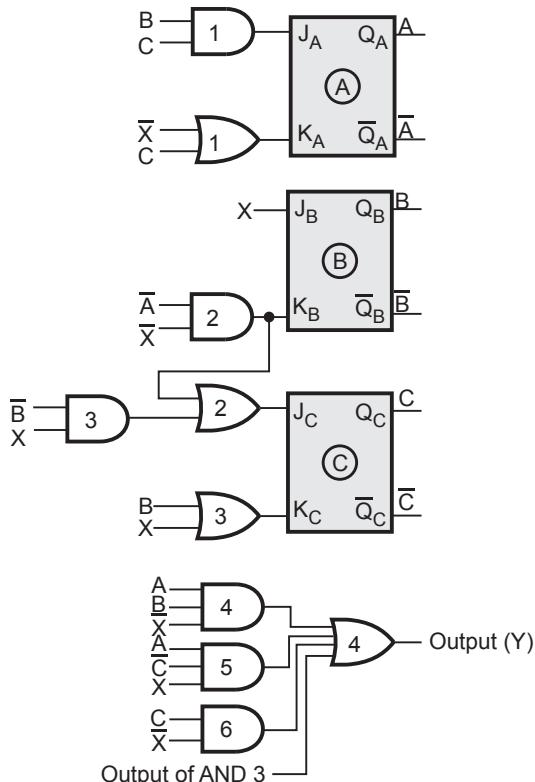


Fig. 7.2.23 Logic diagram

Examples with Solutions

Example 7.2.6 Design a synchronous sequential circuit using JK for the given state diagram.

Solution : Step 1 : Since N = 4. Number of flip-flops needed = 2

Step 2 : Flip-flops to be used : JK

Step 3 : Determine the excitation table.

Present state	Input	Next state	Flip-flop inputs
A	B	X	A ⁺ B ⁺ J _A K _A J _B K _B
0	0	0	0 0 0 X 0 X
0	0	1	0 1 0 X 1 X
0	1	0	1 0 1 X X 1

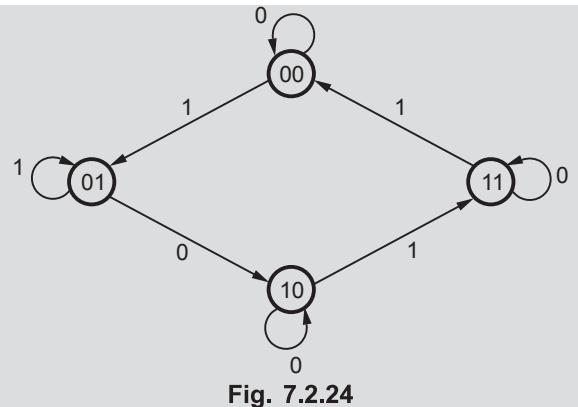


Fig. 7.2.24

0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

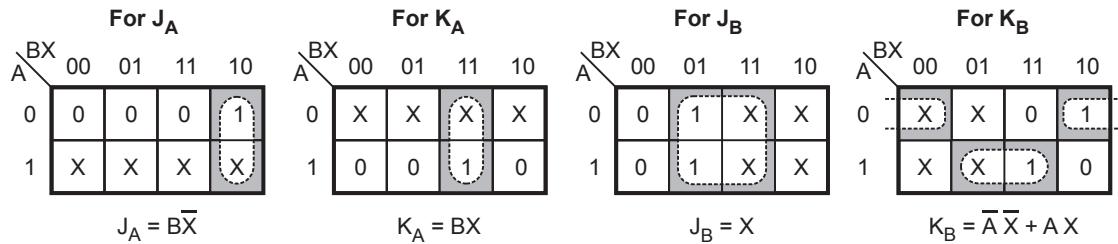
Step 4 : K-map simplification

Fig. 7.2.25

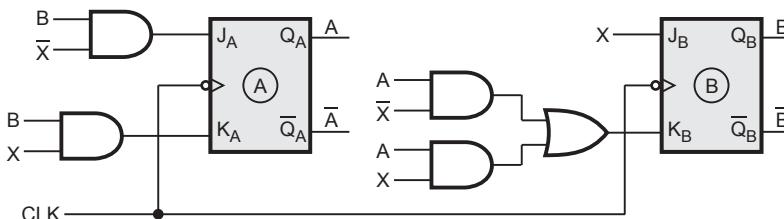
Step 5 : Logic diagram

Fig. 7.2.26

Example 7.2.7 Design the sequential circuit using JK flip-flops, for state diagram shown in Fig. 7.2.27. **SPPU : Dec.-10, May-05,11, Marks 8**

Solution : The state table for the given state diagram is as follows.

Input X	Present State	Next State	Output Y
0	S_0	S_0	0
0	S_1	S_2	1
0	S_2	S_1	1
0	S_3	S_0	1
1	S_0	S_1	0
1	S_1	S_3	0
1	S_2	S_0	1
1	S_3	S_1	0

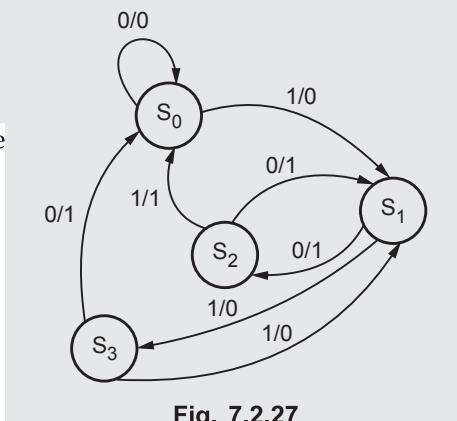
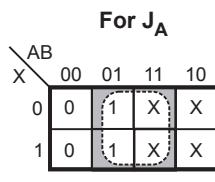


Fig. 7.2.27

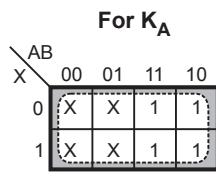
Assigning states $S_0 = 00$, $S_1 = 01$, $S_2 = 10$ and $S_3 = 11$ we have,

Input	Present State		Next State		Flip-Flop Excitation Table				Output
	A	B	A_{+1}	B_{+1}	J_A	K_A	J_B	K_B	
0	0	0	0	0	0	X	0	X	0
0	0	1	1	0	1	X	X	1	1
0	1	0	0	1	X	1	1	X	1
0	1	1	0	0	X	1	X	1	1
1	0	0	0	1	0	X	1	X	0
1	0	1	1	1	1	X	X	0	0
1	1	0	0	0	X	1	0	X	1
1	1	1	0	1	X	1	X	0	0

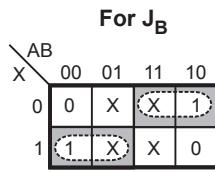
K-map simplification



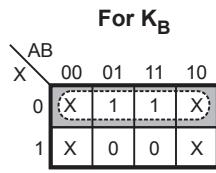
$$J_A = B$$



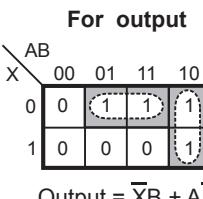
$$K_A = 1$$



$$J_B = \bar{X}A + X\bar{A} \\ = A \oplus X$$



$$K_B = \bar{X}$$



$$\text{Output} = \bar{X}B + A\bar{B}$$

Fig. 7.2.28

Logic diagram

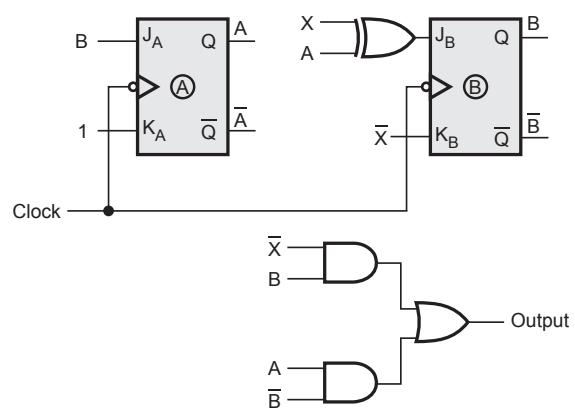


Fig. 7.2.29

Example 7.2.8 Implement the following state diagram using D flip-flop.

SPPU : Dec.-12, Marks 10

Solution : The excitation table for the given state diagram is shown in the Table 7.2.16. Since D flip-flops are used, flip-flop excitations are same as next states.

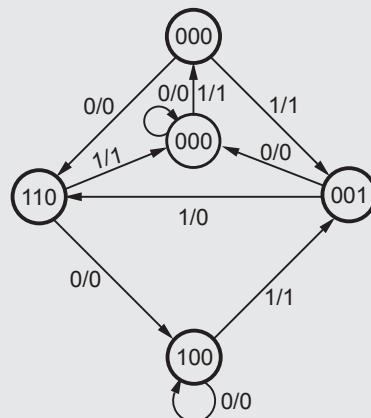


Fig. 7.2.30

Input X	Present state			Next state			Output Y
	A	B	C	A^+	B^+	C^+	
0	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0
0	0	1	1	X	X	X	X
0	1	0	0	1	0	0	0
0	1	0	1	X	X	X	X
0	1	1	0	1	0	0	0
0	1	1	1	X	X	X	X
1	0	0	0	0	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	0	0	0	1
1	0	1	1	X	X	X	X
1	1	0	0	0	0	1	1
1	1	0	1	X	X	X	X
1	1	1	0	0	1	0	1
1	1	1	1	X	X	X	X

K-map Simplification

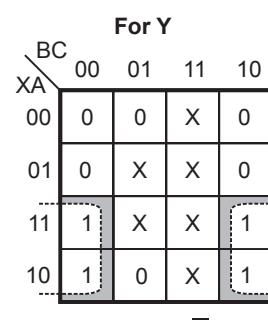
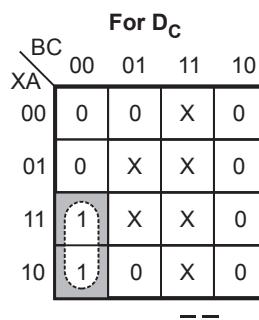
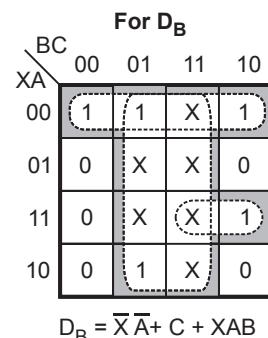
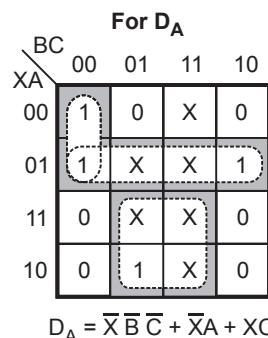


Table 7.2.16 Excitation table

Fig. 7.2.31

Implementation

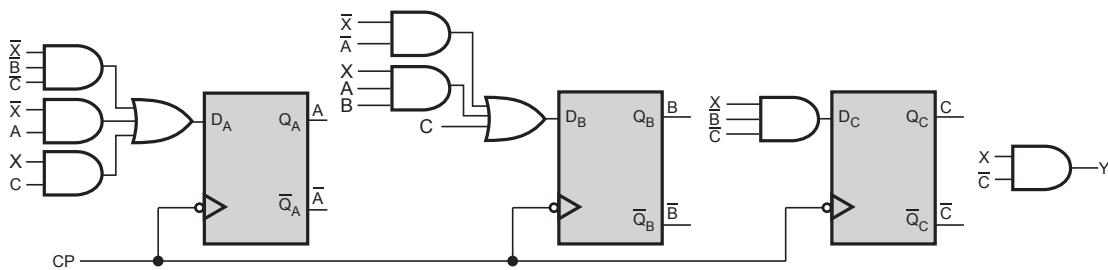


Fig. 7.2.32

Example 7.2.9 Using J-K flip-flops, design a synchronous counter that has the following sequence : 0 → 2 → 5 → 6 → 0
undesired states 1, 3, 4, 7 must always go to 0 on the next clock pulse.

SPPU : May-05, Marks 10

Solution :

Excitation table

Present state	Next state			Flip-flop inputs								
	A	B	C	A_{+1}	B_{+1}	C_{+1}	J_A	K_A	J_B	K_B	J_C	K_C
0 0 0	0	1	0	0	X	1	X	0	X			
0 0 1	0	0	0	0	X	0	X	X	X	1		
0 1 0	1	0	1	1	X	X	1	1	1	X		
0 1 1	0	0	0	0	X	X	1	X	1	X	1	
1 0 0	0	0	0	X	1	0	X	0	X	0	X	
1 0 1	1	1	0	X	0	1	X	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	X	1	

K-map simplification

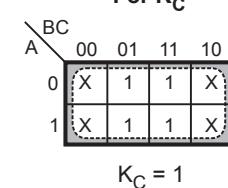
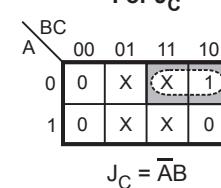
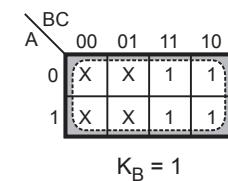
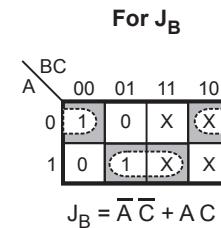
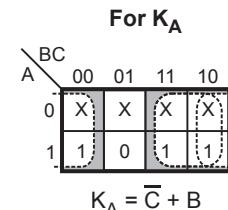
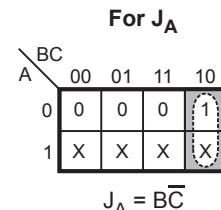


Table 7.2.17

Fig. 7.2.33

Logic diagram

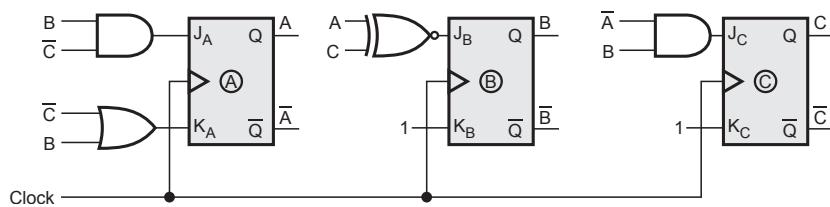


Fig. 7.2.34

Example 7.2.10 Design and implement 4-bit binary counter (using D flip-flops) which counts all possible odd numbers only.

SPPU : Dec.-05, Marks 8

Solution : The state diagram for given problem is as shown in the Fig. 7.2.35.

Excitation table

Present state				Next state			
Q_A	Q_B	Q_C	Q_D	Q_A+	Q_B+	Q_C+	Q_D+
0	0	0	0	x	x	x	x
0	0	0	1	0	0	1	1
0	0	1	0	x	x	x	x
0	0	1	1	0	1	0	1
0	1	0	0	x	x	x	x
0	1	0	1	0	1	1	1
0	1	1	0	x	x	x	x
0	1	1	1	1	0	0	1
1	0	0	0	x	x	x	x
1	0	0	1	1	0	1	1
1	0	1	0	x	x	x	x
1	0	1	1	1	1	0	1
1	1	0	0	x	x	x	x
1	1	0	1	1	1	1	1
1	1	1	0	x	x	x	x
1	1	1	1	0	0	0	1

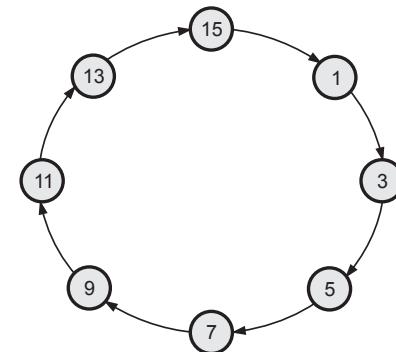
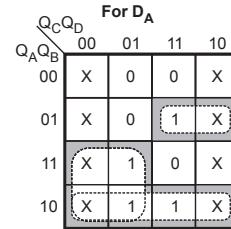
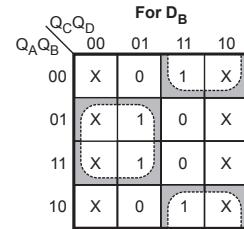


Fig. 7.2.35 State diagram

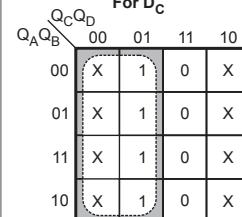
K-map simplification



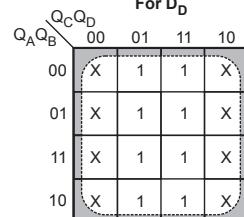
$$D_A = Q_A \bar{Q}_C + Q_A \bar{Q}_B + \bar{Q}_A Q_B Q_C$$



$$D_B = Q_B \bar{Q}_C + \bar{Q}_B Q_C$$



$$D_C = \bar{Q}_C$$



$$D_D = 1$$

Fig. 7.2.36

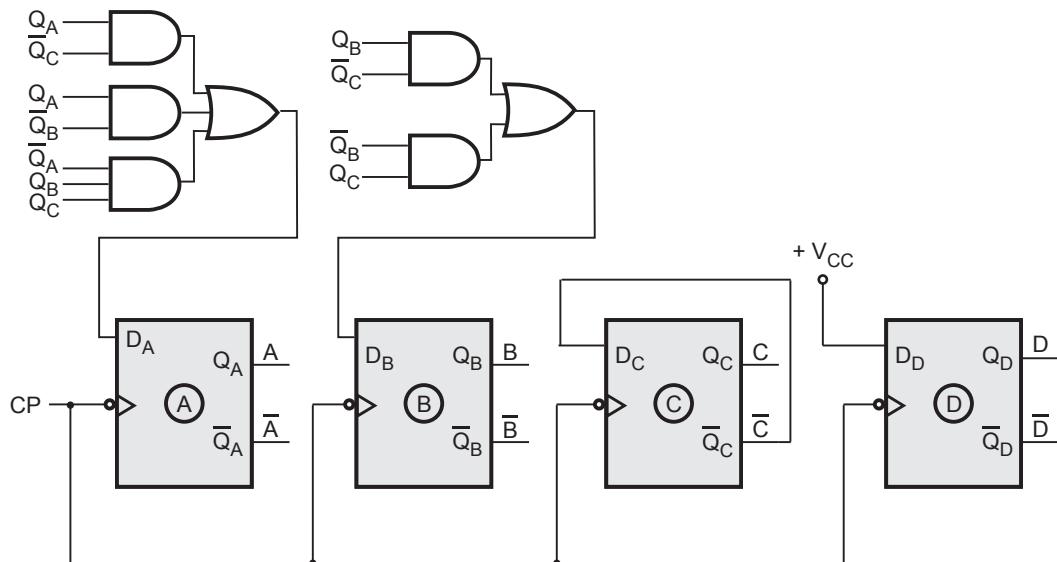
Logic diagram

Fig. 7.2.37

Example 7.2.11 Design a non-sequential counter using J-K flip-flop; as per following state diagram.

SPPU : May-06, Marks 8

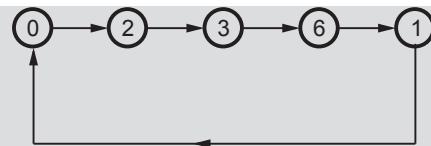


Fig. 7.2.38

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
Q_A	Q_B	Q_C	Q'_A	Q'_B	Q'_C	J_A	K_A	J_B	K_B	J_C	K_C
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	0	1	1	0	x	x	0	1	x
0	1	1	1	1	0	1	x	x	0	x	1
1	0	0	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x
1	1	0	0	0	1	x	1	x	1	1	x
1	1	1	x	x	x	x	x	x	x	x	x

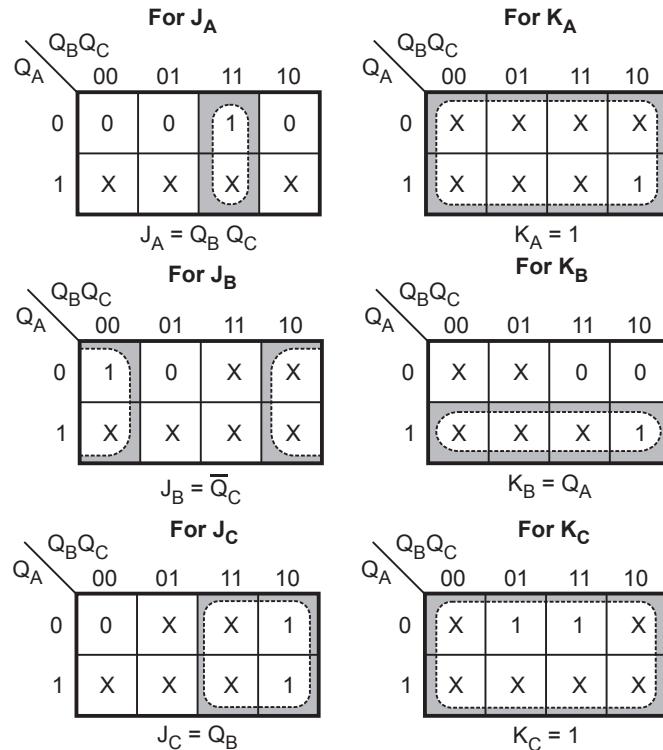
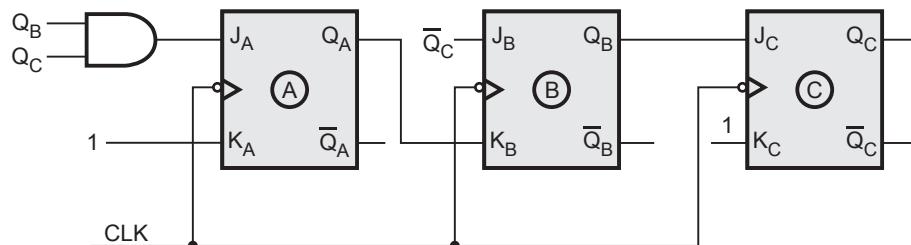
K-map simplification**Logic diagram**

Fig. 7.2.38 (a)

Example 7.2.12 Design the circuit to generate the sequence :

$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$.

SPPU : May-12,13, Dec.-08, Marks 8

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
Q _A	Q _B	Q _C	Q _{A+1}	Q _{B+1}	Q _{C+1}	J _A	K _A	J _B	K _B	J _C	K _C
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	1	0	1	1	X	X	1	1	X

0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	1	1	1	X	0	1	X	1	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	0	1	1	X	1	X	0	X	0

K-map simplification**For J_A**

	Q_B	Q_C	00	01	11	10
0	0	0	0	X	0	1
1	X	4	X	5	X	6

$$J_A = Q_B \bar{Q}_C$$

For K_A

	Q_B	Q_C	00	01	11	10
0	X	0	X	1	X	X
1	0	4	0	5	1	X

$$K_A = Q_B$$

For K_B

	Q_B	Q_C	00	01	11	10
0	1	0	X	1	X	X
1	1	4	0	5	X	X

$$J_B = \bar{Q}_C$$

	Q_B	Q_C	00	01	11	10
0	X	0	X	1	1	2
1	X	4	X	5	0	X

$$K_B = \bar{Q}_A$$

For J_C

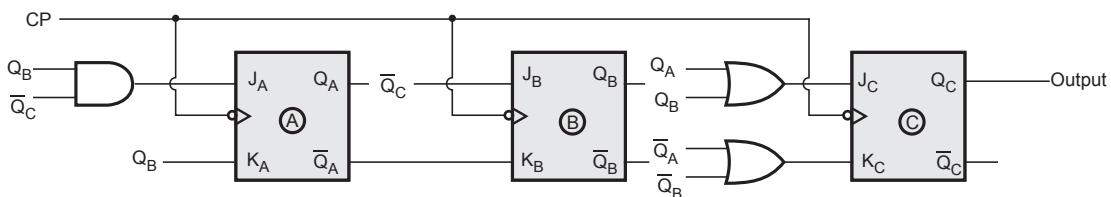
	Q_B	Q_C	00	01	11	10
0	0	0	X	1	X	2
1	1	4	X	5	X	6

$$J_C = Q_A + Q_B$$

For K_C

	Q_B	Q_C	00	01	11	10
0	X	0	X	1	1	2
1	X	4	1	5	0	X

$$K_C = \bar{Q}_A + \bar{Q}_B$$

Fig. 7.2.39 (a)**Logic diagram****Fig. 7.2.39 (b)**

Example 7.2.13 Design sequence generator to generate sequence 1-9-2-7-3-6 using JK flip-flop.

SPPU : Dec.-07, Marks 8

Solution : Excitation table

Present state				Next state				Flip-flop inputs							
Q_A	Q_B	Q_C	Q_D	Q_{A+1}	Q_{B+1}	Q_{C+1}	Q_{D+1}	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x
0	0	0	1	1	0	0	1	1	x	0	x	0	x	x	0
0	0	1	0	0	1	1	1	0	x	1	x	x	0	1	x
0	0	1	1	0	1	1	0	0	x	1	x	x	0	x	1
0	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x
0	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x
0	1	1	0	0	0	0	1	0	x	x	1	x	1	1	x
0	1	1	1	0	0	1	1	0	x	x	1	x	0	x	0
1	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x
1	0	0	1	0	0	1	0	x	1	0	x	1	x	x	1
1	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x

K-map simplification

		For J_A				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	1	0	0
01		X	X	0	0	
11		X	X	X	X	
10		X	X	X	X	

$$J_A = \bar{Q}_C$$

		For K_A				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	X	X	X
01		X	X	X	X	
11		X	X	X	X	
10		X	1	X	X	

$$K_A = 1$$

		For J_B				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	0	1	1
01		X	X	X	X	
11		X	X	X	X	
10		X	0	X	X	

$$J_B = Q_C$$

		For K_B				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	X	X	X
01		X	X	1	1	
11		X	X	X	X	
10		X	X	X	X	

$$K_B = 1$$

		For J_C				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	0	X	X
01		X	X	X	X	
11		X	X	X	X	
10		X	1	X	X	

$$J_C = Q_A$$

		For K_C				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	X	0	0
01		X	X	0	1	
11		X	X	X	X	
10		X	X	X	X	

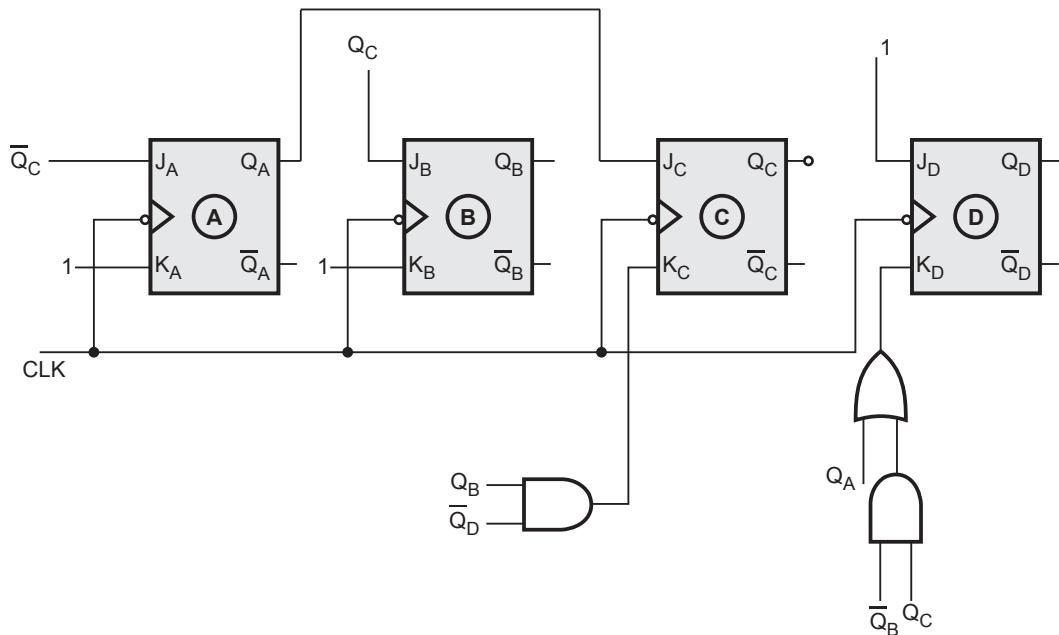
$$K_C = Q_B \bar{Q}_D$$

		For J_D				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	X	X	1
01		X	X	X	1	
11		X	X	X	X	
10		X	X	X	X	

$$J_D = 1$$

		For K_D				
		$Q_C Q_D$	00	01	11	10
$Q_A Q_B$		00	X	0	1	X
01		X	X	0	X	
11		X	X	X	X	
10		X	1	X	X	

$$K_D = Q_A + \bar{Q}_B Q_C$$

Logic diagram**Fig. 7.2.40**

Example 7.2.14 Design given sequence generator using J-K FF. Sequence is

$1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 1$

SPPU : Dec.-14, Marks 6

Solution : Excitation table

P S			N S			Flip-flop Inputs					
A	B	C	A+	B+	C+	J _A	K _A	J _B	K _B	J _C	K _C
0	0	0	x	x	x	x	x	x	x	x	x
0	0	1	0	1	1	0	x	1	x	x	0
0	1	0	x	x	x	x	x	x	x	x	x
0	1	1	1	0	1	1	x	x	1	x	0
1	0	0	x	x	x	x	x	x	x	x	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	1	x	1	x	1	x	0

K-map simplification

For J_A				For K_A				For J_B									
A	BC	00	01	11	10	A	BC	00	01	11	10	A	BC	00	01	11	10
0	X	0	1	X		0	X	X	X	X		0	X	1	X	X	
1	X	X	X	X		1	X	0	1	0		1	X	1	X	X	

$J_A = B$ $K_A = BC$ $J_B = 1$

For K_B				For J_C				For K_C									
A	BC	00	01	11	10	A	BC	00	01	11	10	A	BC	00	01	11	10
0	X	X	1	X		0	X	X	X	X		0	X	0	0	X	
1	X	X	1	0		1	X	X	X	1		1	X	1	0	X	

$K_B = C$ $J_C = 1$ $K_C = A\bar{B}$

Fig. 7.2.41 (a)

Logic diagram

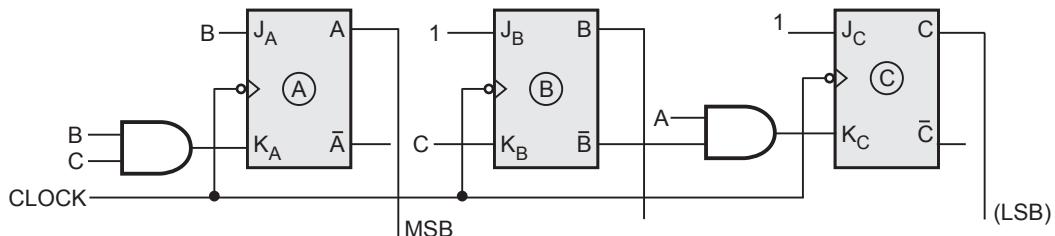


Fig. 7.2.41 (b)

Review Questions

1. Give recommended steps for the design of a clocked synchronous sequential networks.

2. Why is state reduction necessary?

3. Explain the state assignment rules.

SPPU : Dec.-13, Marks 3

4. Explain the procedure of state minimisation using Merger graph and Merger table.

5. Explain state reduction.

SPPU : Dec.-13, Marks 3

6. Explain state assignment.

SPPU : Dec.-12, Marks 12

7. Explain the steps involved in the reduction of state table.

8. Explain : 1) State table 2) State diagram

3) Rules for state reduction 4) State assignment.

SPPU : Dec.-10, May-13, Marks 8

7.3 Sequence Generator

SPPU : May-2000,02,06,08,12,18, Dec.-06,07,08,12,13

7.3.1 Sequence Generator using Counters

A sequential circuit which generates a prescribed sequence of bits, in synchronism with a clock, is referred to as a sequence generator. Fig. 7.3.1 shows the basic structure of a sequence generator using counters.

For the design of sequence generator, we must determine the required number of flip-flops and the logic circuit for next state decoder.

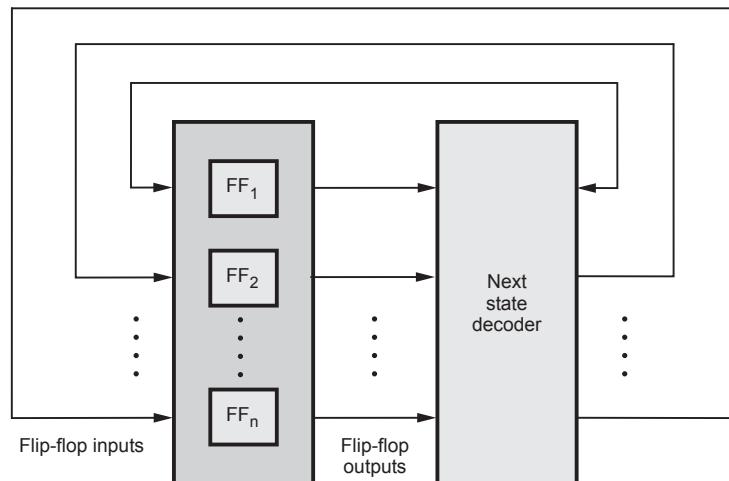


Fig. 7.3.1 Basic structure of a sequence generator

Number of flip-flops required

Number of flip-flops required to generate particular sequence can be determined as follows :

- Find the number of 1s in the sequence.
- Find the number of 0s in the sequence.
- Take the maximum out of two.
- If N is the required number of flip-flops, choose minimum value of n to satisfy equation given below.

$$\max(0s, 1s) \leq 2^n - 1$$

Examples with Solutions

Example 7.3.1 Find the number of flip-flops required to generate the sequence 1101011.

Solution : The given sequence number of 0s are 2 and number of 1s are 5. Therefore equation becomes,

$$\therefore \max(2, 5) \leq 2^n - 1$$

$$\therefore 5 \leq 2^n - 1$$

$$\therefore n = 4$$

Once the number of flip-flops are decided, we have to assign unique states corresponding to each bit in the given sequence such that flip-flop representing least significant bit generates the given sequence. (Usually, the output of the flip-flop representing least significant bit is used to generate the given sequence).

Example 7.3.2 Find the state assignments for sequence 1101011.

Solution : We have already seen that this sequence requires four flip-flops. Assuming the output of D flip-flop as a desired sequence state assignments are shown in Table 7.3.1.

A	B	C	D	States
0	0	0	1	1
0	0	1	1	3
0	0	0	0	0
0	1	0	1	5
0	0	1	0	2
0	1	1	1	7
1	0	0	1	9

Table 7.3.1

Example 7.3.3 Design a pulse-train generator to generate a pulse train 110011..... using 'D' flip-flop.

SPPU : May-06, Marks 8

Solution : The minimum number of flip-flops can be given as $N \leq 2^{n-1}$.

Since max (0s, 1s) = 4, N = 4, therefore n = 3. Table 7.3.2 shows state encoding for given sequence using 3 flip-flops.

Therefore, design a circuit to get above state. In case of D flip-flop, flip-flop inputs are same as next states.

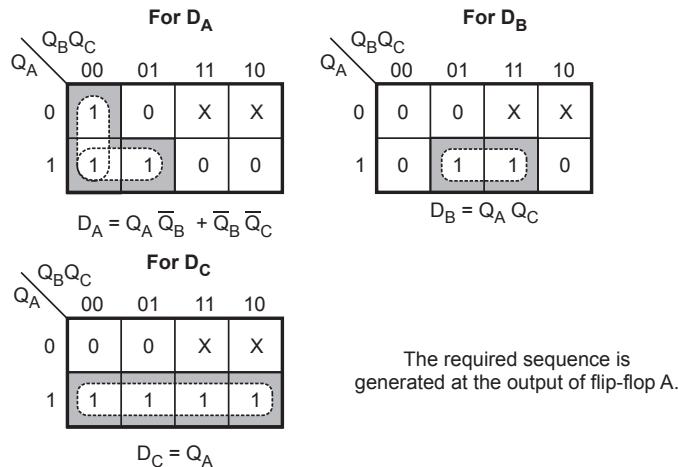
Excitation table

Present state			Next state/Flip-flop inputs		
Q _A	Q _B	Q _C	Q _A	Q _B	Q _C
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	x	x	x
0	1	1	x	x	x
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	1	1

CP	FF O/Ps			States
	Q _A	Q _B	Q _C	
1	1	1	1	7
2	1	1	0	6
3	0	0	1	1
4	0	0	0	0
5	1	0	0	4
6	1	0	1	5

Table 7.3.2

K-map simplification



Logic diagram

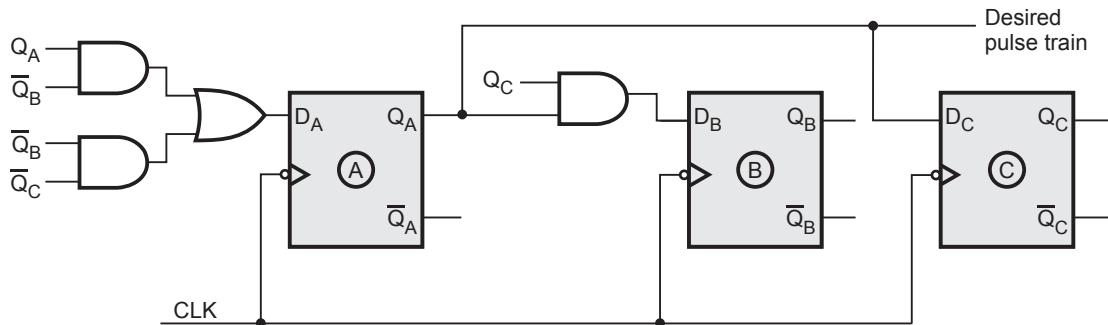


Fig. 7.3.2

7.3.2 Sequence Generator using Shift Register

The simplest way of designing sequence generator using shift register is to take shift register of n -bits where n is equal to the length of sequence. Then load the bit sequence in the shift register by parallel load operation and apply the clock signal.

The Fig. 7.3.3 illustrates the operation of such a sequence generator. Here, the sequence to be generated is 11001.

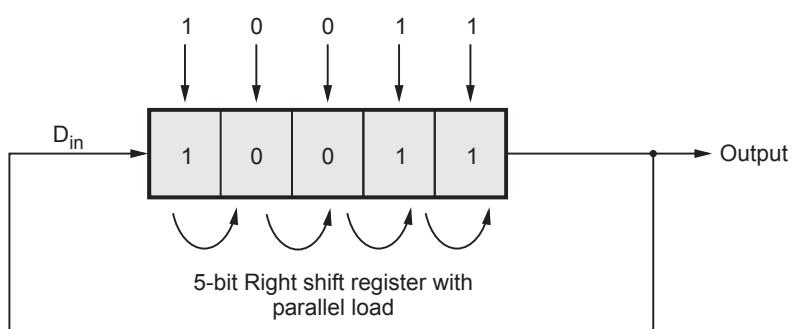


Fig. 7.3.3 Sequence generator

Another design approach is used for sequence generator to reduce the required number of flip-flop stages in the shift register. In this approach a shift register with a next state decoder and preset logic is used. The Fig. 7.3.4 shows the block diagram of this approach. Here, the output of the next state decoder is a function of Q_A, Q_B, \dots, Q_n and it is used to determine the D_{in} input for the shift right register. Initially, start button is pressed to activate parallel load operation. This loads initial value in the shift register. Then at each clock pulse shift register contents are shifted right by 1 bit position. The next state decoder circuit decodes the output and generates D_{in} input for the shift register such that output Q_A generates the desired sequence.

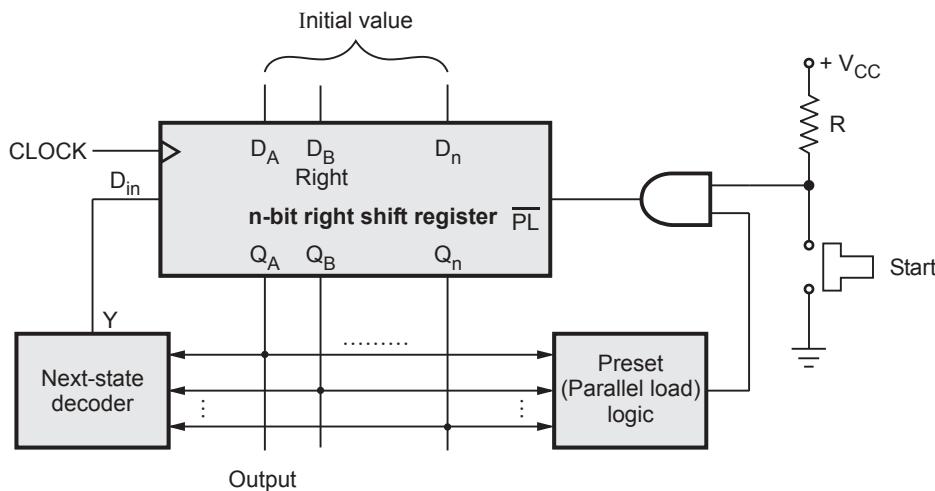


Fig. 7.3.4 Sequence generator using shift register

After completion of one complete sequence, the register is again loaded with initial value to start the next train of sequence.

Examples with Solutions

Example 7.3.4 Design a sequence generator to generate the sequence 1101011 by shift register method. SPPU : May-08, Dec.-08, Marks 8

Solution : In this approach, the minimum number of flip-flops n , required to generate a sequence of length N is given by

$$N \leq 2^n - 1$$

In this example, $N = 7$, therefore, the minimum value of n , which may generate this sequence is 3. However, it is not guaranteed to lead to a solution. Let us try with 3 flip-flops. The Table 7.3.3 shows sequence generation with three flip-flops.

CP	Flip-flop outputs			D _{in}	States
	Q _A	Q _B	Q _C		
1	1	0	0	1	4
2	1	1	0	0	6
3	0	1	1	1	3
4	1	0	1	0	5
5	0	1	0	1	2
6	1	0	1	—	5
—	—	—	—	—	—

State is repeated →

Table 7.3.3

As shown in the Table 7.3.4, the state 6 is repeated. This means that n = 3 is not sufficient. Let us try with 4 flip-flops. The table shows sequence generation with four flip-flops.

CP	Flip-flop outputs				D _{in}	Preset	States
	Q _A	Q _B	Q _C	Q _D			
Initial value	1	1	0	0	0	1	8
	2	1	1	0	0	0	12
	3	0	1	1	0	1	6
	4	1	0	1	1	0	11
	5	0	1	0	1	1	5
	6	1	0	1	0	1	10
Preset flip-flop	7	1	1	0	1	1	13
	1	1	1	0	X	0	14
	1	1	0	0	1	1	8

Table 7.3.4

K-map simplification for D_{in}

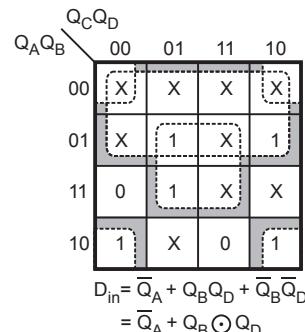


Fig. 7.3.5 (a)

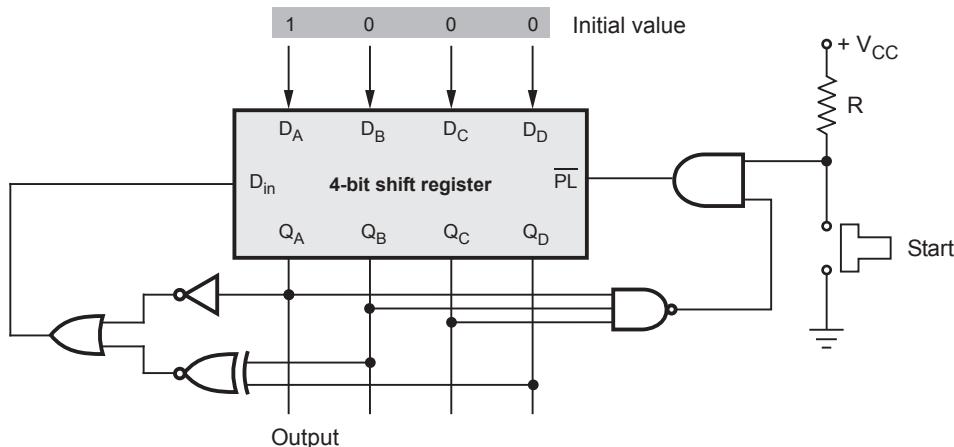
Logic diagram

Fig. 7.3.5 (b)

Example 7.3.5 Design and implement the following sequence generator using shift register
1010

SPPU : Dec.-06, May-18, Marks 4

Solution : Here the sequence is 10 and it is repeated. Thus the length of the sequence N = 2. Let the number of flip-flops required n be.

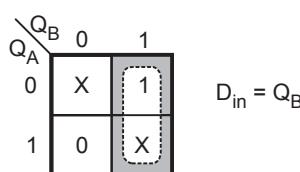
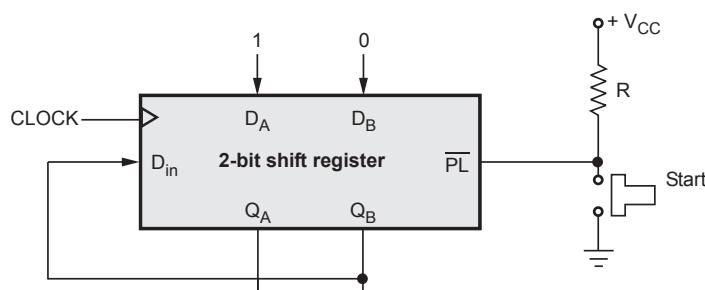
$$N \leq 2^n - 1$$

$$2 \leq 2^n - 1$$

$$\therefore n = 2$$

CP	Flip-flop outputs		D _{in}	States
	Q _A	Q _B		
1	1	0	0	2
2	0	1	1	1
1	1	0	0	2

Fig. 7.3.6

K-map simplification for D_{in}**Logic diagram**

(a)

(b)

Fig. 7.3.6

In the above circuit preset/parallel load logic is only required to load the initial value. After completion of one sequence, the shift register have the same initial value and hence it is not required to reload again.

Example 7.3.6 Design a pulse train generator using shift register for the following pulse train.

.....1 1 1 0 1 0

SPPU : May-2000, 02, Marks 8

Solution : The minimum number of flip-flops n can be given as

$$N \leq 2^n - 1$$

Here $N = 6$, therefore $n = 3$. The Table 7.3.5 shows sequence generation with three flip-flops.

CP	Flip-flop outputs			D_{in}	<u>Preset</u>	States
	Q_A	Q_B	Q_C			
1	1	0	0	1	1	4
2	1	1	0	1	1	6
3	1	1	1	0	1	7
4	0	1	1	1	1	3
5	1	0	1	0	1	5
6	0	1	0	0	1	2
Preset flip-flop	7	0	0	1	X	0
	1	1	0	0	1	4

Table 7.3.5

K-map simplification for D_{in}

$$\begin{aligned} D_{in} &= \overline{Q}_A Q_C + Q_A \overline{Q}_C \\ &= Q_A \oplus Q_C \end{aligned}$$

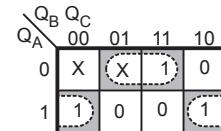


Fig. 7.3.7 (a)

Logic diagram

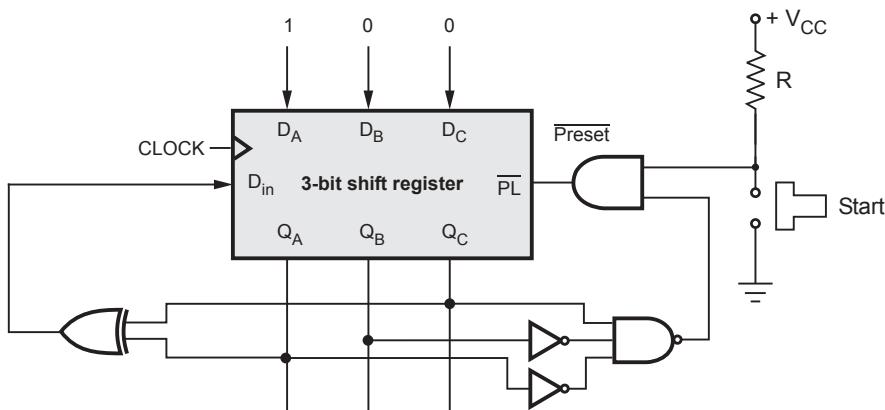


Fig. 7.3.7 (b)

Example 7.3.7 Design a pulse train generator circuit using shift register for the following pulse train.

..... 1 0 0 0 1 1 0

SPPU : Dec.-08, May-12, Marks 8

Solution : The minimum number of flip-flops n , required to generate sequence of length N is given by

$$N \leq 2^n - 1$$

Here, $N = 7$, therefore $n = 3$. The Table 7.3.6 (a) shows sequence generation with three flip-flops.

CP	Flip-flop outputs			D_{in}	States
	Q_A	Q_B	Q_C		
1	1	1	1	0	7
2	0	1	1	0	3
3	0	0	1	0	1
4	0	0	0	1	0
5	1	0	0	1	4
6	1	1	0	0	6
State is repeated	7	0	1	—	3

Table 7.3.6 (a)

As seventh state is repeated, $n = 3$ is not sufficient. Let us try with 4 flip-flops. The Table 7.3.6 (b) shows sequence generation with four flip-flops.

CP	Flip-flop outputs				D_{in}	$\overline{\text{Preset}}$	States
	Q_A	Q_B	Q_C	Q_D			
1	1	1	1	1	0	1	15
2	0	1	1	1	0	1	7
3	0	0	1	1	0	1	3
4	0	0	0	1	1	1	1
5	1	0	0	1	1	1	9
6	1	1	0	0	0	1	12
7	0	1	1	0	1	1	6
Preset flip-flop	1	1	0	1	1	0	11
	1	1	1	1	0	1	15

Table 7.3.6 (b)

K-map simplification for D_{in}

$$\therefore D_{in} = \overline{Q_A} \overline{Q_C} + Q_A \overline{Q_B} + Q_C \overline{Q_D}$$

$Q_A Q_B$	00	01	11	10
$Q_C Q_D$	00	1	0	X
00	X	X	0	1
01	X	X	0	1
11	0	X	0	X
10	X	1	X	X

Logic diagram

Fig. 7.3.8

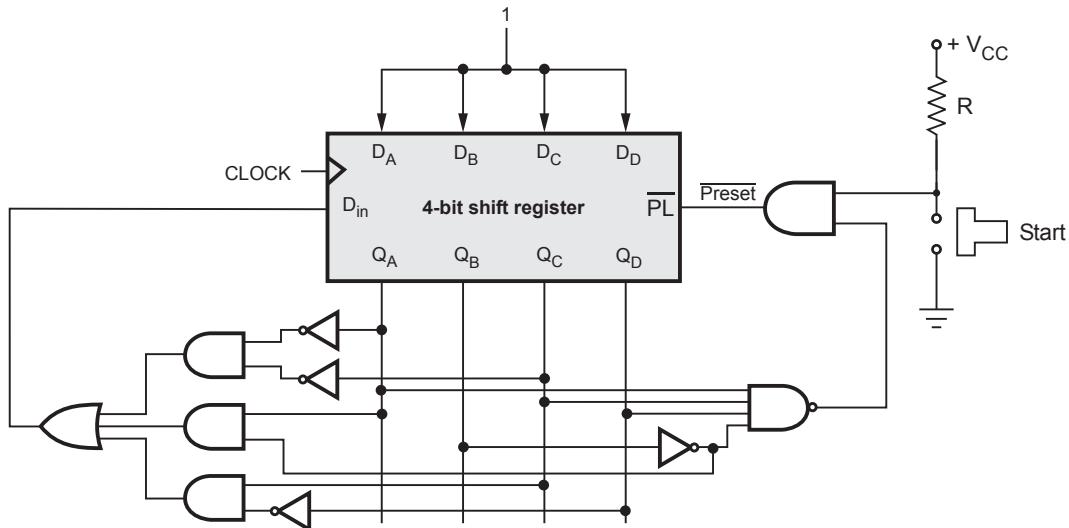


Fig. 7.3.9

Example 7.3.8 Design sequence generator using shift register to generate sequence 1101.

SPPU : Dec.-07, Marks 4

Solution : The minimum number of flip-flops n can be given as

$$N \leq 2^n - 1$$

Here $N = 4$, therefore $n = 3$. The Table 7.3.7 shows sequence generation with three flip-flops.

Preset flip-flop

CP	Flip-flop inputs			D_{in}	\overline{PL}	State s
	Q_A	Q_B	Q_C			
1	1	0	0	1	1	4
2	1	1	0	0	1	6
3	0	1	1	1	1	3
4	1	0	1	0	1	5
5	0	1	0	x	0	2
1	1	0	0	1	1	4

Table 7.3.7

K-map simplification for D_{in}

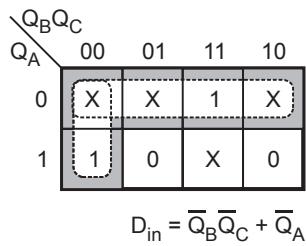


Fig. 7.3.10 (a)

Logic diagram

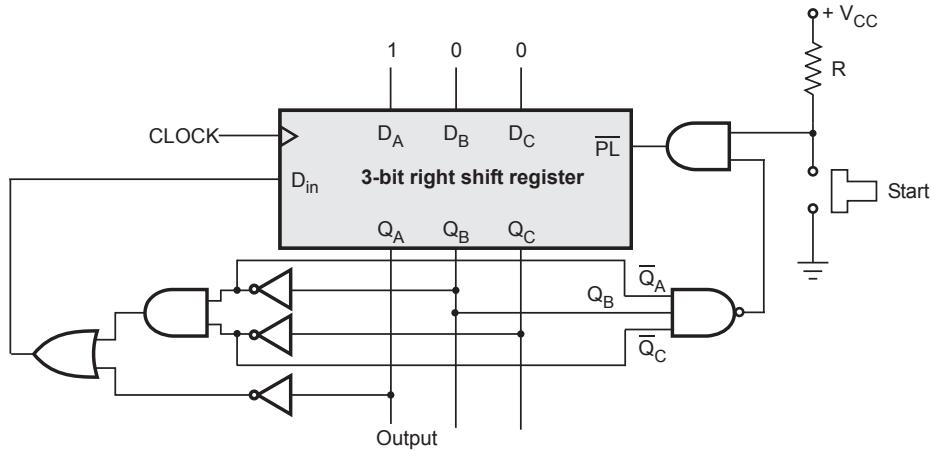


Fig. 7.3.10 (b)

Example for Practice

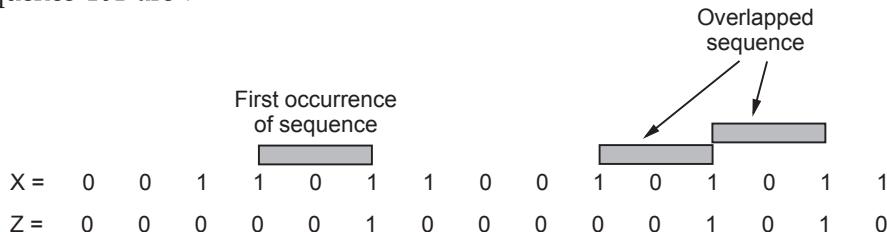
Example 7.3.9 Design pulse train generator using shift register to generate the following pulse
..... 10110.....

SPPU : Dec.-12,13, Marks 8

7.4 Sequence Detector

SPPU : May-06,07,10,12,13,14, Dec.-05,06,07,08,10,12,13,15,16

The specified input sequence can be detected using a sequential machine called **sequence detector**. In this circuit output goes high when a prescribed input sequence occurs. A typical input sequence and the corresponding output sequence for desired input sequence 101 are :



As shown above the detection of required input sequence can occur in a longer data string and the desired input sequence can overlap with another input sequence. It is assumed that input can change only between clock pulses. Once we know the sequence which is to be detected, we can draw the state diagram for it. Then from the state diagram we can design the circuit. It is possible to implement sequence detector using both types of sequential machines : Mealy machine and Moore machine. The following examples illustrate how to determine the state diagram from the given input sequence and then implement the sequence detector.

Illustrative Examples

Example 7.4.1 Design a Mealy type sequence detector to detect a serial input sequence of 101.

SPPU : Dec.-13, Marks 6

Solution : In a Mealy type design, the number of states in the state diagram are equal to the number of bits in the desired input sequence. In this example, we have three bits in the sequence so we have three states in the state diagram. Once the number of states are known we have to draw the direction lines with states of inputs and outputs. Let us start drawing direction lines, assuming state 'a' as an initial state.

1. State a

State a checks for 1. When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence.

When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0 since we have not yet detected all the bits in the sequence.

2. State b

When input is 0, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 1, we have to remain in state b because 1 which we have detected may start the sequence. Output is still zero.

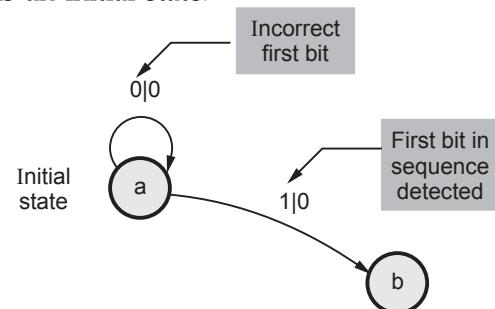


Fig. 7.4.1 (a)

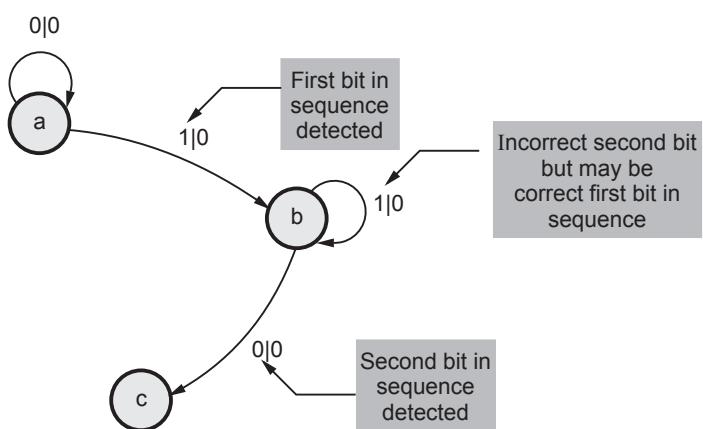


Fig. 7.4.1 (b)

3. State c

As explained for state a and state b, if desired bit is detected we have to go for next state otherwise we have to go to the previous state from where we can continue the desired sequence. When complete sequence is detected we have to make output HIGH and go to the initial state. This is illustrated in Fig. 7.4.1 (c).

From the above state diagram we can determine the state table as shown in Table 7.4.1 (a).

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	b	0	0
c	a	b	0	1

Table 7.4.1 (a) State table

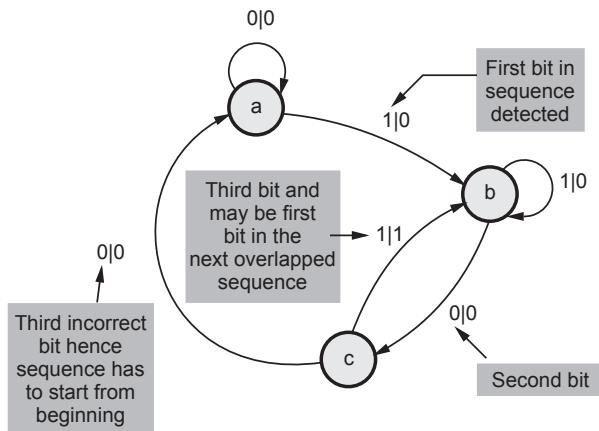


Fig. 7.4.1 (c)

Present state	Next state		Output Z				
	A	B	A ⁺	B ⁺	X = 0	X = 1	
a	0	0	0	0	1	0	0
b	0	1	1	0	0	1	0
c	1	0	0	0	1	0	1

Table 7.4.1 (b) Excitation table

Since there are three states we need two flip-flops. Assigning state a = 00, state b = 01 and state c = 10 we can determine the excitation table as shown in Table 7.4.1 (b).

K-map simplification

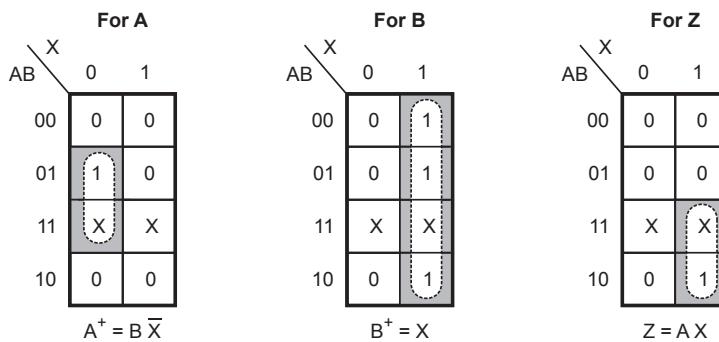


Fig. 7.4.2 (a)

Logic diagram

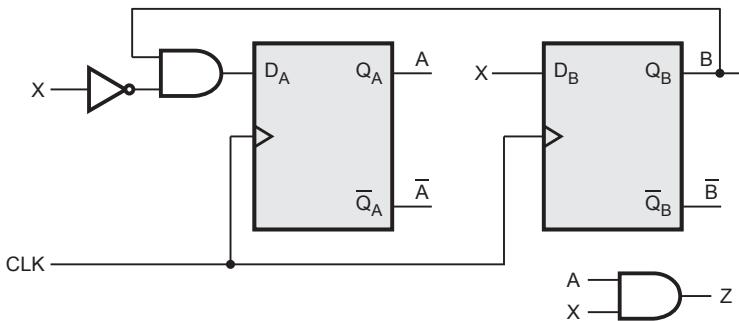


Fig. 7.4.2 (b)

Example 7.4.2 Design a Moore type sequence detector to detect a serial input sequence of 101.

SPPU : Dec.-13, Marks 6

Solution : In a Moore type design output depends only on flip-flop states. Therefore, in the state diagram of Moore machine, the output is written with the state instead of with the transition between the states. Apart from this the process of determining the state diagram is similar to that used for Mealy machine. Let us start with state 'a' as an initial state.

1. State a : When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence.

When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0 since we have not yet detected all the bits in the sequence.

2. State b : When input is 0, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 1, we have to remain in state b because 1 which we have detected may start the sequence. Output is still zero.

3. State c : Now, when a 1 input occurs, the 101 sequence is completed and output must equal to 1. Here, we

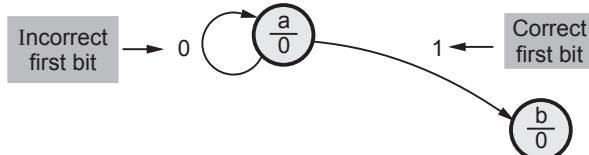


Fig. 7.4.3 (a)

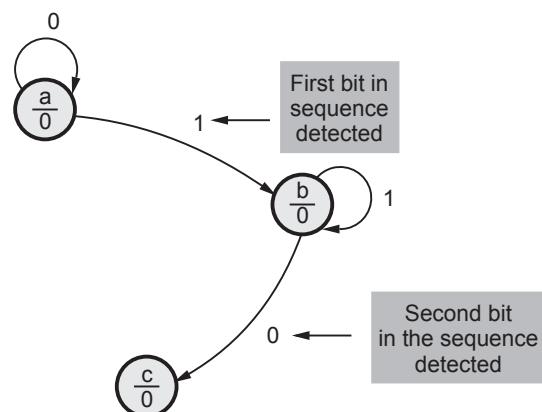


Fig. 7.4.3 (b)

cannot go back to state b (since output in state b is zero) and hence we have to create new state d with a output 1.

In case if input is zero, we have to restart checking of input sequence and hence we have to return to state a.

4. State d : Since the sequence is detected, this is the last state. When input is 1, we have detected the first bit in the next sequence, hence we have to go to state b.

When input is 0, we have detected the second bit in the overlapped sequence, hence we have to go to state c.

From the above state diagram we can determine the state table as shown in Table 7.4.2 (a).

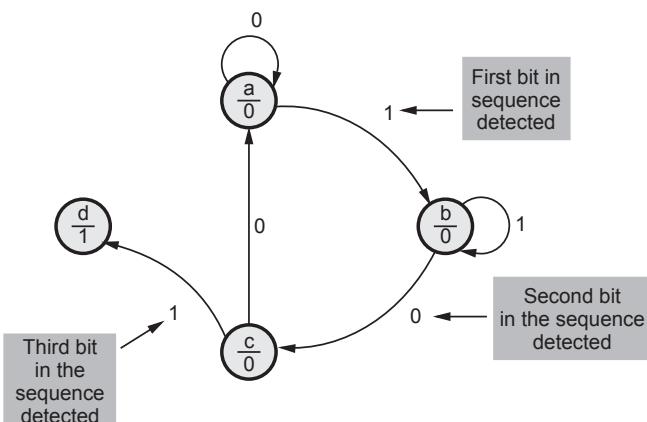


Fig. 7.4.3 (c)

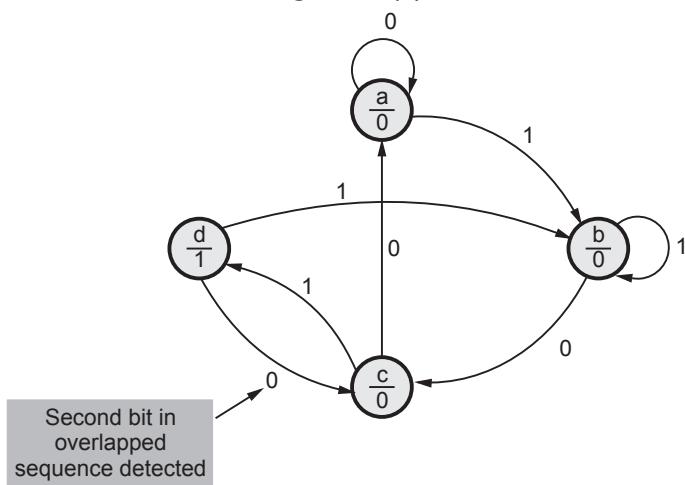


Fig. 7.4.3 (d)

Present state	Next state		Output Z
	X = 0	X = 1	
a	a	b	0
b	c	b	0
c	a	d	0
d	c	b	1

Table 7.4.2 (a) State table

Present state	Next state A ⁺ B ⁺		Output				
	A	B	X = 0	X = 1			
0	0	0	0	0	1	0	
0	1	1	1	0	0	1	0
1	0	0	0	1	1	0	
1	1	1	1	0	0	1	1

Table 7.4.2 (b) Excitation table

Since there are four states we need two flip-flops. Assigning state a = 00, b = 01, c = 10 and d = 11 we can determine the excitation table as shown in Table 7.4.2 (b).

K-map simplification

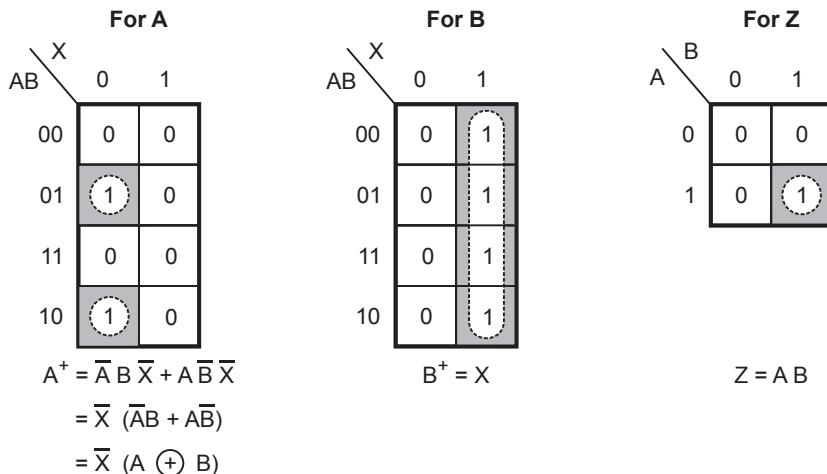


Fig. 7.4.4

Logic diagram

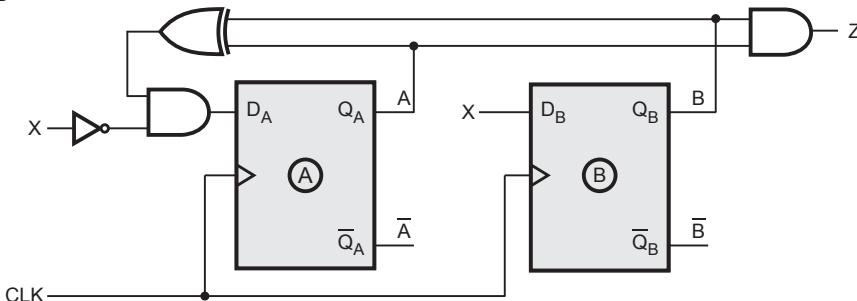


Fig. 7.4.5

Examples with Solutions

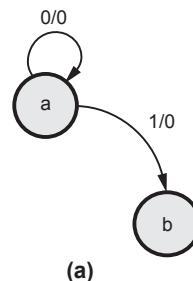
Example 7.4.3 Design a sequence detector to detect the following sequence using JK flip-flops.

(Use Mealy Machine) ... 110

SPPU : May-07,14, Dec.-08, Marks 4

Solution : The given sequence has 3-bit, so we require 3 states in the state diagram. Let us start drawing direction lines, assuming state 'a' is an initial state.

State a : When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0, since we have not yet detected all the bits in the sequence.



State b : When input is 1, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 0, we have to go to the state 'a' to detect the first bit in the sequence, i.e. 1.

State c : When input is 0, we have detected the last bit in the sequence, hence we have to go to the initial state, to detect the next sequence and make output high indicating sequence is detected. When input is 1, we have to go to the state 'b' because 1 which we have detected may start the sequence.

Assuming state assignments as $a = 00$, $b = 01$ and $c = 10$, we can determine excitation table for above state diagram as shown in Table 7.4.3.

Input	Present state		Next state		Output	Flip-flop inputs				
	X	A_n	B_n	A_{n+1}	B_{n+1}	J_A	K_A	J_B	K_B	
0	0	0	0	0	0	0	X	0	X	
0	0	1	0	0	0	0	X	X	1	
0	1	0	0	0	1	X	1	0	X	
1	0	0	0	1	0	0	X	1	X	
1	0	1	1	0	0	1	X	X	1	
1	1	0	0	1	0	X	1	1	X	

Table 7.4.3 Excitation table

K-map simplification

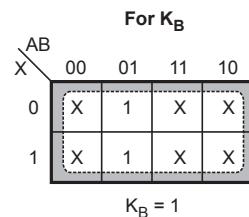
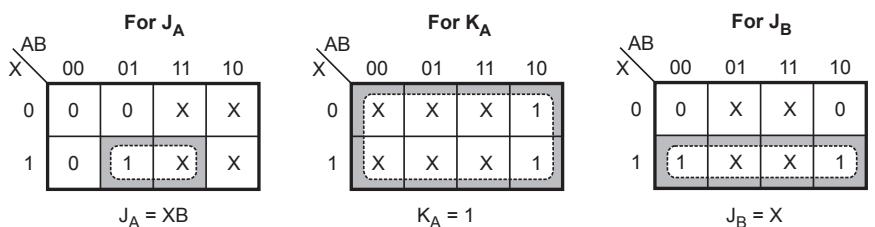


Fig. 7.4.7

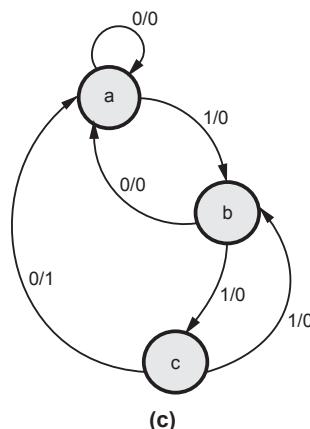
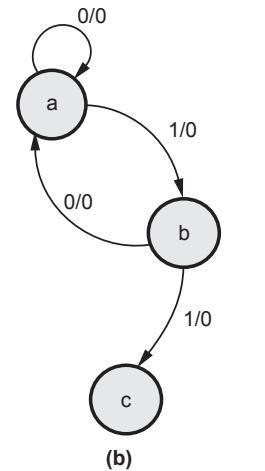


Fig. 7.4.6 State diagram

Logic diagram

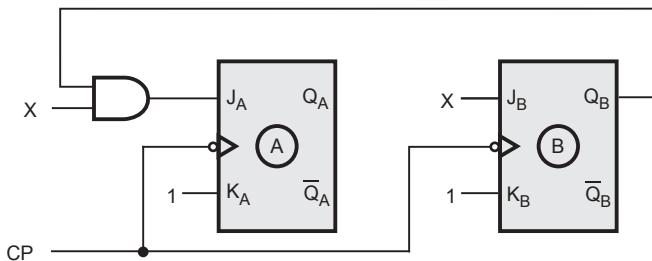


Fig. 7.4.8

Example 7.4.4 Design a sequence detector to detect sequence 1101. **SPPU : Dec.-07, Marks 8**

Solution : State diagram

In Moore model, output depends only on the present state and not on the input, so state diagram is as shown in the Fig. 6.4.9.

State table

Present state	Next state		Output
	X = 0	X = 1	
a	a	b	0
b	a	c	0
c	d	c	0
d	a	e	0
e	a	b	1

Table 7.4.4

State assignment

$$a = 000, \quad b = 001, \quad c = 010, \quad d = 011 \quad \text{and} \quad e = 100$$

State transition table

Present state	Input			Next state			Output	
	Q _A	Q _B	Q _C	X	Q _{A+}	Q _{B+}	Q _{C+}	
0 0 0	0	0	0	0	0	0	0	0
0 0 0	0	0	0	1	0	0	1	0
0 0 1	0	0	1	0	0	0	0	0
0 0 1	0	0	1	1	0	1	0	0

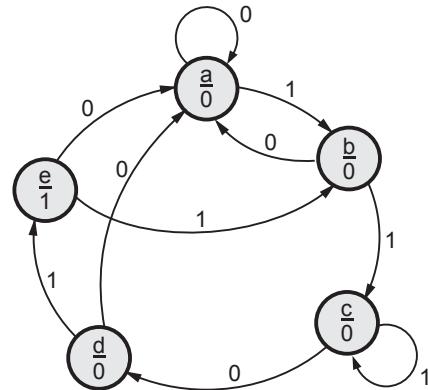
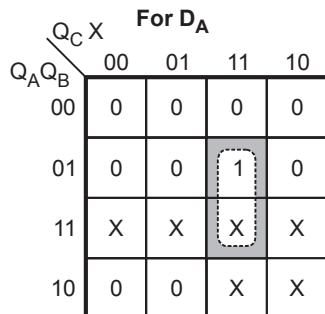


Fig. 7.4.9

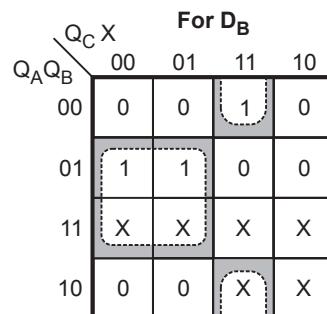
0	1	0	0	0	1	1	0
0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

Table 7.4.5

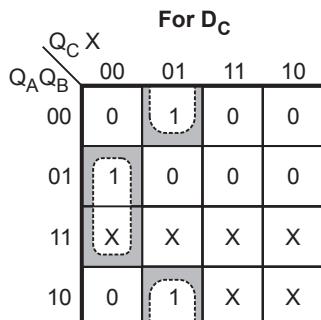
K-map simplification



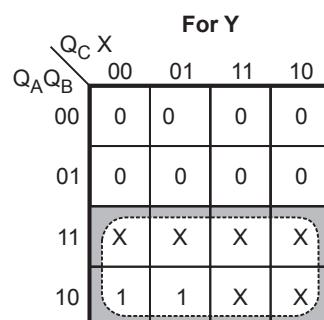
$$D_A = Q_B Q_C X$$



$$D_B = Q_B \bar{Q}_C + \bar{Q}_B Q_C X$$



$$D_C = Q_B \bar{Q}_C \bar{X} + \bar{Q}_B \bar{Q}_C X$$



$$Y = Q_A$$

Fig. 7.4.10

Logic diagram

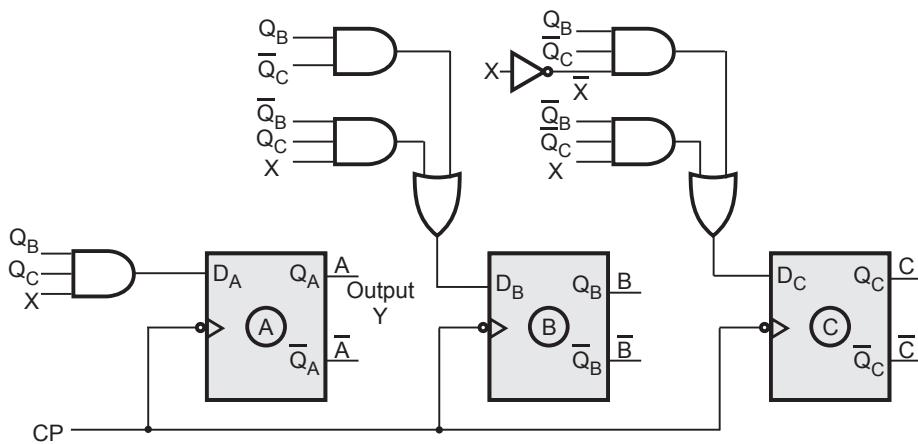


Fig. 7.4.11

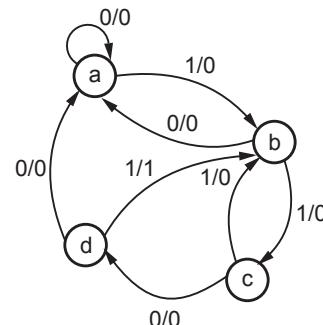
Example 7.4.5 Design a sequence detector using D-FFs to detect the following sequence based on Mealy machine : 1101.

SPPU : May-12, Marks 8

Solution : Sequence detector (1101) using mealy machine and D F/F :

State table :

Present state	Next state		Output Y	
	X = 0	X = 1		
a	a	b	0	0
b	a	c	0	0
c	d	b	0	0
d	a	b	0	1



Excitation table

Present state		Next State				Output	Y
		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	0	0	1	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	1

K-map simplification

		For D_A				
		AB	00	01	11	10
X	0	0	0	0	(1)	
	1	0	(1)	0	0	

$$D_A = Q_A \bar{Q}_B \bar{X} + \bar{Q}_A Q_B X$$

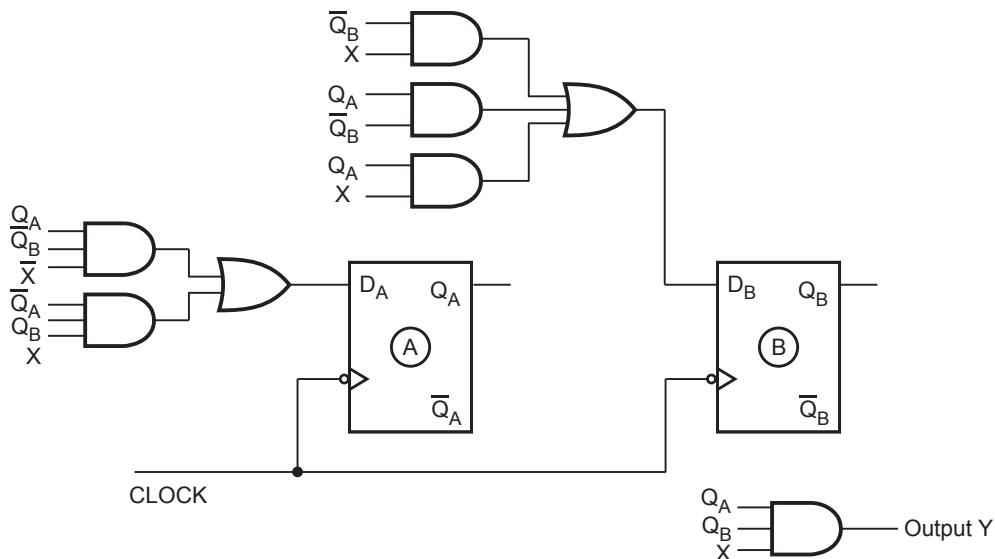
		For D_B				
		AB	00	01	11	10
X	0	0	0	0	(1)	
	1	1	0	1	1	

$$D_B = \bar{Q}_B X + Q_A X \quad Q_A \bar{Q}_B$$

		For Output Y				
		AB	00	01	11	10
X	0	0	0	0	0	
	1	0	0	(1)	0	

$$D_B = Q_A Q_B X$$

Implementation



Example 7.4.6 Design a sequential circuit using Mealy machine for detecting the sequence 1011.....

Use JK flip-flop.

SPPU : Dec.-12, Marks 8

Solution : State diagram

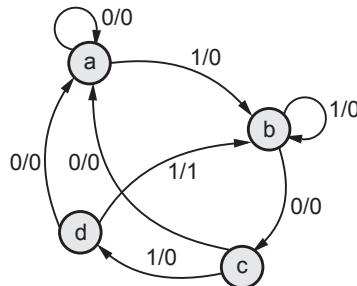
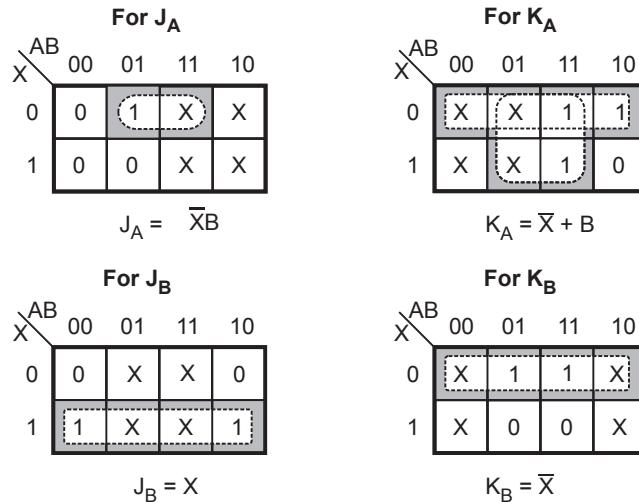
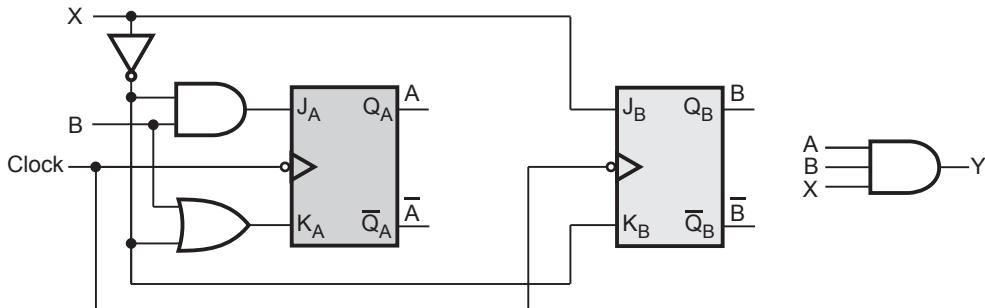


Fig. 7.4.12

State Table :

Input X	Present State		Next State		Output Y	Flip-flop inputs			
	A	B	A_+	B_+		J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	1	1	0	0	1	X	X	1
0	1	0	0	0	0	x	1	0	X
0	1	1	0	0	0	x	1	X	1
1	0	0	0	1	0	0	X	1	X
1	0	1	0	1	0	0	X	X	0
1	1	0	1	1	0	X	0	1	X
1	1	1	0	1	1	X	1	X	0

K-map Simplification**Fig. 7.4.13****Logic diagram****Fig. 7.4.14**

Example 7.4.7 Design a SEQUENCE DETECTOR using JK-flip-flops to detect the following sequence

..... 1 0 0 1

Use state diagram, state transition tables and K-map as design tools. Remove all redundant states and draw the final circuit diagram.

SPPU : Dec.-05.15, Marks 12

Solution : State table :

Present State	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	b	0	0
c	d	b	0	0
d	a	b	0	1

Let a = 00, b = 01, c = 10 and d = 11

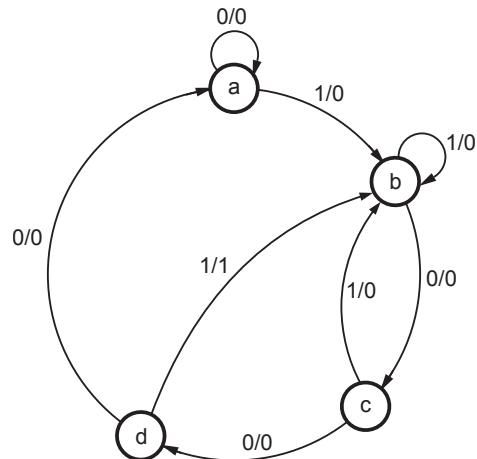
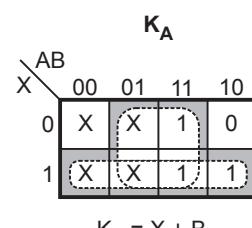
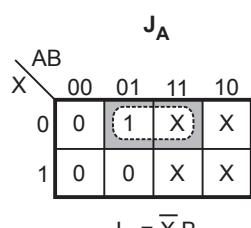


Fig. 7.4.15

Input	Present state		Next state		Flip-flop inputs				
	X	A	B	A ⁺	B ⁺	J _A	K _A	J _B	K _B
0	0	0	0	0	0	0	X	0	X
0	0	1	1	1	0	1	X	X	1
0	1	0	1	1	1	X	0	1	X
0	1	1	0	0	0	X	1	X	1
1	0	0	0	0	1	0	X	1	X
1	0	1	0	0	1	0	X	X	0
1	1	0	0	0	1	X	1	1	X
1	1	1	0	0	1	X	1	X	0

K-map simplification



		J_B						K_B					
		AB	00	01	11	10			AB	00	01	11	10
X	AB	0	0	X	X	1	X	AB	0	X	1	1	X
1	1	X	X	X	X	1	1	AB	1	X	0	0	X

$J_B = X + A$ $K_B = \bar{X}$

Output = XAB

Logic diagram :

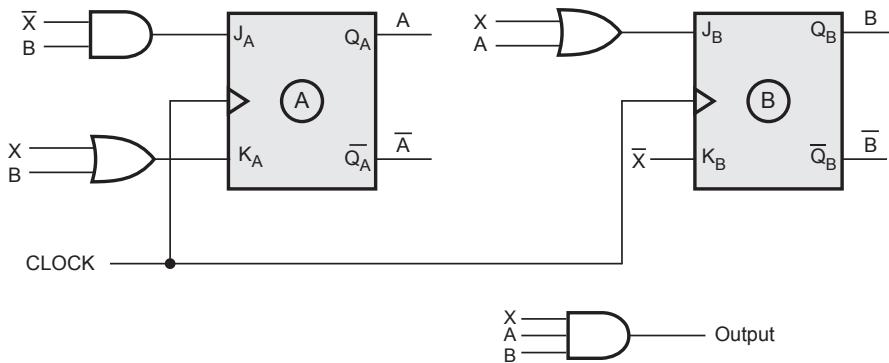


Fig. 7.4.16

Example 7.4.8 Design SEQUENCE DETECTOR using JK flip-flops to detect the following sequence :1 1 1....

Use state diagram, state transition tables and K-map as design tools. Remove all redundant states and draw the final circuit diagram.

SPPU : May-06, Marks 12

Solution : State assignment :

State name	Binary state
a	0 0
b	0 1
c	1 0

State diagram :

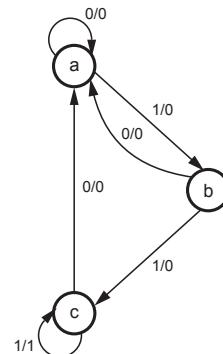


Fig. 7.4.17

State Transition Table :

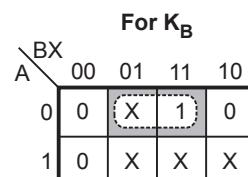
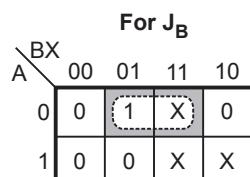
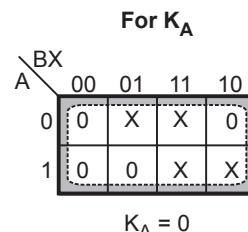
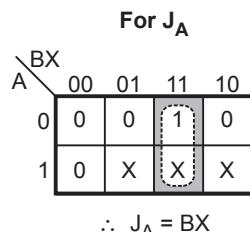
Present State		I/P	Next State		O/P
A	B	X	A+	B+	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	1

Excitation Table of JK flip-flop

Q o/p	Next state	I/P	
		J	K
Present state			
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation Table of our circuit :

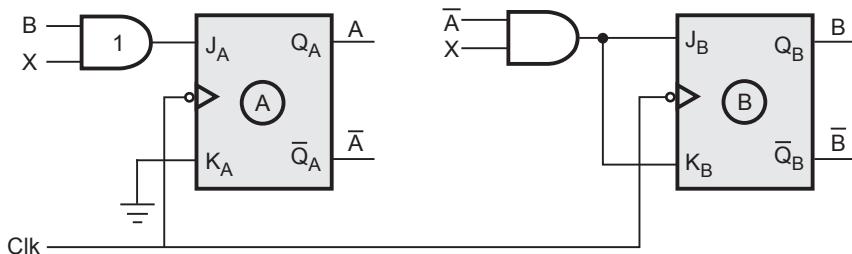
Present State	I/P	Next state	Flip-flop I/Ps								
			A	B	X	A+	B+	J _A	K _A	J _B	K _B
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	X	1	X	X
0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	1	X	X	1	1
1	0	0	0	0	0	0	0	0	0	0	0

K-map

$$J_B = \bar{A}X$$

$$K_B = \bar{A}X$$

Fig. 7.4.18

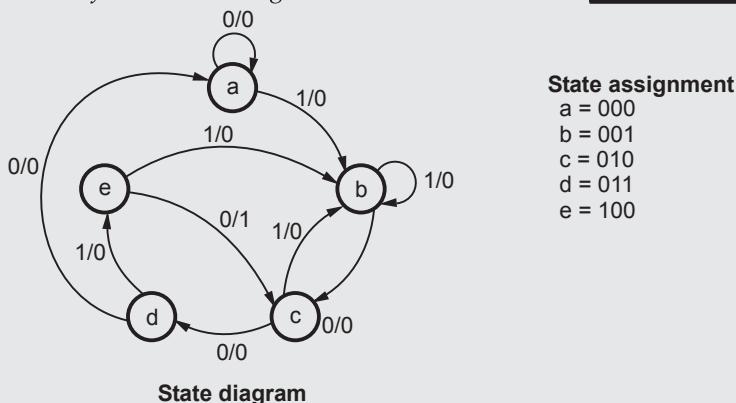
Logic Diagram**Fig. 7.4.19**

Example 7.4.9 Design a SEQUENCE DETECTOR using JK flip-flops to detect the following sequence :

----- 10010 -----

Use state diagram, state transition tables and K-map as design tools. Remove all redundant states and draw the final circuit diagram.

SPPU : Dec.-06, Marks 12

**Fig. 7.4.20****Solution :**

Input	Present state			Next state			Output	Flip-Flop Inputs						
	X	A	B	C	A ₊	B ₊	C ₊	J _A	K _A	J _B	K _B	J _C	K _C	
0	0	0	0	0	0	0	0	0	0	X	0	X	0	X
0	0	0	1	0	0	1	0	0	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	0	X	X	0	1	X
0	0	1	1	0	0	0	0	0	0	X	X	1	X	1
0	1	0	0	0	0	1	0	1	X	1	1	X	0	X
1	0	0	0	0	0	0	1	0	0	X	0	X	1	X
1	0	0	1	0	0	0	1	0	0	X	0	X	X	0
1	0	1	0	0	0	0	1	0	0	X	X	1	1	X
1	0	1	1	0	1	0	0	0	1	X	X	1	X	1
1	1	0	0	0	0	0	1	0	X	1	0	X	1	X

		For J_A						For K_A								
		XA	BC	00	01	11	10	XA	BC	00	01	11	10	XA	BC	
		00	00	0	0	0	0	00	00	X	X	X	X	00	00	
		01	X	4	5	X	X	01	1	4	X	5	X	01	1	
		11	X	12	X	X	15	11	11	12	X	13	X	11	12	
		10	0	8	0	9	1	10	0	8	X	9	X	10	0	
		$J_A = XBC$								$K_A = \bar{B}$						
		For J_B						For K_B								
		XA	BC	00	01	11	10	XA	BC	00	01	11	10	XA	BC	
		00	00	0	1	X	X	00	X	0	X	1	3	00	00	
		01	1	4	5	X	X	01	X	4	X	5	X	01	1	
		11	0	12	X	X	15	11	11	X	12	X	X	11	12	
		10	0	8	0	9	X	11	0	8	X	9	1	11	0	
		$J_B = \bar{X}A + \bar{X}C$								$K_B = C + X$						
		For J_C						For K_C								
		XA	BC	00	01	11	10	XA	BC	00	01	11	10	XA	BC	
		00	00	0	X	1	X	00	X	0	1	1	3	00	00	
		01	0	4	X	5	X	01	X	4	X	5	X	01	1	
		11	1	12	X	X	X	11	X	12	X	13	X	11	12	
		10	1	8	X	X	1	10	0	8	X	9	1	11	0	
		$J_C = X + B$								$K_C = \bar{X} + B$						
		For Y														
		XA	BC	00	01	11	10									
		00	00	0	0	0	0									
		01	1	4	X	X	X									
		11	0	12	X	X	X									
		10	0	8	0	9	0									
																$Y = \bar{X}A$

Fig. 7.4.21

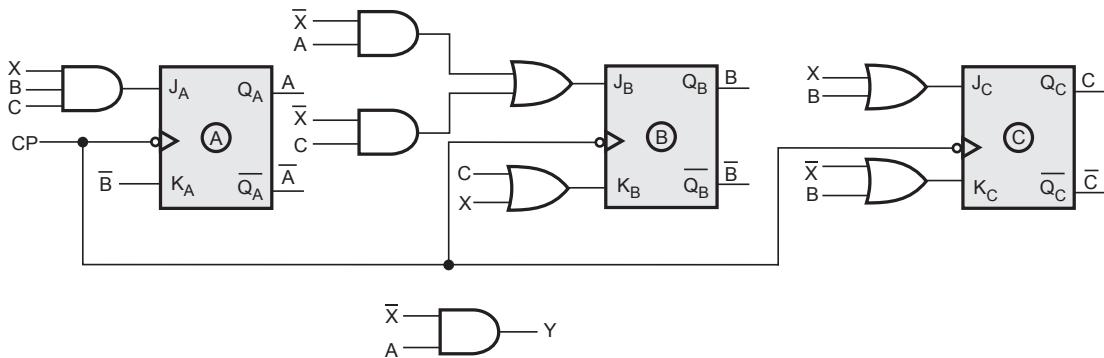


Fig. 7.4.22

Examples for Practice

Example 7.4.10 Design a sequential circuit using Mealy machine for detecting overlapping sequence 1101. Use JK flip-flops. **SPPU : May-10, Dec.-16, Marks 8**

(Ans. : Refer example 7.4.3 and apply similar procedure.)

Example 7.4.11 Design a sequence detector to detect a sequence 1101 (Using D FF and Mealy circuit). **SPPU : Dec.-10, Marks 8**

(Ans. : Refer similar example 7.4.1)

Example 7.4.12 Design and implement 1011 sequence detector using Mealy machine.

(Ans. : Refer example 7.4.3 and apply similar procedure.)

Example 7.4.13 Design a sequential circuit using Mealy machine for detecting the sequence ... 1001... Use JK Flip-flop **SPPU : May-13, Marks 10**

(Ans. : Refer example 7.4.3 and apply similar procedure.)



Notes

UNIT - IV

8

Algorithmic State Machines

Syllabus

Finite State Machines (FSM) and ASM, ASM charts, notations, construction of ASM chart and realization for sequential circuits.

Contents

8.1	<i>Introduction</i>	<i>Dec.-08,10,11,13,15,16,</i>	
		<i>May-09,11,12,13,15,16,18,19,</i>	
			<i>Marks 8</i>
8.2	<i>Design of Simple Controller</i>	<i>Dec.-08,15, May-09,11,12,13,19,</i>	
			<i>Marks 8</i>
8.3	<i>ASM Examples : Sequence Generator, Types of Counters</i>	<i>Dec.-98,06,11,12,13,16,17,18,19,</i>	
		<i>May-2000,05,06,07,08,12,14,16,17</i>	
			<i>Marks 16</i>

8.1 Introduction**SPPU : Dec.-08,10,11,13,15,16, May-09,11,12,13,15,16,18,19**

The main components of digital system are : Control logic and datapath. The relationship between these two is shown in the Fig. 8.1.1. The data processing path, commonly known as a **datapath**, manipulates data in registers according to system's requirements. The control logic initiates a sequence of commands to the datapath. It uses status conditions from the datapath as decision variables for determining the sequence of control signals.

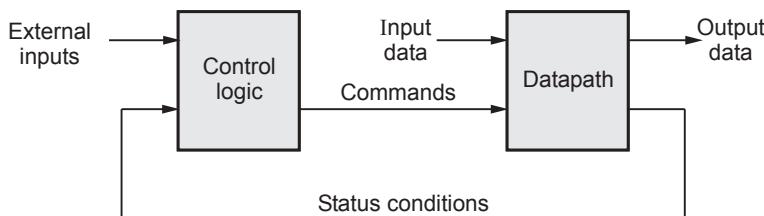


Fig. 8.1.1 Interaction between control logic and datapath

The control sequence and datapath tasks of a digital system are specified by means of a hardware algorithm. A hardware algorithm is a step-by-step procedure to implement the desired task with a selected circuit components. We know that, a flow chart is a convenient way to specify the sequence of procedural steps and decision paths for an algorithm. A special flowchart that has been developed specifically to define digital hardware algorithms is called an **Algorithmic State Machine** (ASM) chart. A state machine is another term used for sequential circuit.

The ASM chart resembles a conventional flow chart, but is interpreted somewhat differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.

8.1.1 ASM Symbols / Notations

An ASM chart consists of three basic elements : The state box, the decision box, and the conditional box.

State box : A state in the control sequence is indicated by a state box, as shown in the Fig. 8.1.2.

As shown in the Fig. 8.1.2, rectangle shape is used to represent state box. (At the left of this box is the name of the state, such as A, B, Q₀, Q₂. On the right hand top corner of the box is a list of the flip-flop values that define that state. The circuit outputs that can occur whenever the circuit is in the corresponding state regardless of input values

are listed within the box.) Finally, a line drawn from state box, known as exit, indicates the path to the next state.

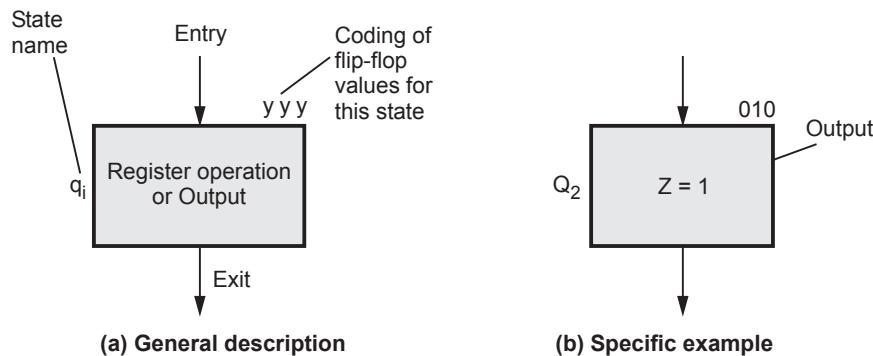


Fig. 8.1.2 State box

Decision box : The decision box describes the effect of an input on the control subsystem. It has a diamond shape box with two or more exit paths, as shown in the Fig. 8.1.3. The input condition to be tested is written inside the diamond box. Each value either 0 or 1 that may be assumed by an input or expression in a diamond is associated with an exit path from that diamond. These paths lead to the blocks corresponding to the next states of the circuit following the next clock pulse.

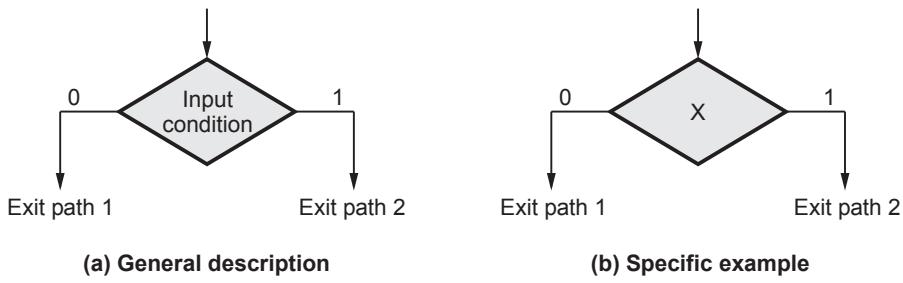


Fig. 8.1.3 Decision box

Conditional box : The state and decision boxes are familiar from use in conventional flowcharts. The third basic element, the conditional box, is unique to the ASM chart. It is an oval shape box. Its rounded corners differentiate it from the state box. The input path for the conditional box always comes from one of the exit paths of a decision box. The outputs that occur when the path to a conditional output box is satisfied are listed within the box.

The outputs that are not listed in either the state box or a conditional box in a particular ASM chart are always inactive when the circuit is in that state.

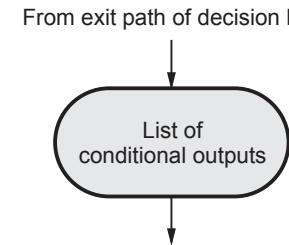


Fig. 8.1.4 Conditional box

Example 8.1.1 Draw the ASM chart for the following state diagram.

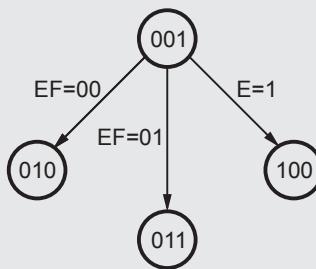


Fig. 8.1.5

Solution : ASM block : As shown in the Fig. 8.1.5 (a), an ASM block consists of one state box and all the conditional and decision boxes necessary to determine the path for next state box/boxes. It has one entrance and one or many exit paths. It describes the state of the system during one clock pulse interval.

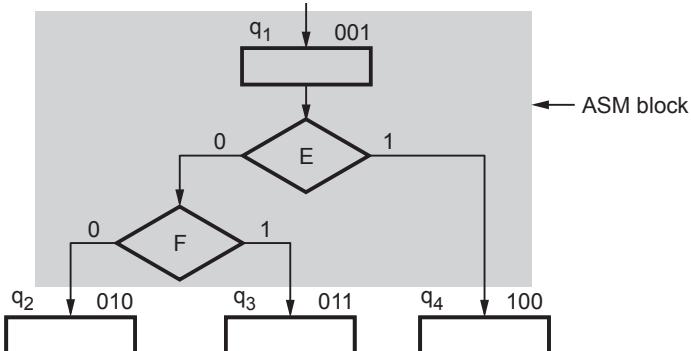


Fig. 8.1.5 (a)

8.1.2 Salient Features of ASM Chart

- An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.
- Every block in an ASM chart specifies the operations that are to be performed during one common clock pulse.
- The operations specified within the state and conditional boxes in the block are performed in the datapath subsystem.
- The change from one state to next is performed in the control subsystem.
- An ASM chart consists of one or more interconnected blocks.
- An ASM block has one entrance and any number of exit paths represented by the structure of the decision boxes.
- Each block in the ASM chart describes the state of the system during one clock-pulse interval. When a digital system enters the state associated with a given ASM block, the outputs indicated within the state box becomes true. The conditions associated with the decision boxes are evaluated to determine which path or paths to be followed to enter into the next ASM block.

A **Link path** is a path through an ASM block from entrance to exit.

Review Questions

1. What is ASM chart ? SPPU : Dec.-08,10,15,16, May-11,13,19, Marks 8
2. How does ASM chart differ from conventional flowchart ? SPPU : May-09,12, Marks 2
3. List the features of ASM chart. SPPU : May-12,15,16, Marks 4
4. State and explain basic components of ASM chart. SPPU : Dec.-11, May-13,15,16,18, Marks 6
5. Mention application of ASM chart ? SPPU : May-13, Dec.-15, Marks 2
6. State and explain basic component of ASM chart. What is difference between ASM chart and conventional flow chart ? SPPU : Dec.-13, Marks 7

8.2 Design of Simple Controller

SPPU : Dec.-08,15, May-09,11,12,13,19

We have seen that, the control subsystem consists of sequential circuit. The process to implement the sequential circuit described by ASM charts consists of three steps :

- Translate the ASM chart to a state table.
 - Convert the state table to a transition table.
 - Develop Boolean expressions for circuit outputs and memory element inputs.
- Let us study these steps with an example.

Example 8.2.1 Develop an ASM chart and state table for a controllable waveform generator that will output any one of the four waveforms given in Fig. 8.2.1, as determined by the values of its two inputs x_1 and x_2 . The period of the first two waveforms is four clock cycles, the period of the third is three, and the period of the fourth waveform is two clock cycles, respectively. When an input change does occur, the new waveform may begin at any point in its period.

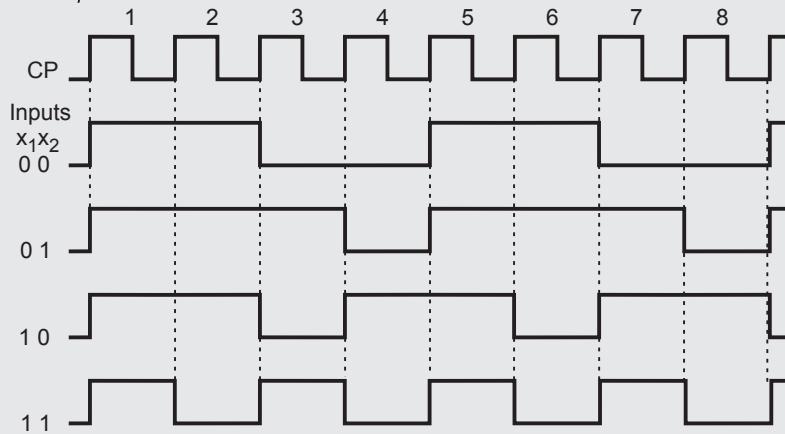


Fig. 8.2.1 Output waveforms

Solution : The ASM chart can be drawn for above waveform with four states, one for each clock cycle of the waveforms with the longest period. For each state the output will be conditional on the values of the input lines in effect at that time. Let us observe the different conditions at different states.

1. State Q_1 : In the first state, Q_1 , i.e. in the first clock cycle, all waveforms are at logic 1. Therefore, the output, $Z = 1$ and is listed in the state box for the first state, Q_1 .

2. State Q_2 : In the second state, Q_2 , i.e. in the second clock cycle, the output is 1 except for the waveform corresponding to inputs $x_1 x_2 = 11$. This condition can be tested by expression $x_1 \wedge x_2$ in the decision box. When the result of expression is 0, the inputs are other than $x_1 x_2 = 11$ and hence output $Z = 1$. This is represented by decision box and conditional box in state Q_2 .

3. State Q_3 : During the third state Q_3 , i.e. in the third clock pulse, the output will be 1, if $x_2 = 1$. The condition of x_2 is checked and accordingly output is made 1 by decision box and conditional box in state Q_3 . Looking at Fig. 8.2.2 we can realise that the output of third waveform in state Q_1 and state Q_4 is same, i.e. logic 1. In other words, in state Q_4 , the third waveform starts new cycle. Therefore, when $x_1 x_2 = 10$, the fourth state is not used and line goes back to first state to start the new cycle.

4. State Q_4 : In the fourth state Q_4 , i.e. in the fourth cycle, the output is always 0. From state Q_4 the circuit returns to Q_1 so that the waveforms may be repeated.

Let us express the ASM chart as a state table. We know that in the state tables we list the outputs along with the next states in the columns corresponding to each combination of input values. Fig. 8.2.3 shows state table for the ASM chart of waveform generator.

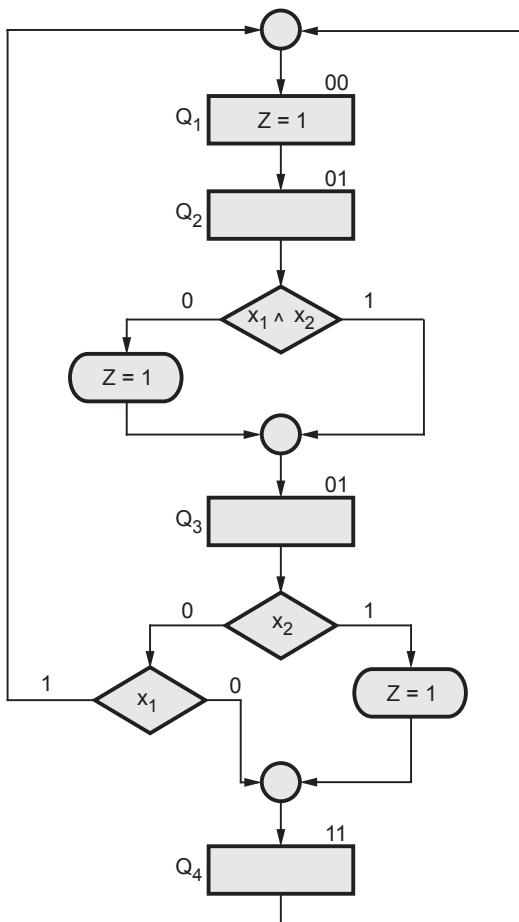


Fig. 8.2.2 ASM chart for waveform generator

Example 8.2.2 Determine the transition table for waveform generator from the state table given in Fig. 8.2.3.

Present state	x_1x_2			
	00	01	10	11
Q_1	$Q_2, 1$	$Q_2, 1$	$Q_2, 1$	$Q_2, 1$
Q_2	$Q_3, 1$	$Q_3, 1$	$Q_3, 1$	$Q_3, 0$
Q_3	$Q_4, 0$	$Q_4, 1$	$Q_1, 0$	$Q_4, 1$
Q_4	$Q_1, 0$	$Q_1, 0$	X	$Q_1, 0$

Fig. 8.2.3 State table

Solution : The state values for $Q_1 = 00$, $Q_2 = 01$, $Q_3 = 10$ and $Q_4 = 11$. Substituting these values in the state table we get transition table as shown in the Fig. 8.2.4.

Once the transition table is determined, the circuit can be realised by determining Boolean expressions for circuit outputs and memory element inputs.

Two methods are usually used for this purpose :

K-map simplification method

Multiplexer method

Present state	x_1x_2			
	00	01	10	11
00	01, 1	01, 1	01, 1	01, 1
01	10, 1	10, 1	10, 1	10, 0
10	11, 0	11, 1	00, 0	11, 1
11	00, 0	00, 0	X	00, 0

Fig. 8.2.4 Transition table

8.2.1 K-map Simplification Method

		Simplification for D_A					
		AB	x_1x_2	00	01	11	10
AB		00		0	0	0	0
01		1	1	1	1		
11		0	0	0	X		
10		1	1	1		0	

$$D_A = \overline{A}\overline{B} + A\overline{B}x_1 + A\overline{B}x_2$$

		Simplification for D_B					
		AB	x_1x_2	00	01	11	10
AB		00		1	1	1	1
01		0		0	0	0	0
11		0		0	0	X	
10		1	1	1		0	

$$D_B = \overline{A}\overline{B} + \overline{B}x_1 + \overline{B}x_2$$

		Simplification for Z					
		AB	x_1x_2	00	01	11	10
AB		00		1	1	1	1
01		1		1		0	1
11		0		0	0	0	X
10		0		1	1	1	0

$$Z = \overline{A}\overline{x}_1 + \overline{A}\overline{x}_2 + \overline{B}x_2$$

Fig. 8.2.5

We are familiar with the K-map simplification process. Let us find the Boolean expression for D_A , D_B (flip-flop inputs) and output Z using the transition table given in Fig. 8.2.5.

Logic diagram

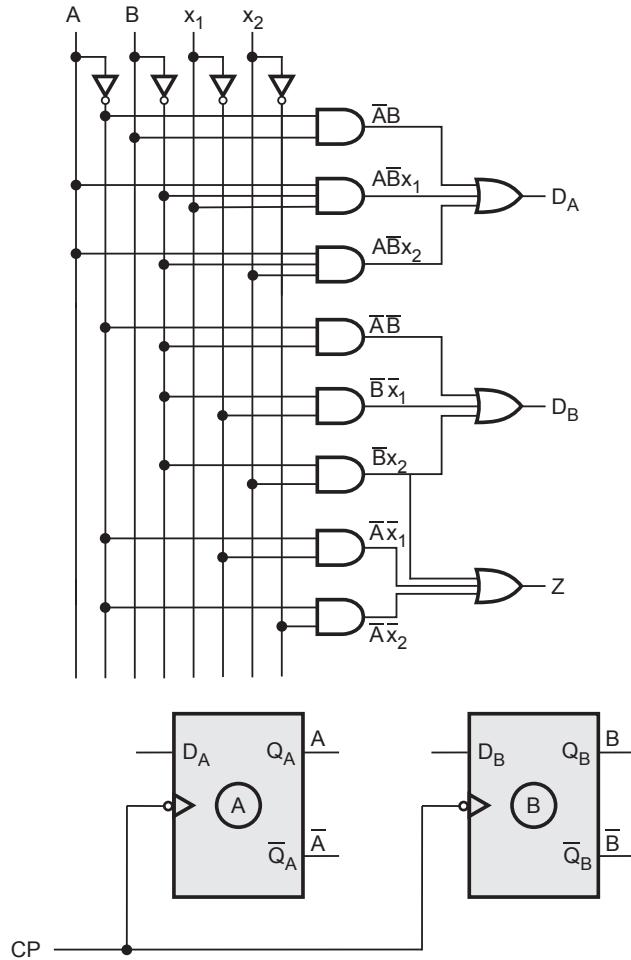


Fig. 8.2.6

8.2.2 Multiplexer Control Method

It is simpler and straight forward method for realisation of combinational circuit for any controller. In this method, the gates and flip-flops are replaced by multiplexers and registers, respectively. In this method there are three levels of components. The first level consists of multiplexers that determine the next state of the register. The second level contain a register that holds the present binary state. The third level has the decoder that provides a separate output for each control state. Sometimes combinational circuit is used in place of decoder.

Consider, for example, the ASM chart of Fig. 8.2.2. It consists of four state and two control inputs x_1 and x_2 . Fig. 8.2.7 shows three level implementation. It consists of two multiplexers, MUX 1 and MUX 2 ; a register with two flip-flops, A and B; and a combinational circuit to determine the output. The outputs of the register are used to select the inputs of the multiplexers. In this way, the present state of the register is used to select one of the inputs from each multiplexer. The outputs of the multiplexers are then applied to the D inputs of A and B. The purpose of each multiplexer is to produce an input to its corresponding flip-flop equal to the binary value of the next state.

The inputs of the multiplexers are determined from the decision boxes and state transitions given in the ASM charts (Refer Fig. 8.2.2). The present states, next states and conditions for transition can be tabulated for ASM chart given in Fig. 8.2.2, as shown in the Table 8.2.1.

No.	Present state		Next state		Condition of transition
1	(Q ₁)	0	0	(Q ₂)	0 1
2	(Q ₂)	0	1	(Q ₃)	1 0
3	(Q ₃)	1	0	(Q ₁)	0 0
				(Q ₄)	1 1
4	(Q ₄)	1	1	(Q ₁)	0 0

Table 8.2.1

Inputs for Multiplexers

MUX 1	MUX 2
0 → 0	0 → 1
1 → 1	1 → 0
2 → $\bar{x}_1 \bar{x}_2 + x_2$	2 → $\bar{x}_1 \bar{x}_2 + x_2$
3 → 0	3 → 0

Table 8.2.2

Note MUX 1 generates input for flip-flop A and MUX 2 generates input for flip-flop B. The multiplexer input can be determined by including condition of transition corresponding to logic 1 bit position in the next state. Consider the transition from Q₁ to Q₂. For flip-flop A the next state is 0, hence the corresponding input of multiplexer 1 is 0. For flip-flop B the next state is 1, hence the corresponding input of multiplexer 1 is the given condition of transition, i.e. 1.

Consider the transition from Q₃ to Q₁ or Q₄. In this case, the next state for flip-flop A is 0 for Q₁ and 1 for Q₄. Therefore, the condition of transition corresponding to Q₄ is taken as corresponding input of multiplexer 1 i.e. $\bar{x}_1 \bar{x}_2 + x_2$.

The equation for output Z can be directly derived from ASM chart. For this we have to observe the ASM chart and find the conditions when output is 1. For example, in state Q₁, Z = 1 therefore Q₁ will appear in the equation for output Z. After collecting all the conditions where Z = 1 we get

$$Z = Q_1 + Q_2 \bar{x}_1 \bar{x}_2 + Q_3 x_2 = \bar{A} \bar{B} + \bar{A} B \bar{x}_1 \bar{x}_2 + A \bar{B} x_2$$

Logic diagram

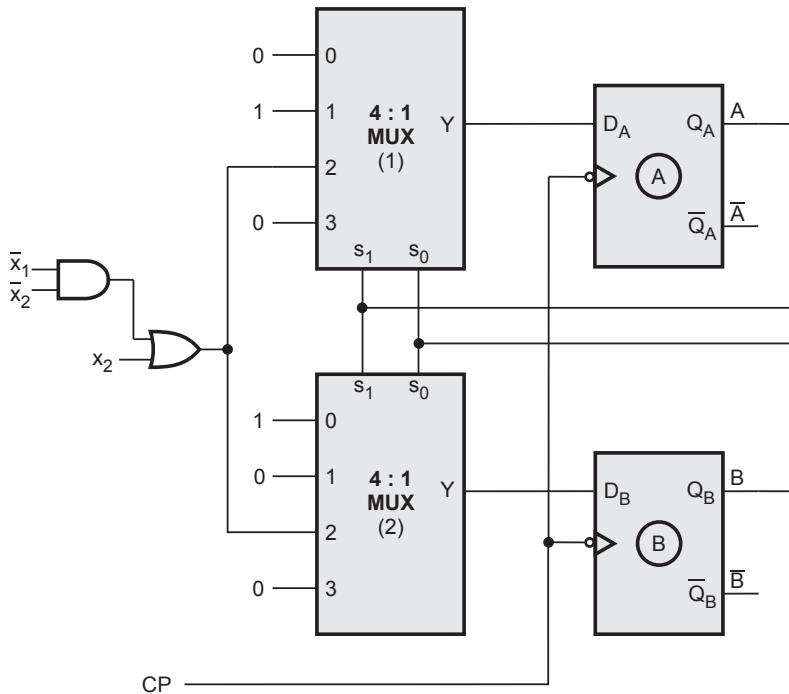


Fig. 8.2.7 (a)

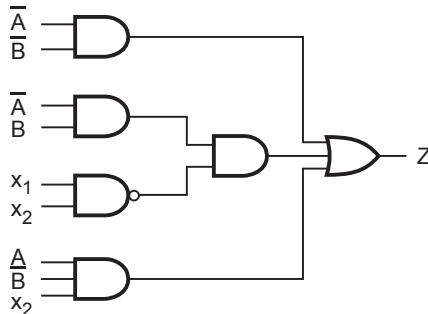


Fig. 8.2.7 (b)

Review Questions

1. Explain in detail, ASM technique of designing the sequential circuit.

SPPU : May-09,12, Marks 6

2. Explain the MUX controller method with the suitable example

SPPU : Dec.-08,15, May-11,19, Marks 8

3. What is ASM chart ? Design ASM chart for 3-bit octal number sequence with up-down conditions.

SPPU : May-11, Marks 8

4. Draw an ASM chart and state table for a sequential ring counter with suitable present state and conditional input.

SPPU : May-11, Marks 8

5. Draw an ASM chart for the 3-bit down counter having one enable line such that :
 $E = 1$ (counting enabled)
 $E = 0$ (counting disabled)
Also draw the state diagram.

SPPU : May-13, Marks 8

8.3 ASM Examples : Sequence Generator, Types of Counters

SPPU : Dec.-98,06,11,12,13,16,17,18,19, May-2000,05,06,07,08,12,14,16,17

Example 8.3.1 Draw an ASM chart for a 2-bit binary counter having enable line E such that : $E = 1$ (counting enabled) $E = 0$ (hold present count).

SPPU : Dec.-98,16,17,18, May-12, Marks 6

Solution :

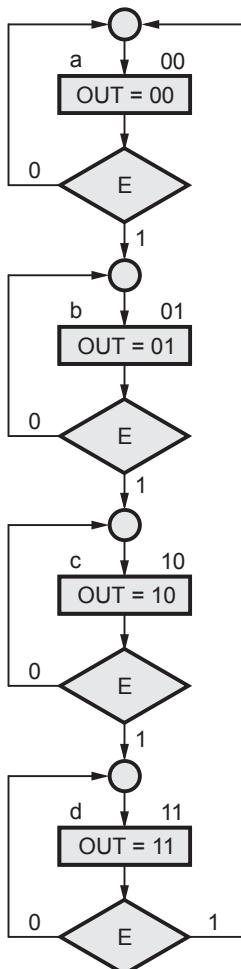


Fig. 8.3.1 ASM Chart

The output represents the output of the counter.

Example 8.3.2 Draw an ASM chart for 2-bit up/down counter having a mode control input M. For M = 1 : Count up and M = 0 : Count down.

Solution :

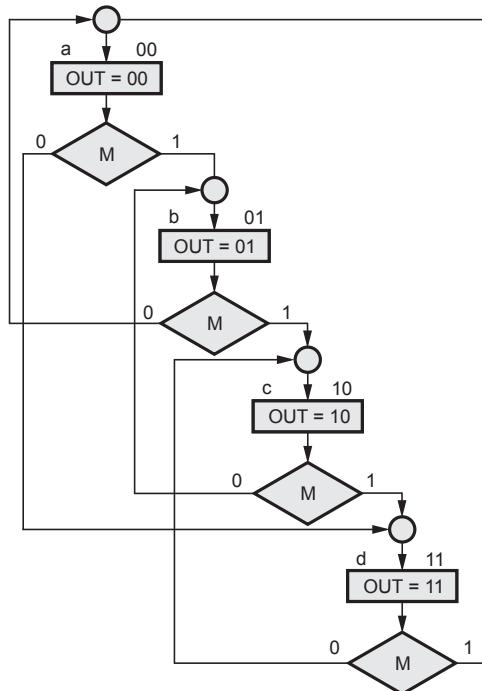


Fig. 8.3.2 ASM Chart

Example 8.3.3 Draw an ASM chart and state table for a 2-bit UP-DOWN counter having mode control input :

M = 1 : Up counting

M = 0 : DOWN counting

The circuit should generate a output 1 whenever count becomes minimum or maximum.

SPPU : May-05,14, Dec.-12, Marks 8

Solution : State diagram

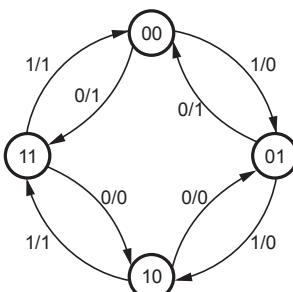
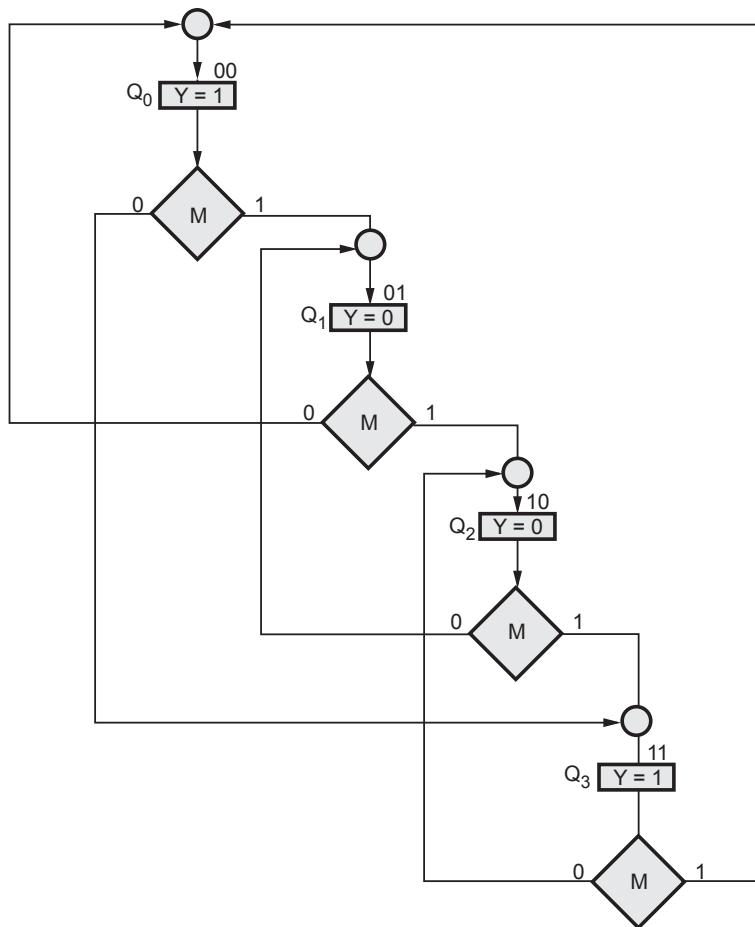


Fig. 8.3.3

ASM chart**Fig. 8.3.4**

Example 8.3.4 Draw an ASM chart for the synchronous circuit having the following description, "The circuit has a control input X, clock and outputs A and B. If $X = 1$, on every clock rising edge the code on BA changes from $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ and repeats. If $X = 0$, the circuit holds the present state.". SPPU : Dec.-11, Marks 10

Solution :

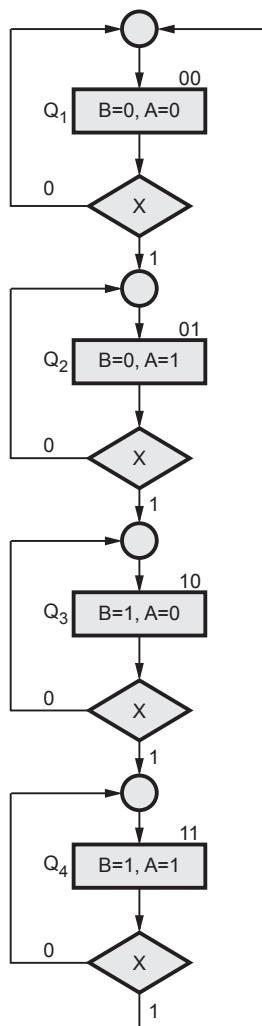


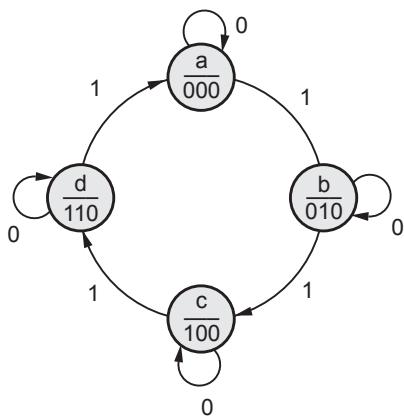
Fig. 8.3.5 ASM chart

Example 8.3.5 Draw an ASM chart and state diagram for the synchronous circuit having the following description :

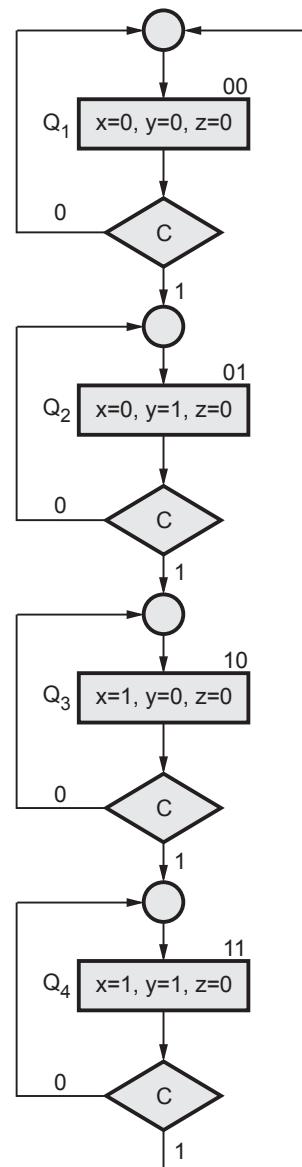
The circuit has control input C, clock and outputs x, y and z.

- If $C = 1$, on every clock rising edge the code on output x, y and z changes from $000 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 000$ and repeats.
- If $C = 0$, the circuit holds the present state.

Solution :



(a) State diagram



(b) ASM chart

Fig. 8.3.6

Example 8.3.6 Draw ASM chart for the following state machine :

A two-bit up counter with output 'Q1Q0' and enable signal 'X' is to be design. If 'X' = 0, Counter changes the state as '00-01-10-11-00'. If 'X' = 1, Counter should remain in present state.

Design your circuit using JK-FF and suitable MUXs.

SPPU : May-06,07,14,17, Marks 7

Solution : Refer example 8.3.4 for ASM chart.

The Table 8.3.1 shows the multiplexer input conditions for given example.

Present state		Next state		Input condition	Multiplexer input	
A	B	A^+	B^+		MUX 1	MUX 2
0	0	0	0	\bar{X}	0	X
0	0	0	1	X		
0	1	0	1	\bar{X}	X	\bar{X}
0	1	1	0	X		
1	0	1	0	\bar{X}	1	X
1	0	1	1	X		
1	1	1	1	\bar{X}	\bar{X}	\bar{X}
1	1	0	0	X		

Table 8.3.1

Logic Diagram

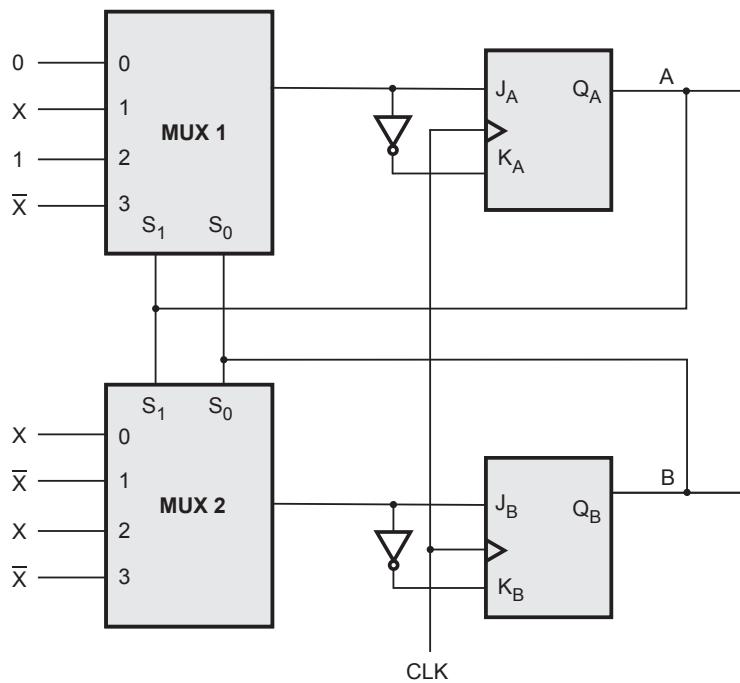


Fig. 8.3.7

Example 8.3.7 A sequential circuit has to count DOWN from '111' to '100'. The circuit also has an input 'X'. If $X = 0$ then the circuit will count DOWN and if $X = 1$ then they will remain in the current state. Draw an ASM chart and state table for this circuit and design the circuit to generate the output using MUX controller method.

SPPU : Dec.-06, Marks 16

Solution :

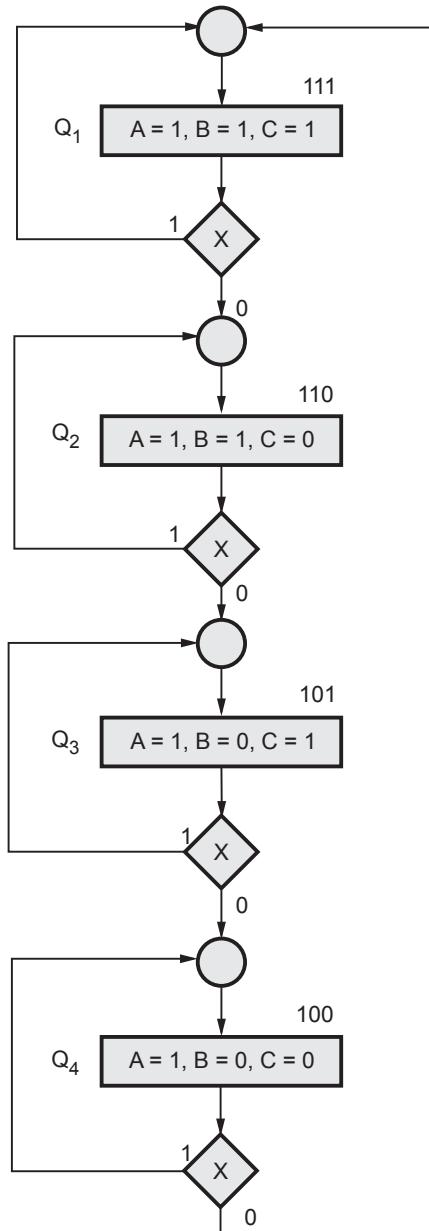


Fig. 8.3.8 ASM chart

The Table 8.3.2 shows the multiplexer input conditions for given example.

Present state			Next state			Input condition	Multiplexer input		
A	B	C	A^+	B^+	C^+		MUX 1	MUX 2	MUX
1	1	1	1	1	1	X			
1	1	1	1	1	0	\bar{X}	1	1	X
1	1	0	1	1	0	X			
1	1	0	1	0	1	\bar{X}	1	X	\bar{X}
1	0	1	1	0	1	X			
1	0	1	1	0	0	\bar{X}	1	0	X
1	0	0	1	0	0	X			
1	0	0	0	0	0	\bar{X}	X	0	0

Table 8.3.2

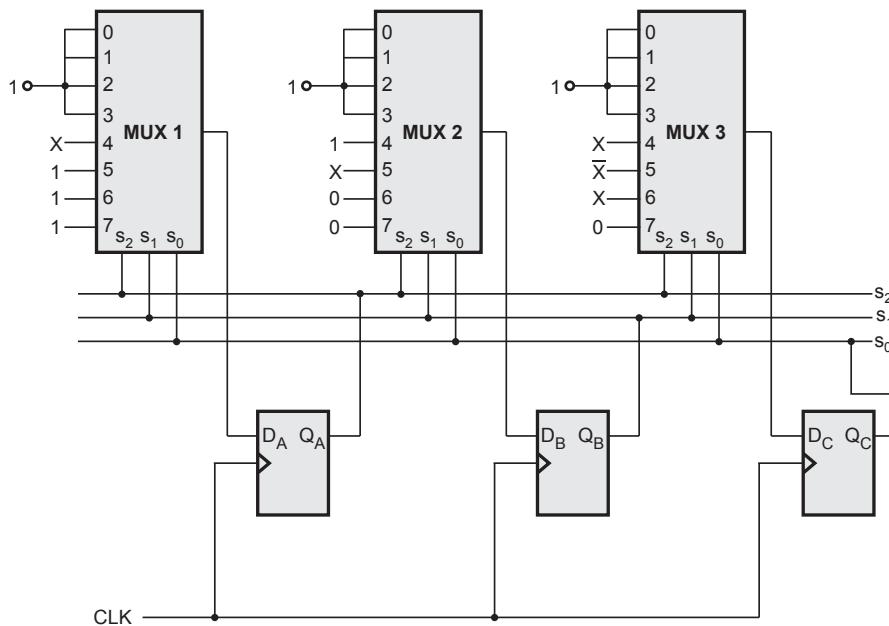


Fig. 8.3.9

It is important to note that unused inputs of multiplexer are connected to logic 1. Therefore, if circuit resets in any other state, the next state will be 111. The circuit will then start down counting if $X = 0$.

Example 8.3.8 A sequential circuit has to COUNT UP from 0 to 3. The circuit also has a control input, E. If $E = 0$, the circuit will remain in the current state and if $E = 1$, the circuit will go to the next state. Draw the necessary ASM chart and state diagram. Also design the circuit with the help of D flip-flops.

SPPU : May-08, Marks 16

Solution : ASM chart :

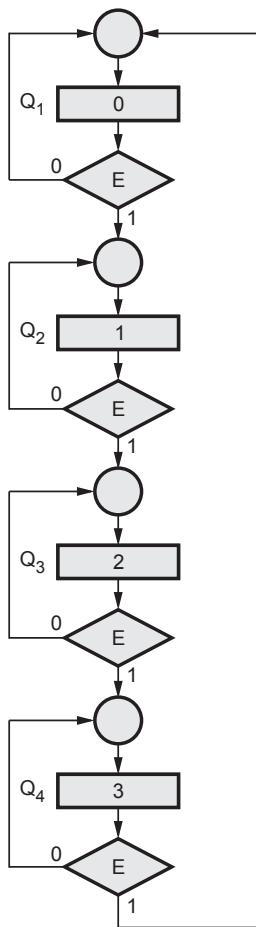


Fig. 8.3.10

State Table :

Input (E)	Present state		Next state		Flip-flop inputs	
	B	A	B	A	D _B	D _A
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0

State diagram :

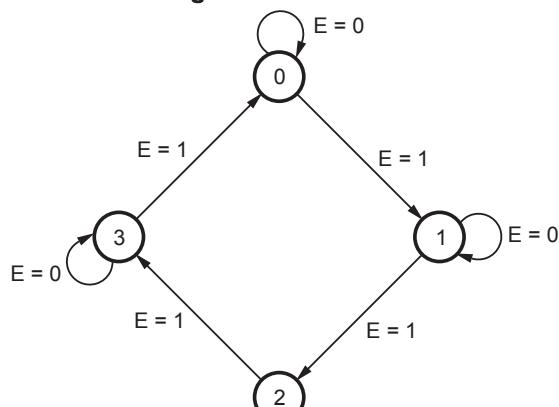
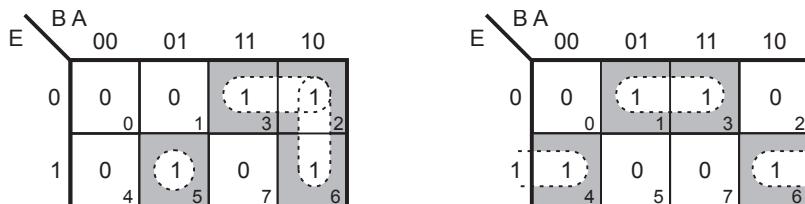


Fig. 8.3.11

K-map simplification :

$$D_B = \bar{E}B + \bar{B}\bar{A} + E\bar{B}A$$

$$D_A = \bar{E}A + E\bar{A}$$

$$E \oplus A$$

Fig. 8.3.12

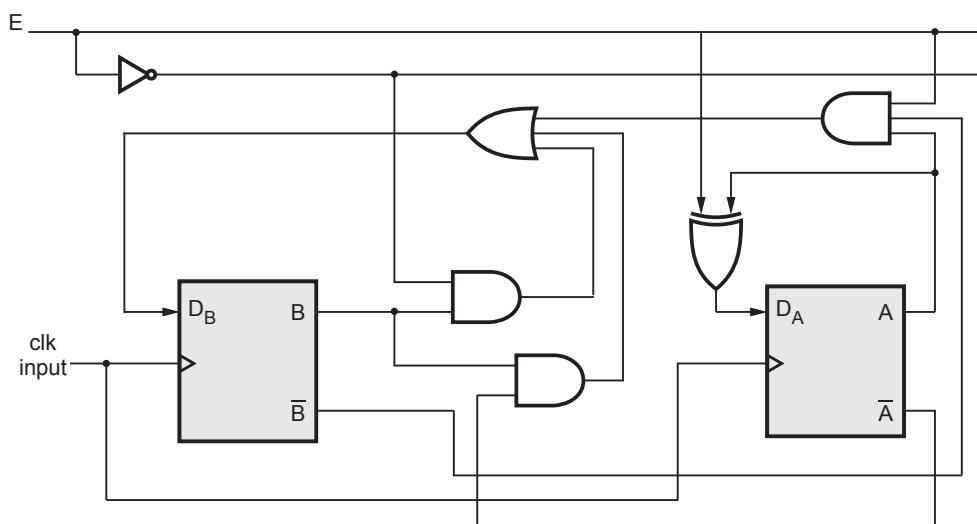
Circuit diagram :

Fig. 8.3.13

Example 8.3.9 Design a sequence generator circuit to generate the sequence 1-3-5-7 using multiplexer controller based ASM approach.

Consideration :

- i) If control input $C = 0$, the sequence generator circuit in the same state.
- ii) If control input $C = 1$, the sequence generator circuit goes into next state.

SPPU : Dec.-13, Marks 7

Solution :

Present state			Next state			Input condition	Multiplexer input		
A	B	C	A^+	B^+	C^+		MUX 1	MUX 2	MUX 3
0	0	1	0	0	1	\bar{X}	0	X	1
0	0	1	0	1	1	X			

0	1	1	0	1	1	\bar{X}	X	\bar{X}	1
0	1	1	1	0	1	X			
1	0	1	1	0	1	\bar{X}	1	X	1
1	0	1	1	1	1	X			
1	1	1	1	1	1	\bar{X}	\bar{X}	\bar{X}	1
1	1	1	0	0	1	X			

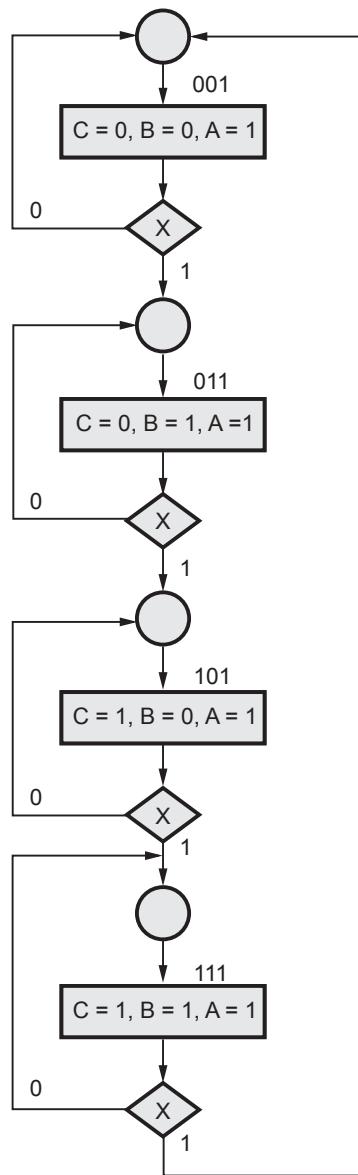


Fig. 8.3.14

Logic diagram

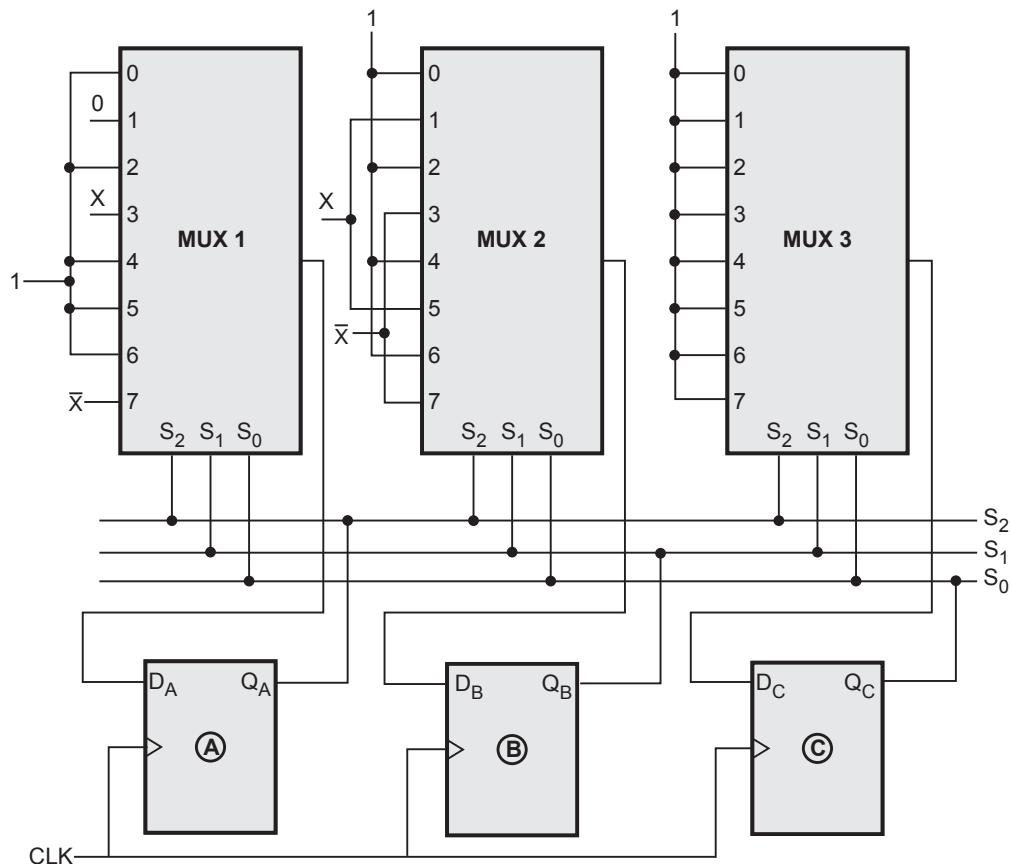


Fig. 8.3.15

Example 8.3.10 Draw an ASM chart for the 3-bit down counter having one enable line such that :

$E = 1$ (counting enabled)

$E = 0$ (counting disabled) Also draw the state diagram

SPPU : Dec.-19, Marks 6

Solution :

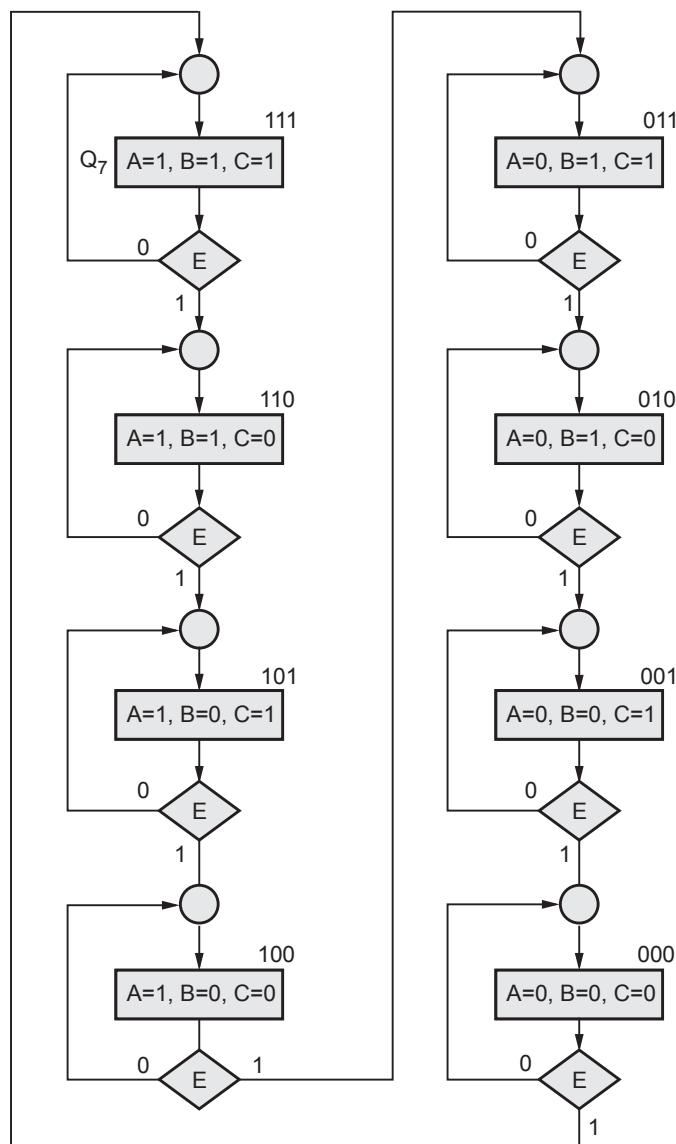


Fig. 8.3.16

State diagram

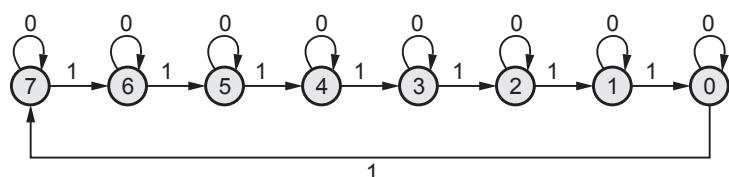


Fig. 8.3.17

Review Questions

1. Draw the ASM chart for the following state machine. A 2-bit up counter is to be designed with output Q, Q_0 and enable signal 'X'. If $X = 0$, then counter changes the state as 00 - 01 - 10 - 11 - 00. If 'X' = 1 then counter remains in same state.

Design the circuit using JK-FF and suitable MUX.

SPPU : May-14, Marks 7

2. Draw an ASM chart and state table for 2-bit up-down counter having mode control input M

When $M = 1$: UP counting

When $M = 0$: Down counting

The circuit should generate output whenever counter becomes minimum or maximum.

SPPU : May-14, Marks 7

3. Draw ASM chart for the following state machine :

A two bit up counter with output 'BA' and enable signal 'X' is to be designed. If 'X' = 0, counter changes the state as '00-01-11-00'. If 'X'=1, counter should remain in present state. Design the circuit using multiplexer controller method.

SPPU : May-16, Marks 8



UNIT IV

9

Programmable Logic Devices

Syllabus

PLD, ROM as PLD, Programmable Logic Array (PLA), Programmable Array Logic (PAL), Designing combinational circuits using PLDs.

Contents

9.1	<i>Introduction</i>	May-06,07,08,16,	
		Dec.-06,08,13,18, Marks 10
9.2	<i>PROM (Programmable Read Only Memory)</i>			
9.3	<i>PLA (Programmable Logic Array)</i>	Dec.-05,08,10,13,14,16,19,	
		May-10,13,14,16,17,18,19,	· Marks 8
9.4	<i>PAL (Programmable Array Logic)</i>	May-05,11,14,	
		Dec.-08,12,13,14,18, Marks 8
9.5	<i>Comparison between PROM, PLA and PAL</i>	Dec.-05,06,07,	
		May-06,07,08,15,16, Marks 4

9.1 Introduction

SPPU : May-06,07,08,16, Dec.-06,08,13,18

So far we have discussed various digital ICs for performing basic digital operations and other functions, such as adders, comparators, arithmetic logic unit, multiplexers, demultiplexers, code converters, shift registers, counters etc. These ICs, due to their fix function are known as **fixed function ICs**. These ICs are designed by their manufacturers and produced in large quantities to satisfy the needs of a wide variety of applications.

We have seen the design of digital circuits using fixed function ICs. There are two more approaches for the design of digital circuits.

- Use of Application Specific Integrated Circuits (ASICs)
- Use of Programmable Logic Devices (PLDs)

In the fixed function IC approach, we have to use various fixed function ICs to implement different functional blocks in the digital circuit. On the other hand, in ASIC, a single IC is designed and manufactured to implement the entire circuit. In the third approach programmable logic devices are used to implement logic functions. The main advantage of PLD approach is that PLDs can be easily configurable by the individual user for specific applications. The Table 9.1.1 shows the comparison between these three design approaches.

Comparison parameter	Fixed-function IC approach	ASIC approach	PLD approach
Development cost	Low	High	Low
Space required	Large	Minimum	Less
Power required	Large	Less	Less
Reliability	Less compared to other two approaches.	Highest	High
Circuit testing	Easy	Specialized testing methods are required with may increase cost and effort.	Easy
Design flexibility	Less	No	More
Modification in design	Possible with change in circuit and/or with change in components.	No	May be possible without any circuit or component changes but only by reconfiguring the device.
Design security	Lack of security i.e. circuit can easily be copied by others.	High	High
Design time	Less	More	Less

Table 9.1.1 Comparison between design approaches

PLDs can be reprogrammed in few seconds and hence gives more flexibility to experiment with designs. Reprogramming feature of PLDs also make it possible accept changes/modifications in the previously design circuits. These two main advantages and others discussed in Table 9.1.1 make PLDs very popular in digital design.

According to architecture, complexity and flexibility in programming PLDs are classified as

- PROMs : Programmable Read Only Memories
- PLAs : Programmable Logic Arrays
- PAL : Programmable Array Logic
- FPGAs : Field Programmable Gate Arrays
- CPLDs : Complex Programmable Logic Devices

Review Questions

1. Define PLD. SPPU : Dec.-18, Mark 1

2. State two advantages of PLD over fixed function IC and application specific IC.

3. What is PLD ? What do you mean by designing using PLDs ? SPPU : Dec.-08, Marks 4

Explain with suitable examples. What are the advantages of designing using PLDs over designing discrete LSI components ? Explain. SPPU : May-06, Dec.-06, Marks 10

4. What is ASIC ? Explain its advantages over conventional discrete circuits. SPPU : May-07, Marks 8, May-08, Marks 6

5. What are different types of PLDs ? SPPU : Dec.-13,18, May-16 Marks 3

9.2 PROM (Programmable Read Only Memory)

The Fig. 9.2.1 shows the block diagram of PROM. It consists of n-input lines and m-output lines. Each bit combination of the input variables is called an **address**. Each bit combination that comes out of the output lines is called a **word**. The number of bits per word is equal to the number of output lines, m. The address specified in binary number denotes one of the minterms of n variables. The number of distinct addresses possible with n-input variables is 2^n . An output word can be selected by a unique address and since there are 2^n distinct addresses in PROM, there are 2^n distinct words in the PROM. The word available on the output lines at any given time depends on the address value applied to the input lines.

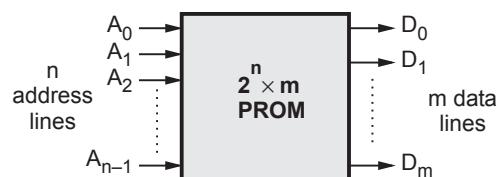


Fig. 9.2.1

Let us consider 64×4 PROM. The PROM consists of 64 words of 4-bits each. This means that there are four output lines and particular word from 64 words presently available on the output lines is determined from the six input lines. There are only six inputs in a 64×4 PROM because $2^6 = 64$ and with six variables, we can specify 64 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 000000, word number 0 is selected and applied to the output lines. If the input address is 111111, word number 63 is selected and applied to the output lines.

The Fig. 9.2.2 shows the internal logic construction of a 64×4 PROM. The six input variables are decoded in 64 lines by means of 64 AND gates and 6 inverters. Each output of the decoder represents one of the minterms of a function of six variables. The 64 outputs of the decoder are connected through fuses to each OR gate. Only four of these fuses are shown in the diagram, but actually each OR gate has 64 inputs and each input goes through a fuse that can be blown as desired.

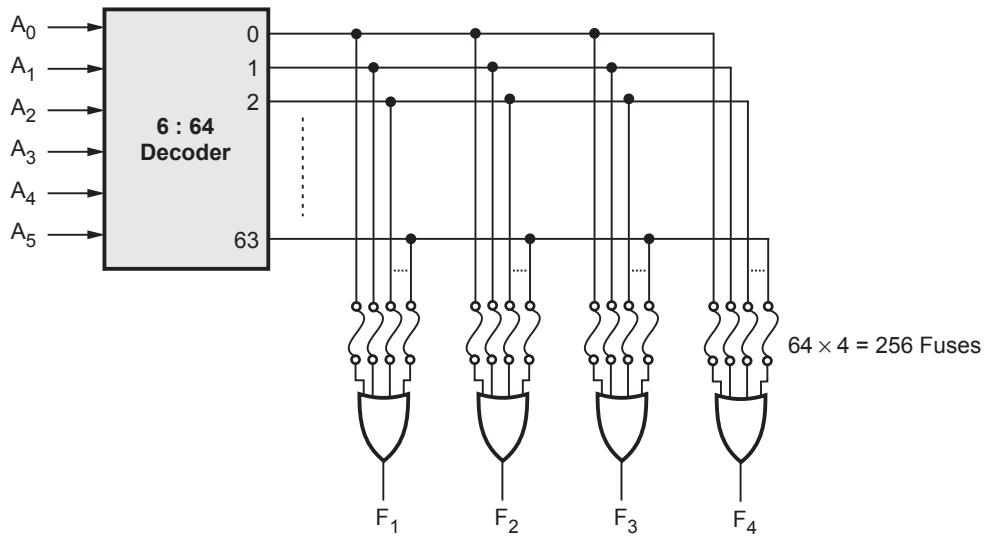
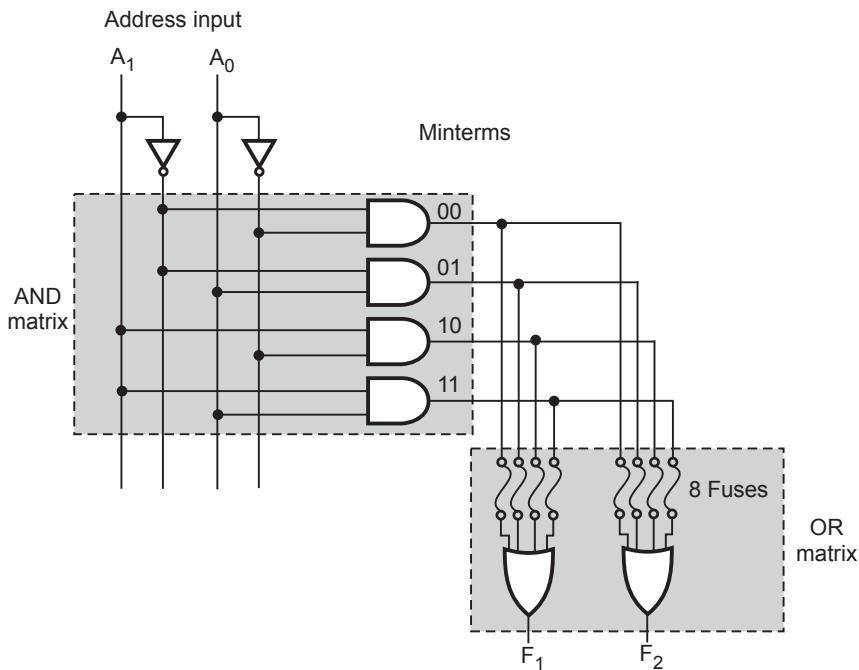
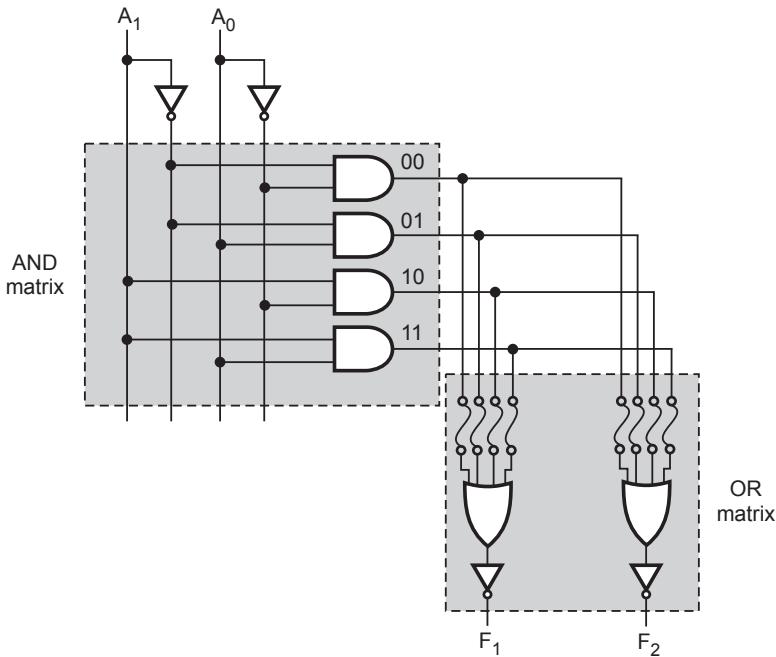


Fig. 9.2.2 Logic construction of 64×4 PROM

The PROM is a two level implementation in sum of minterms form. Let us see AND-OR and AND-OR-INVERTER implementation of PROM. Fig. 9.2.3 shows the 4×2 PROM with AND-OR and AND-OR-INVERTER implementations. (Refer Fig. 9.2.3 on next page)

9.2.1 AND Matrix

The Fig. 9.2.4 shows the AND matrix. It is used to form product terms. It has m AND gates with $2n$ -inputs and m -outputs, one for each AND gate. The Fig. 9.2.4 shows the AND gates formed by diodes and resistors structure. Each AND gate has all the

Fig. 9.2.3 (a) 4×2 PROM with AND-OR gatesFig. 9.2.3 (b) 4×2 PROM with AND-OR-INVERTER gates

input variables in complemented and uncomplemented form. There is a nichrome fuse link in series with each diode which can be burn out to disconnect particular input for

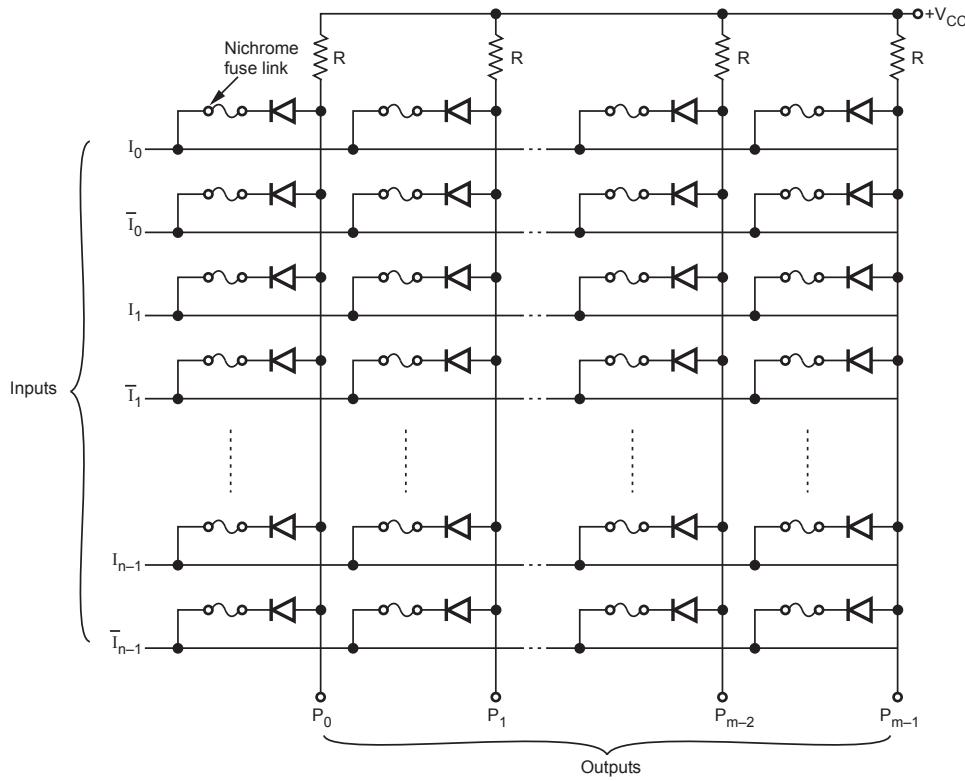


Fig. 9.2.4 Internal structure of AND matrix

that AND gate. Before programming, all fuse links are intact and the product term for each AND gate is given by

$$P = I_0 \cdot \bar{I}_0 \cdot I_1 \cdot \bar{I}_1 \dots \cdot I_{n-1} \cdot \bar{I}_{n-1}$$

The Fig. 9.2.5 shows the simplified and equivalent representation of input connections for one AND gate. The array logic symbol shown in Fig. 9.2.5 (b) uses a single horizontal line connected to the gate input and multiple vertical lines to indicate the individual inputs. Each intersection between horizontal line and vertical line indicates the fuse connection.

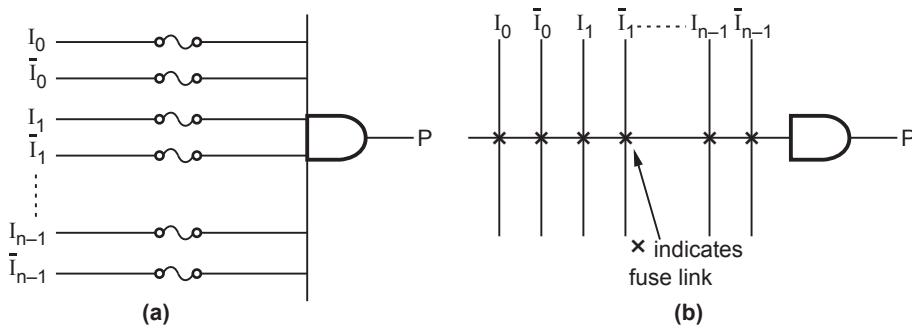


Fig. 9.2.5 Equivalent representation of AND gate

The Fig. 9.2.6 shows the simplified representation of AND matrix with input buffer.

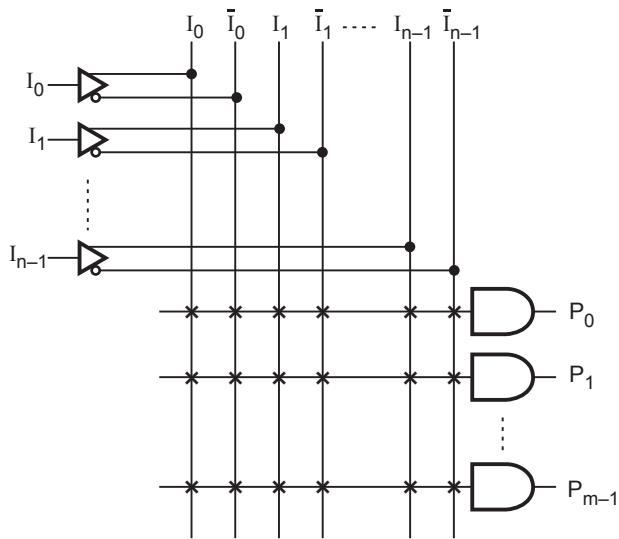


Fig. 9.2.6 Simplified representation of AND matrix with input buffer

9.2.2 OR Matrix

The OR matrix is provided to produce the logical sum of the product term outputs of the AND matrix. The Fig. 9.2.7 shows the OR gates formed by diodes and resistors structure. Each OR gate has all the product terms as input variables. There is a nichrome

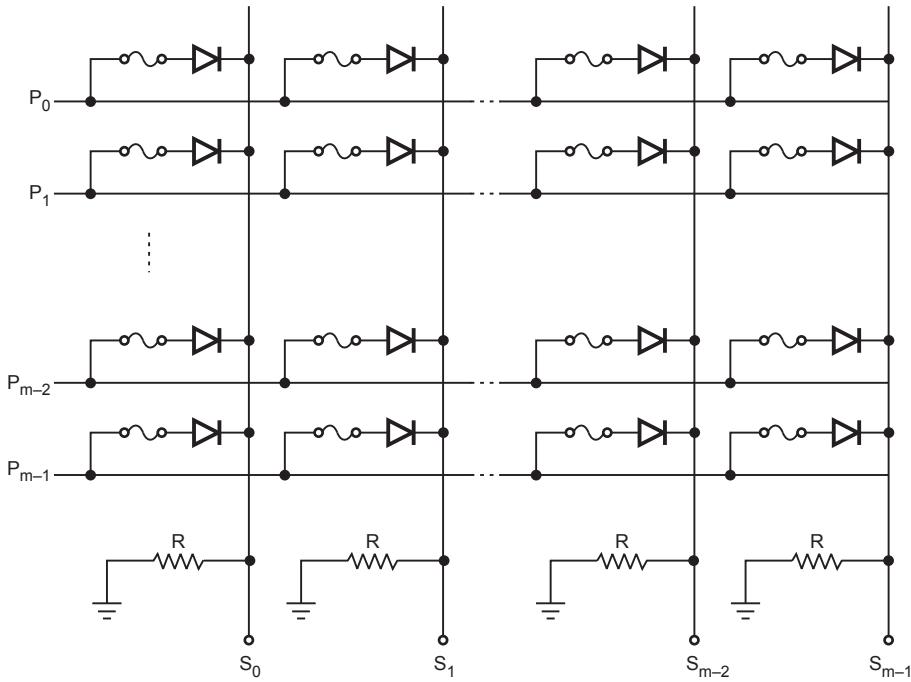


Fig. 9.2.7

fuse link in series with each diode which can be burn out to disconnect particular product term for that OR gate. Before programming, all fuse link in OR matrix are also intact and the sum term for each OR gate is given by,

$$S = P_0 + P_1 + \dots + P_{m-2} + P_{m-1}$$

The Fig. 9.2.8 shows the simplified and equivalent representation of input connections for one OR gate.

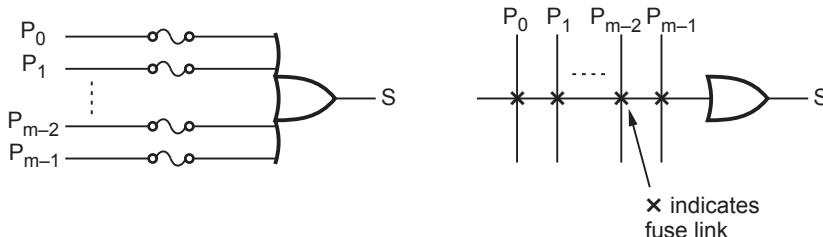


Fig. 9.2.8 Equivalent representation of OR gate

The Fig. 9.2.9 shows the simplified representation of OR matrix.

9.2.3 Invert / Non-invert Matrix

Invert/Non-invert matrix provides output in the complement or uncomplemented form. The user can program the output in either complement or uncomplement form as per design requirements. The typical circuits for invert/non-invert matrix is as shown in Fig. 9.2.10. In both the cases if fuse is intact the output is in its uncomplemented form; otherwise output is in the complemented form.

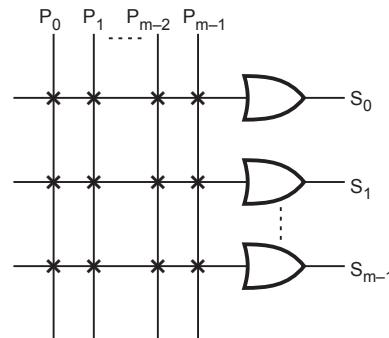


Fig. 9.2.9 Simplified representation of OR matrix

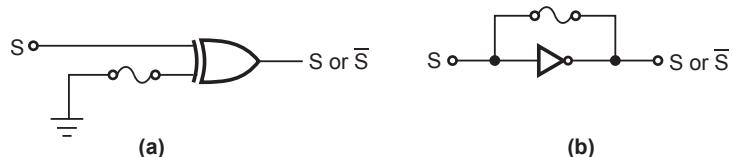


Fig. 9.2.10 Inverting and non-inverting circuits

9.2.4 Combinational Logic Implementation using PROM

Looking at the logic diagram of the PROM, we can realize that each output provides the sum of all the minterms of n-input variables. We know that any Boolean function can be expressed in sum of minterms form. By breaking the links of those minterms not included in the function, each PROM output can be made to represent the Boolean function of one of the output variables in the combinational circuit. For an n-input, m-output combinational circuit, we need a $2^n \times m$ PROM.

Illustrative Examples

Example 9.2.1 Using PROM realize the following expressions.

$$F_1(a, b, c) = \sum m(0, 1, 3, 5, 7)$$

$$F_2(a, b, c) = \sum m(1, 2, 5, 6)$$

Solution :

The given functions have three inputs. They generate $2^3 = 8$ minterms and since there are two functions, there are two outputs. The functions can be realized as shown in Fig. 9.2.11.

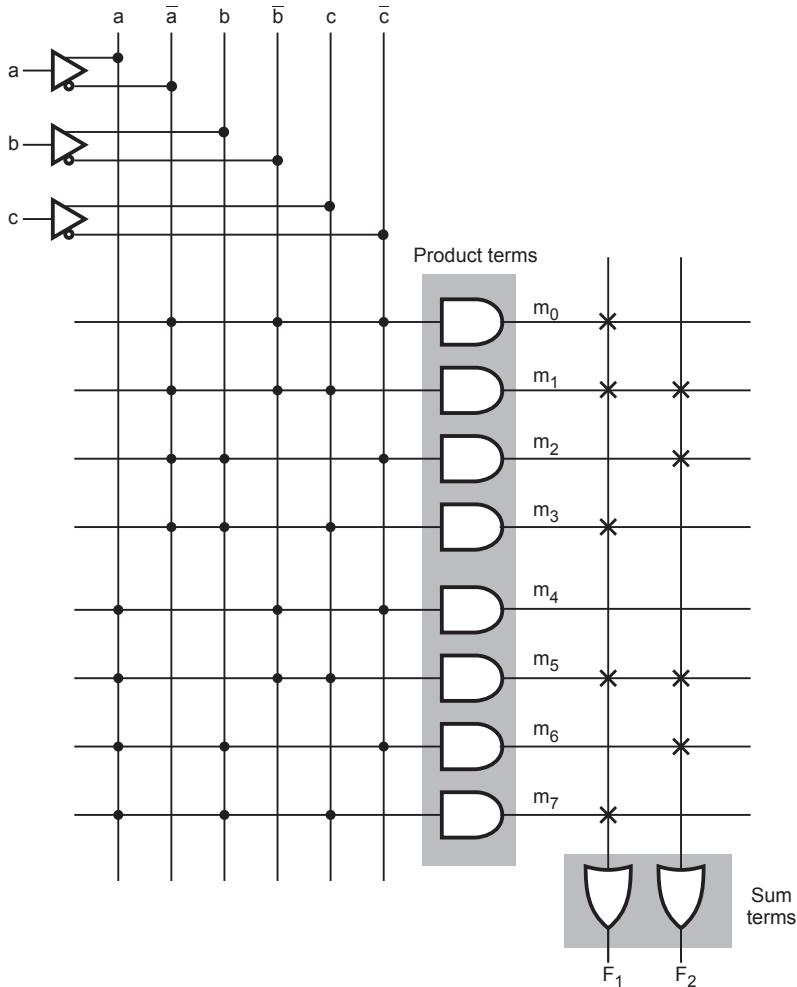
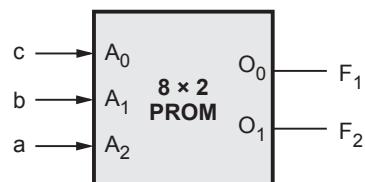


Fig. 9.2.11

The Fig. 9.2.12 shows the block diagram and truth table of PROM.



(a) Block diagram

A₂	A₁	A₀	F₁	F₂
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	1	0

(b) PROM truth table

Fig. 9.2.12

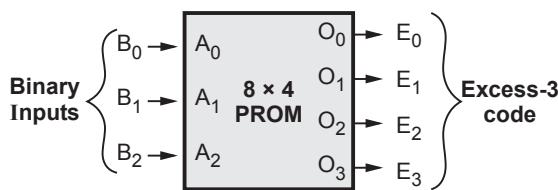
Example 9.2.2 Design a combinational circuit using a PROM. The circuit accepts 3-bit binary number and generates its equivalent Excess-3 code.

Solution : Let us derive the truth table for the given combination circuit. Table 9.2.1 shows the truth table.

In practice when we are designing combinational circuits with PROM, it is not necessary to show the internal gate connections of fuses inside the unit, as shown in the Fig. 9.2.13. (Refer Fig. 9.2.13 on next page). This was shown for demonstration purpose only. The designer has to only specify the PROM (inputs and outputs) and its truth table, as shown in the Fig. 9.2.14.

Inputs			Outputs			
B₂	B₁	B₀	E₃	E₂	E₁	E₀
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Table 9.2.1 Truth table for 3-bit binary to excess-3 converter



(a) Block diagram

A₂	A₁	A₀	E₃	E₂	E₁	E₀
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

(b) PROM truth table

Fig. 9.2.14

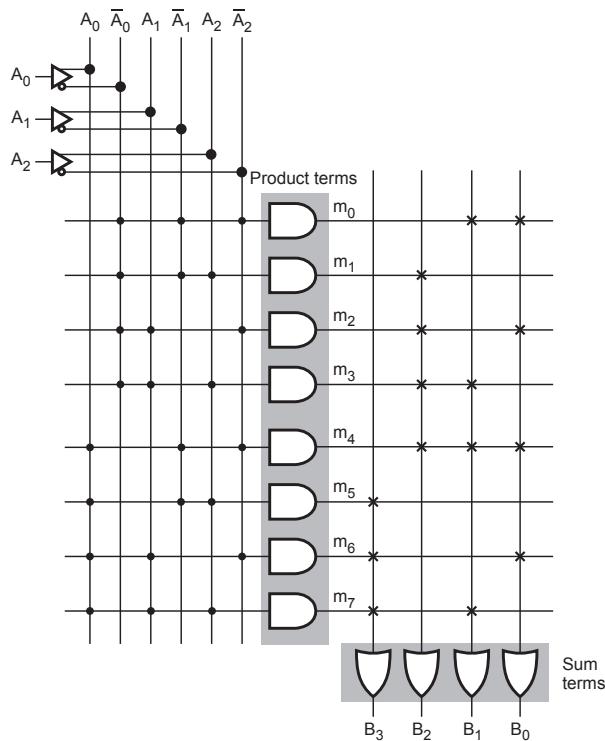
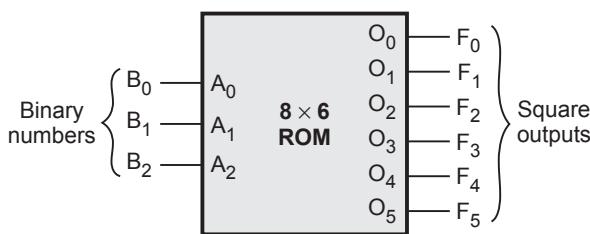


Fig. 9.2.13

Example 9.2.3 Design a combinational circuit using ROM. The circuit accepts 3-bit number and generates an output binary number equal to square of input number.

Solution :



(a) Block diagram

Binary input on address lines			Square of number on data lines								
B_2	B_1	B_0	F_5	F_4	F_3	F_2	F_1	F_0			
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1		
0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	0	0	1		
1	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	1		
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	1		

(b) ROM truth table

Fig. 9.2.15

Example 9.2.4 Implement binary to excess 3 code converter using ROM.

Solution :

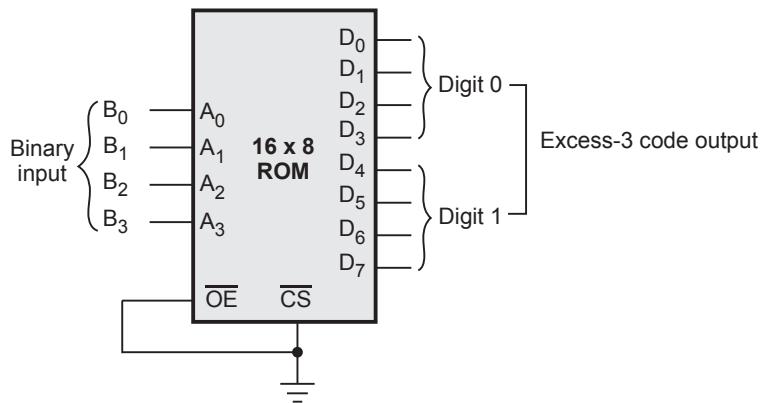


Fig. 9.2.16

Address				Memory contents							
A ₃	A ₂	A ₁	A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	1	0	0	1	1	0	1	0	0
0	0	1	0	0	0	1	1	0	1	0	1
0	0	1	1	0	0	1	1	0	1	1	0
0	1	0	0	0	0	1	1	0	1	1	1
0	1	0	1	0	0	1	1	1	0	0	0
0	1	1	0	0	0	1	1	1	0	0	1
0	1	1	1	0	0	1	1	1	0	1	0
1	0	0	0	0	0	1	1	1	0	1	1
1	0	0	1	0	0	1	1	1	1	0	0
1	0	1	0	0	1	0	0	0	0	1	1
1	0	1	1	0	1	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	1	0	1
1	1	0	1	0	1	0	0	0	1	1	0
1	1	1	0	0	1	0	0	0	1	1	1
1	1	1	1	0	1	0	0	1	0	0	0

Table 9.2.2 ROM contents

Examples for Practice

Example 9.2.5 : Design a switching circuit that converts a 4 bit binary code into a 4 bit Gray code using ROM array.

Example 9.2.6 : Design a 3-bit gray to binary code converter using suitable ROM.

Review Question

1. Write a note on PROM as a PLD.

9.3 PLA (Programmable Logic Array)

SPPU : Dec.-05,08,10,13,14,16,19, May-10,13,14,16,17,18,19,

The combinational circuit do not use all the minterms every time. Occasionally, they have don't care conditions. Don't care condition when implemented with a PROM becomes an address input that will never occur. The result is that not all the bit patterns available in the PROM are used, which may be considered a waste of available equipment.

For cases where the number of don't care conditions is excessive, it is more economical to use a second type of LSI component called a **Programmable Logic Array (PLA)**. A PLA is similar to a PROM in concept; however it does not provide full decoding of the variables and does not generates all the minterms as in the PROM. The PLA replaces decoder by group of AND gates, each of which can be programmed to generate a product term of the input variables. In PLA, both AND and OR gates have fuses at the inputs, therefore in PLA both AND and OR gates are programmable. Fig. 9.3.1 shows the block diagram of PLA. It consists of n-inputs, output buffer with m outputs, m product terms, m sum terms, input and output buffers. The product terms constitute a group of m AND gates and the sum terms constitute a group of m OR gates, called OR matrix. Fuses are inserted between all n-inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the

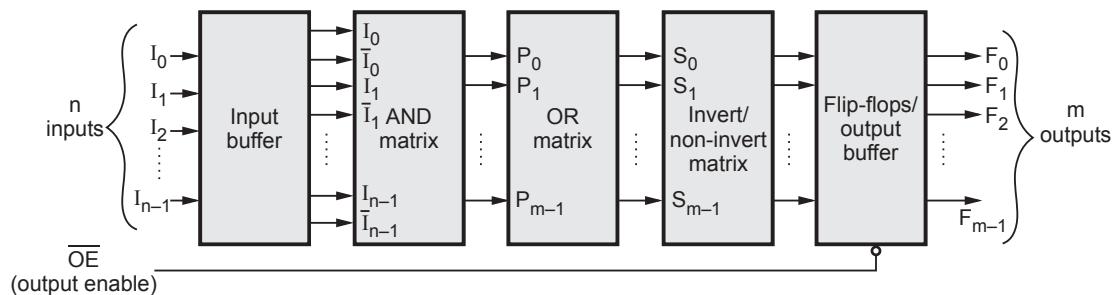


Fig. 9.3.1 Block diagram of a PLA

AND gates and the inputs of the OR gates. The third set of fuses in the output inverters allows the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form. When inverter is bypassed by link we get AND-OR implementation. To get AND-OR-INVERTER implementation inverter link has to be disconnected.

9.3.1 Input Buffer

Input buffers are provided in the PLA to limit loading of the sources that drive the inputs. They also provide inverted and non-inverted form of inputs at its output. The Fig. 9.3.2 shows two ways of representing input buffer for single input.

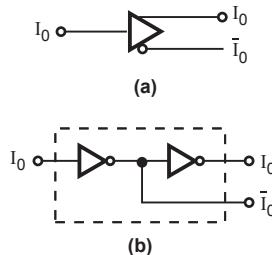


Fig. 9.3.2 Input buffer for single input line

9.3.2 Output Buffer

The driving capacity of PLA is increased by providing buffers at the output. They are usually TTL compatible. The Fig. 9.3.3 shows the tri-state, TTL compatible output buffer. The output buffer may provide totem-pole, open collector or tri-state output.

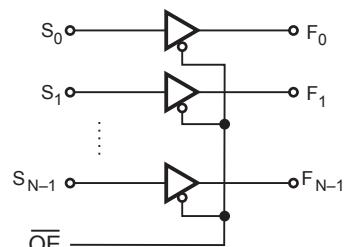


Fig. 9.3.3 Output buffers

9.3.3 Output through Flip-Flops

For the implementation of sequential circuits we need memory elements, flip-flops and combinational circuitry for deriving the flip-flop inputs. To satisfy both the needs some PLAs are provided with flip-flop at each output, as shown in the Fig. 9.3.4.

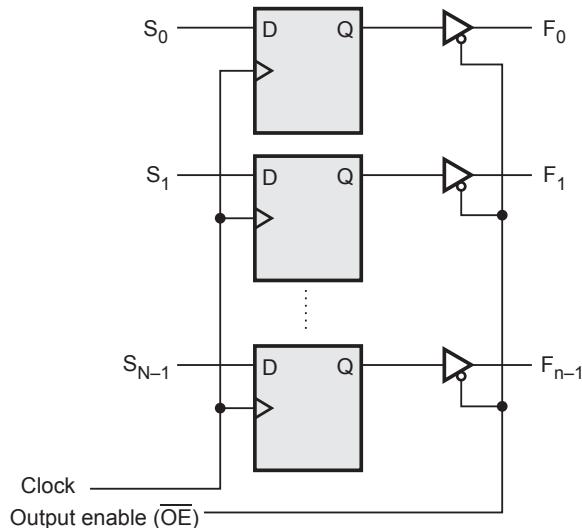


Fig. 9.3.4 PLA with flip-flop at the output

9.3.4 Implementation of Combination Logic Circuit using PLA

Like ROM, PLA can be **mask-programmable** or **field-programmable**. With a mask-programmable PLA, the user must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a user-made PLA that has the required internal paths between inputs and outputs. A second type of PLA available is called a **field-programmable logic array** or FPLA. The FPLA can be programmed by the user by means of certain recommended procedures. FPLAs can be programmed with commercially available programmer units.

As mentioned earlier, user has to submit PLA program table to the manufacturers to get the user-made PLA. Let us study how to determine PLA program table with the help of example.

Illustrative Examples

Example 9.3.1 A combinational circuit is defined by the functions :

$$F_1 = \sum m(3, 5, 7), F_2 = \sum m(4, 5, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs.

SPPU : May-14, Dec.-14, Marks 6

Solution :

Step 1 : Simplify the given Boolean functions

The Boolean functions are simplified, as shown in the Fig. 9.3.5. The simplified functions in sum of products are obtained from the maps are :

$$F_1 = AC + BC, \quad F_2 = A\bar{B} + AC$$

Step 2 : Write PLA program table

Therefore, there are three distinct product terms : AC , BC and $A\bar{B}$, and two sum terms. The PLA program table shown in Table 9.3.1 consists of three columns specifying product terms, inputs and outputs. The first column gives the lists of product terms numerically. The second

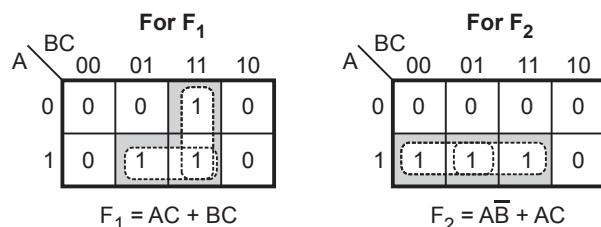


Fig. 9.3.5

Product term	Inputs		Outputs		
	A	B	C	F_1	F_2
AC	1	1	-	1	1
BC	2	-	1	1	-
$A\bar{B}$	3	1	0	-	1
				T	T
					T/C

Table 9.3.1 PLA program table

column specifies the required paths between inputs and AND gates. The third column specifies the required paths between the AND gates and the OR gates. Under each output variable, we write a T (for true) if the output inverter is to be bypassed, and C (for complement) if the function is to be complemented with the output inverter. The product terms listed on the left of first column are not the part of PLA program table they are included for reference only.

Step 3 : Implementation

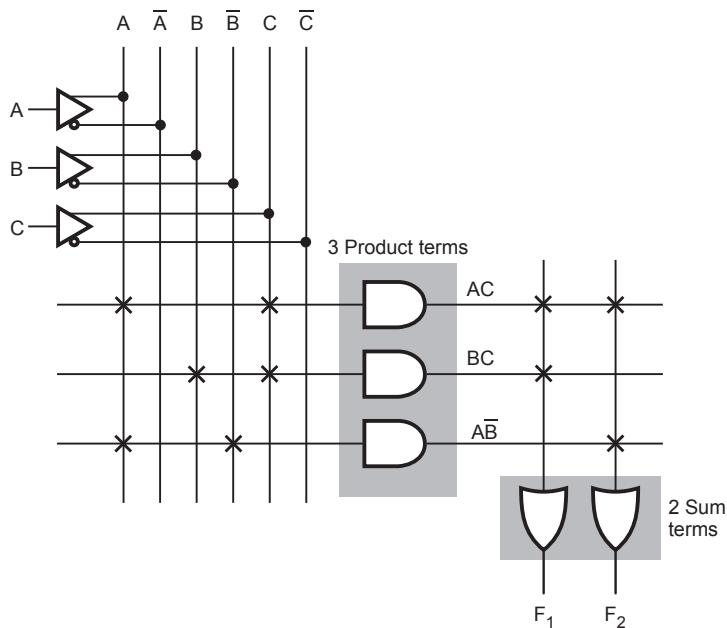


Fig. 9.3.6

Example 9.3.2 Draw a PLA circuit to implement the logic functions $\bar{ABC} + A\bar{B}C + A\bar{C}$ and $\bar{A}\bar{B}\bar{C} + BC$.

Solution :

Step 1 : Simplify the Boolean functions

$$\begin{aligned}
 \bar{A}BC + A\bar{B}C + A\bar{C} &= \bar{A}BC + A(\bar{B}C + \bar{C}) \\
 &= \bar{A}BC + A(\bar{B} + \bar{C}) \quad \because A + \bar{A}B = A + B \\
 &= \bar{A}BC + A\bar{B} + A\bar{C}
 \end{aligned}$$

Note : The second Boolean function is in simplified form.

Step 2 : Implementation (see Fig. 9.3.7 on next page)

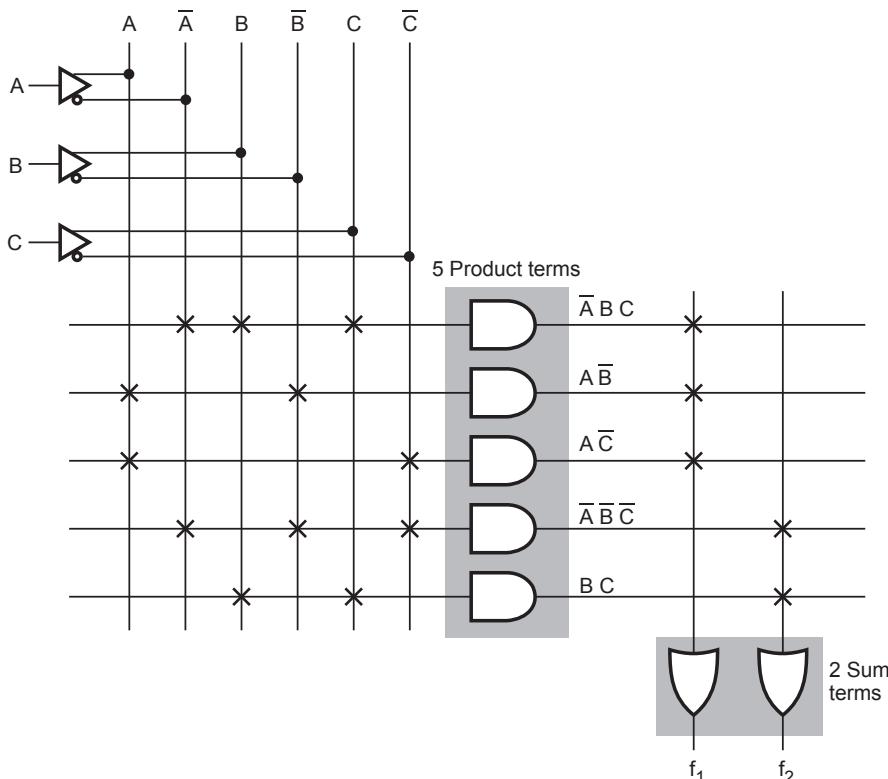


Fig. 9.3.7

Example 9.3.3 Implement the following multiboolian function using $3 \times 4 \times 2$ PLA PLD.

$$f_1(a_2, a_1, a_0) = \sum m(0, 1, 3, 5) \text{ and } f_2(a_2, a_1, a_0) = \sum m(3, 5, 7)$$

Solution : Step 1 : Simplify the Boolean functions.

$$\therefore f_1 = \bar{a}_2 \bar{a}_1 + \bar{a}_2 a_0 + \bar{a}_1 a_0$$

$$f_2 = a_2 a_0 + a_1 a_0$$

To implement functions f_1 and f_2 we require $3 \times 5 \times 2$ PLA and we have to implement them using $3 \times 4 \times 2$ PLA. Therefore, we have to examine product terms by grouping 0s instead of 1. That is product terms for complement of a function.

$$\therefore \bar{f}_1 = a_2 \bar{a}_0 + a_1 \bar{a}_0 + a_2 a_1$$

$$\bar{f}_2 = \bar{a}_2 \bar{a}_1 + a_1 \bar{a}_0 + a_2 \bar{a}_0$$

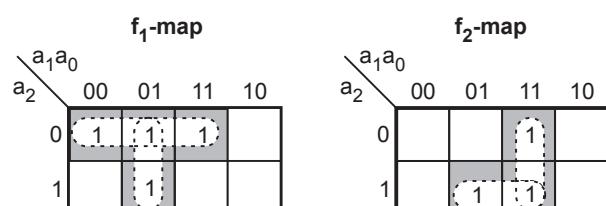


Fig. 9.3.8 K-map simplification

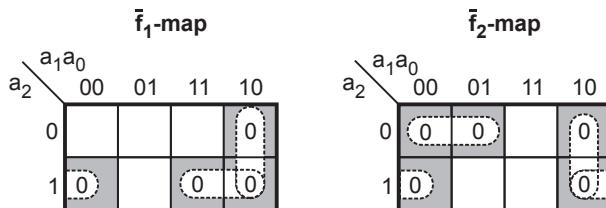


Fig. 9.3.9

Step 2 : Implementation

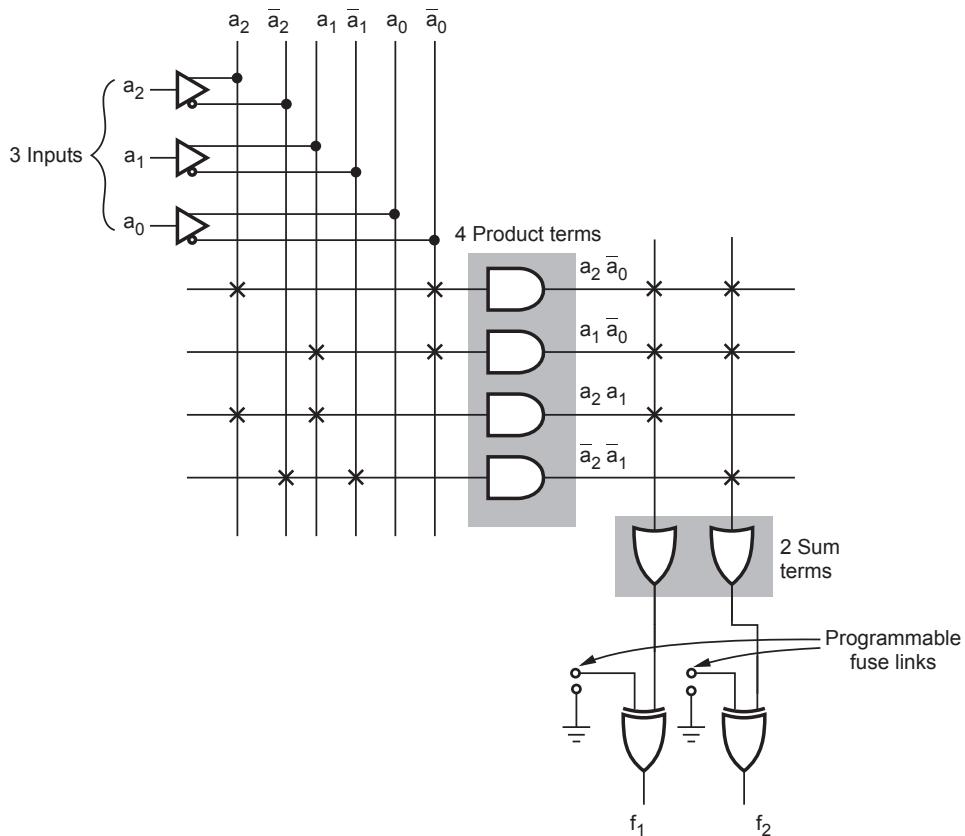


Fig. 9.3.10 Logic diagram

Looking at function outputs we can realize that product terms $a_2 \bar{a}_0$ and $a_1 \bar{a}_0$ are common in both functions. Therefore, we need only 4 product terms and functions can be implemented using a $3 \times 4 \times 2$ PLA as shown in Table 9.3.2 and Fig. 9.3.10.

PLA

Product terms	Inputs			Outputs	
	a_2	a_1	a_0	f_1	f_2
$a_2 \bar{a}_0$	1	—	0	1	1
$a_1 \bar{a}_0$	—	1	0	1	1
$a_2 a_1$	1	1	—	1	—
$\bar{a}_2 \bar{a}_1$	0	0	—	—	1
				C	C

Table 9.3.2

As shown in the Fig. 9.3.10 exclusive-OR gate is programmed to invert the function to get the desired function outputs.

Example 9.3.4 A combinational circuit is defined by the functions,

$$F_1 = \sum m(1, 3, 5) \quad F_2 = \sum m(5, 6, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs.

Solution : K-map simplification

To implement functions F_1 and F_2 we require $3 \times 4 \times 2$ PLA and we have to implement them using $3 \times 3 \times 2$ PLA. There we have to examine product terms by grouping 0s instead of 1. That is product terms for complement of a function.

Looking at function outputs, functions \bar{F}_1 and \bar{F}_2 have one common product term. Thus they have total 3 product terms and can be implemented using $3 \times 3 \times 2$ PLA.

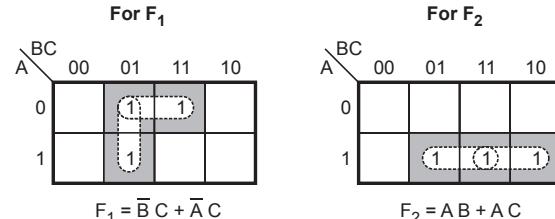


Fig. 9.3.11

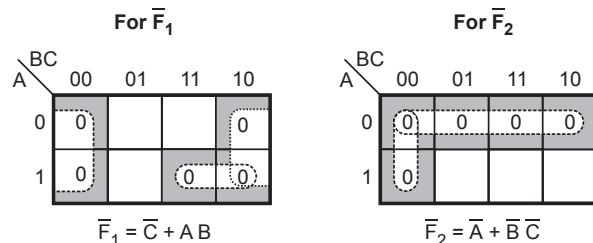


Fig. 9.3.12

PLA program table

Product terms	Inputs			Outputs	
	A	B	C	F_1	F_2
\bar{C}	-	-	0	1	-
AB	1	1	-	1	1
AC	1	-	1	-	1
				C	T

Table 9.3.3

Implementation : Refer Fig. 9.3.13 on next page.

Example 9.3.5 Design a BCD to Excess-3 code converter and implement using suitable PLA.

SPPU : May-18, Marks 6

Solution : Step 1 : Derive the truth table of BCD to Excess-3 converter

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0

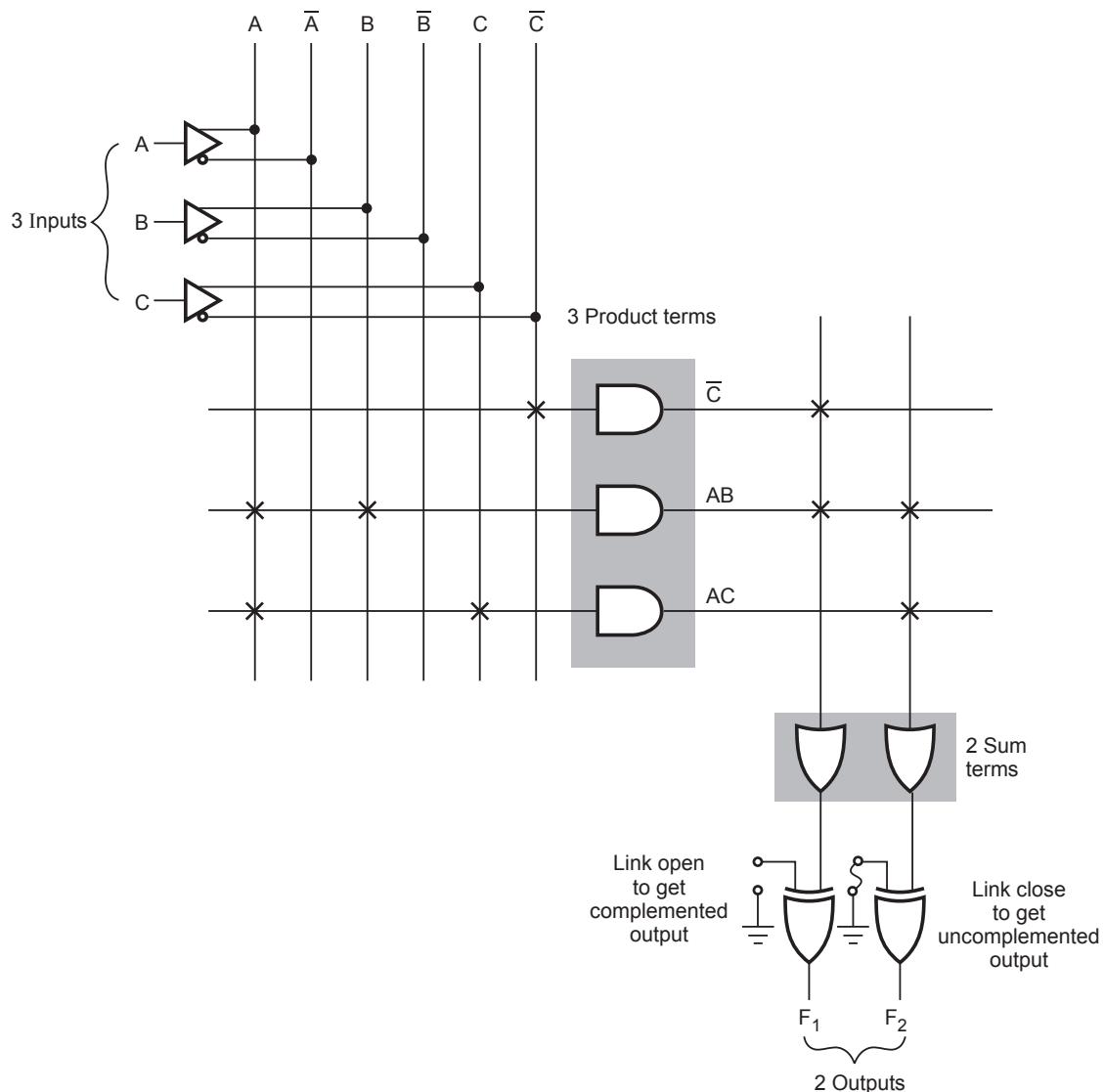


Fig. 9.3.13

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

Table 9.3.4 Truth table for BCD to Excess-3 code converter

Step 2 : Simplify the Boolean functions for Excess-3 code

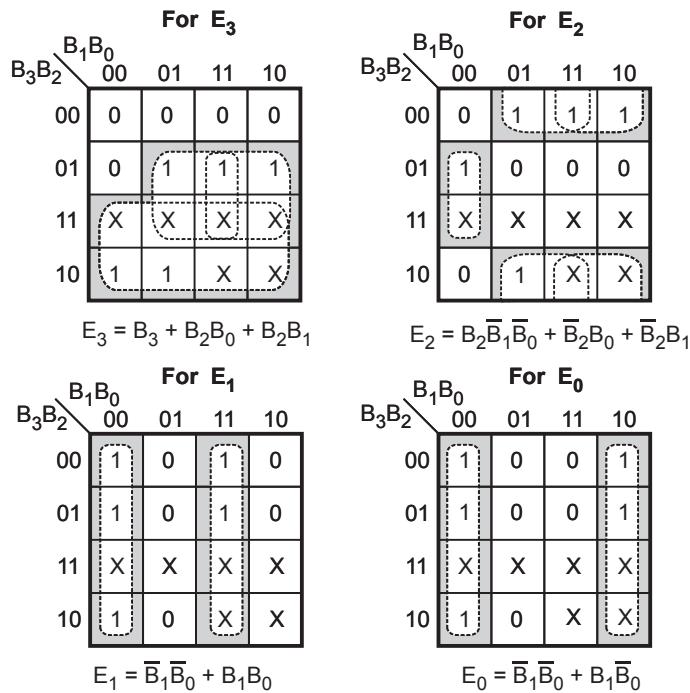
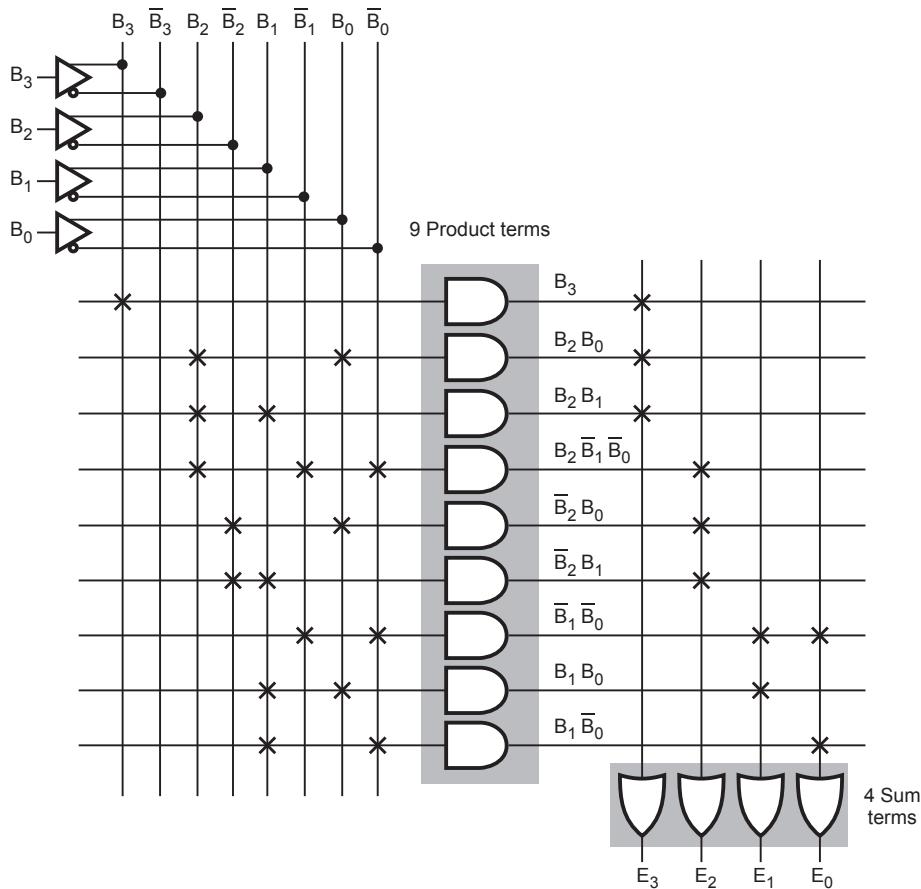


Fig. 9.3.14

Step 3 : Write PLA program table

Product terms	Inputs				Outputs			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
B_3	1	1	—	—	—	1	—	—
B_2B_0	2	—	1	—	1	1	—	—
B_2B_1	3	—	1	1	—	1	—	—
$\bar{B}_2\bar{B}_1\bar{B}_0$	4	—	1	0	0	—	1	—
\bar{B}_2B_0	5	—	0	—	1	—	1	—
\bar{B}_2B_1	6	—	0	1	—	—	1	—
\bar{B}_1B_0	7	—	—	0	0	—	—	1
B_1B_0	8	—	—	1	1	—	—	1
$B_1\bar{B}_0$	9	—	—	1	0	—	—	1
					T	T	T	T/C

Table 9.3.5 PLA program table

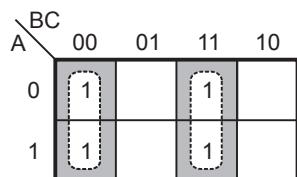
Step 4 : Implementation**Fig. 9.3.15**

Example 9.3.6 Implement the following function using PLA :

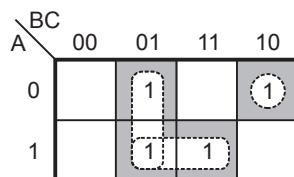
$$f_1 = \sum m(0, 3, 4, 7)$$

$$f_2 = \sum m(1, 2, 5, 7)$$

SPPU : Dec.-08, 14, Marks 8

Solution : K-map simplification

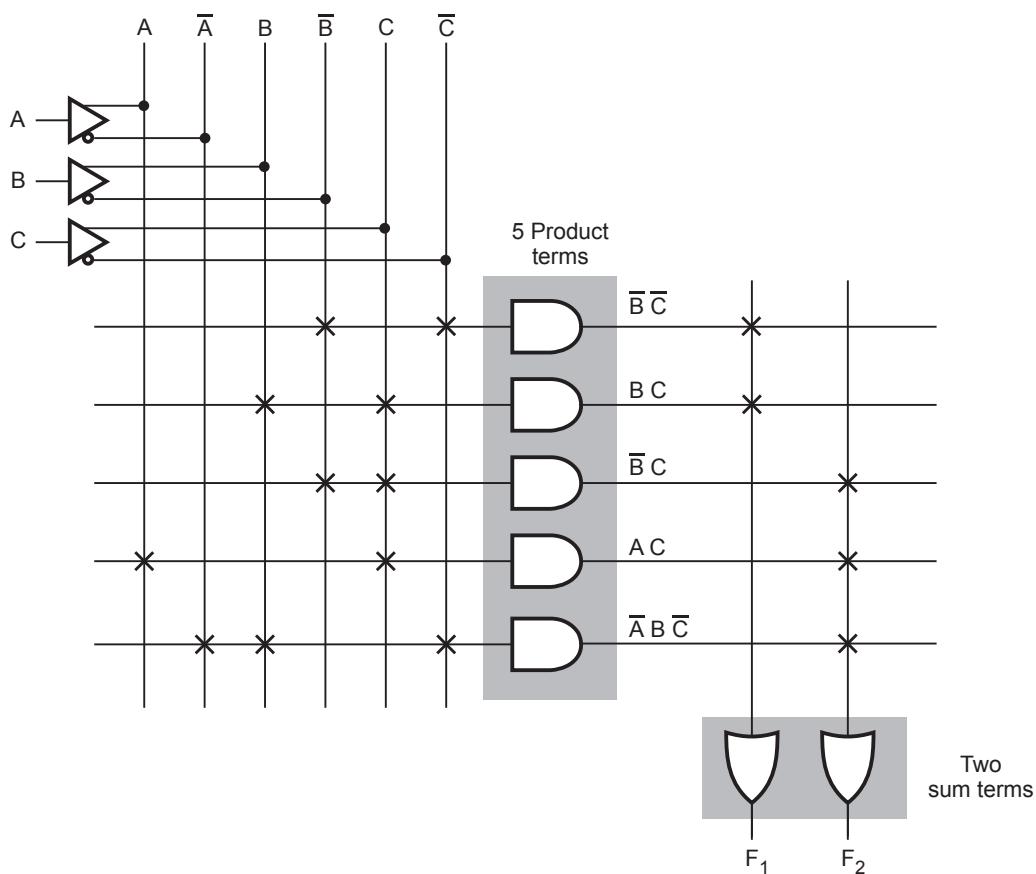
$$f_1 = \overline{B} \overline{C} + B C$$



$$f_2 = \overline{B} C + A C + \overline{A} \overline{B} \overline{C}$$

Fig. 9.3.16

Refere Fig. 9.3.17.

**Fig. 9.3.17**

Example 9.3.7 Design using PLD a 3 : 8 decoder.

SPPU : May-10, Marks 8; Dec.-13, May-16 Marks 4

Solution : The Fig. 9.3.18 shows 3 : 8 decoder using PLD.

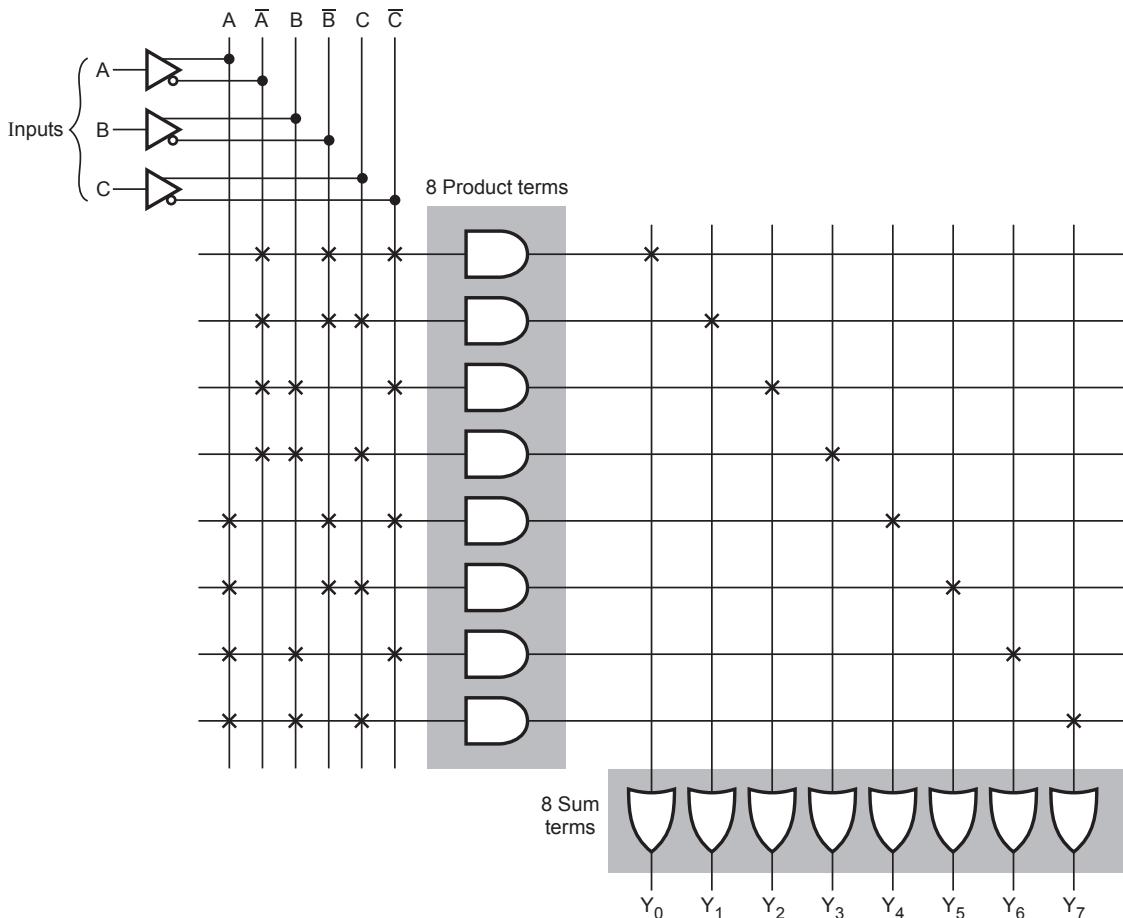


Fig. 9.3.18

Example 9.3.8 A combinational circuit is defined by the function

$$F_1(A, B, C) = \sum m(0, 1, 2, 4)$$

$$F_2(A, B, C) = \sum m(1, 3, 5, 6)$$

Implement this circuit with PLA.

SPPU : Dec.-13, Marks 7

Solution : K-map simplifications

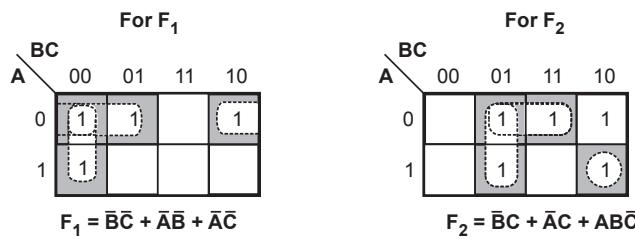


Fig. 9.3.19

Implementation : Refer Fig. 9.3.20.

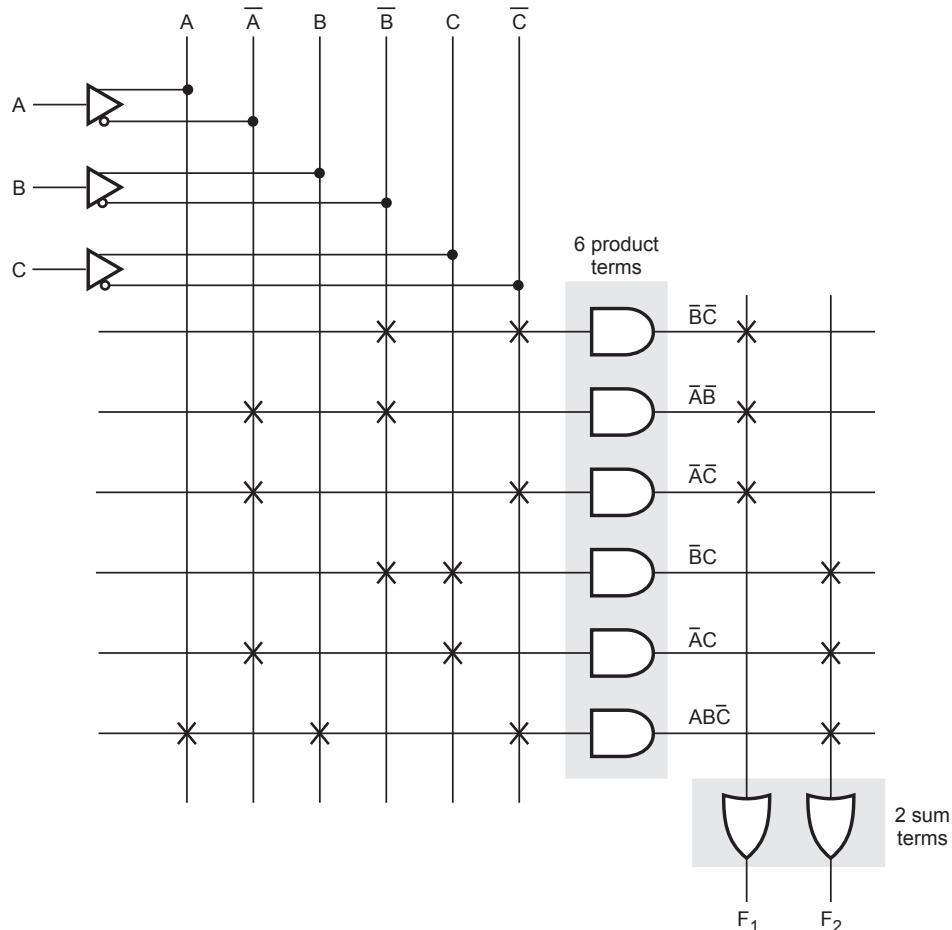


Fig. 9.3.20

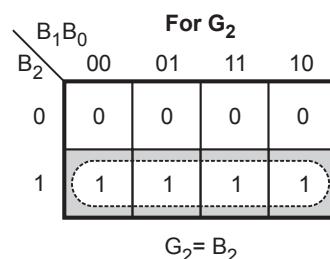
Example 9.3.9 Implement 3 bit binary to gray code converter using PLA.

SPPU : Dec.-16, May-18,19, Marks 6

Solution :

Binary code			Gray code		
B ₂	B ₁	B ₀	G ₂	G ₁	G ₀
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

K-map Simplification



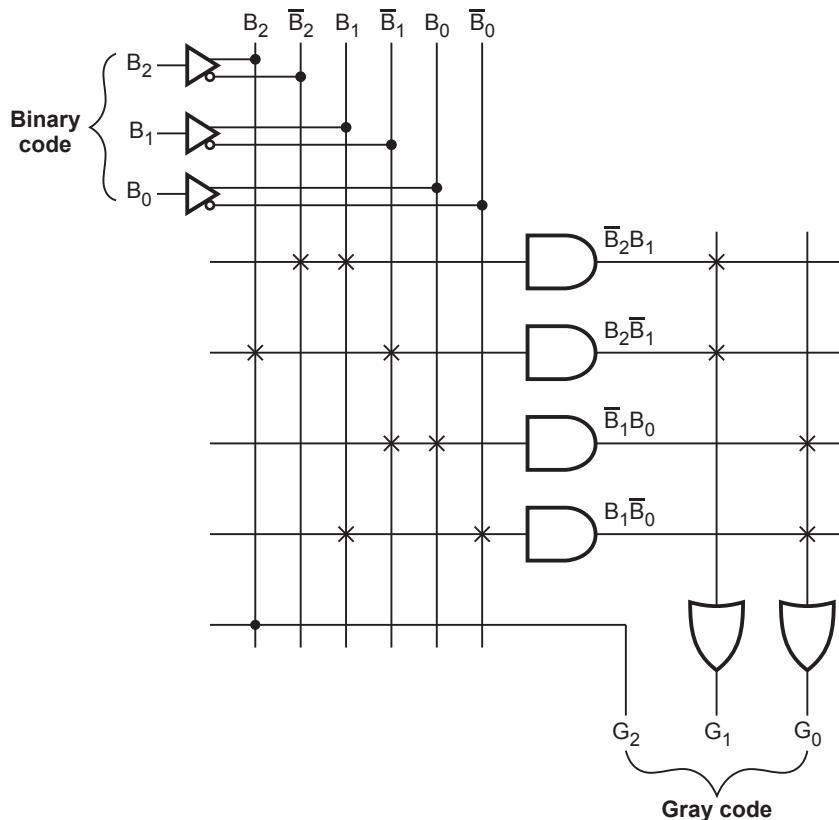
		For G ₁				
		B ₁ B ₀	00	01	11	10
B ₂	0	0	0	1	1	
	1	1	1	0	0	

$$G_1 = \overline{B}_2 B_1 + B_2 \overline{B}_1$$

		For G ₀				
		B ₁ B ₀	00	01	11	10
B ₂	0	0	1	0	1	
	1	0	1	0	1	

$$G_0 = \overline{B}_1 B_0 + B_1 \overline{B}_0$$

Implementation using PLA :



Example 9.3.10 Design 4 input and 6 output combinational circuit using PLA. The input variables are A, B, C and D :

$$Y_1 = \Sigma m(0, 3, 5, 6, 9, 10, 12, 15) \quad Y_2 = \Sigma m(0, 1, 2, 3, 11, 12, 14, 15)$$

$$Y_3 = \Sigma m(0, 4, 8, 12) \quad Y_4 = \Sigma m(0, 2, 3, 5, 7, 8, 12, 13)$$

$$Y_5 = \Sigma m(0, 1, 3, 4, 5, 6, 11, 13, 14, 15)$$

$$Y_6 = \Sigma m(1, 2, 6, 8, 15)$$

SPPU : May-17, Marks 6

Solution :

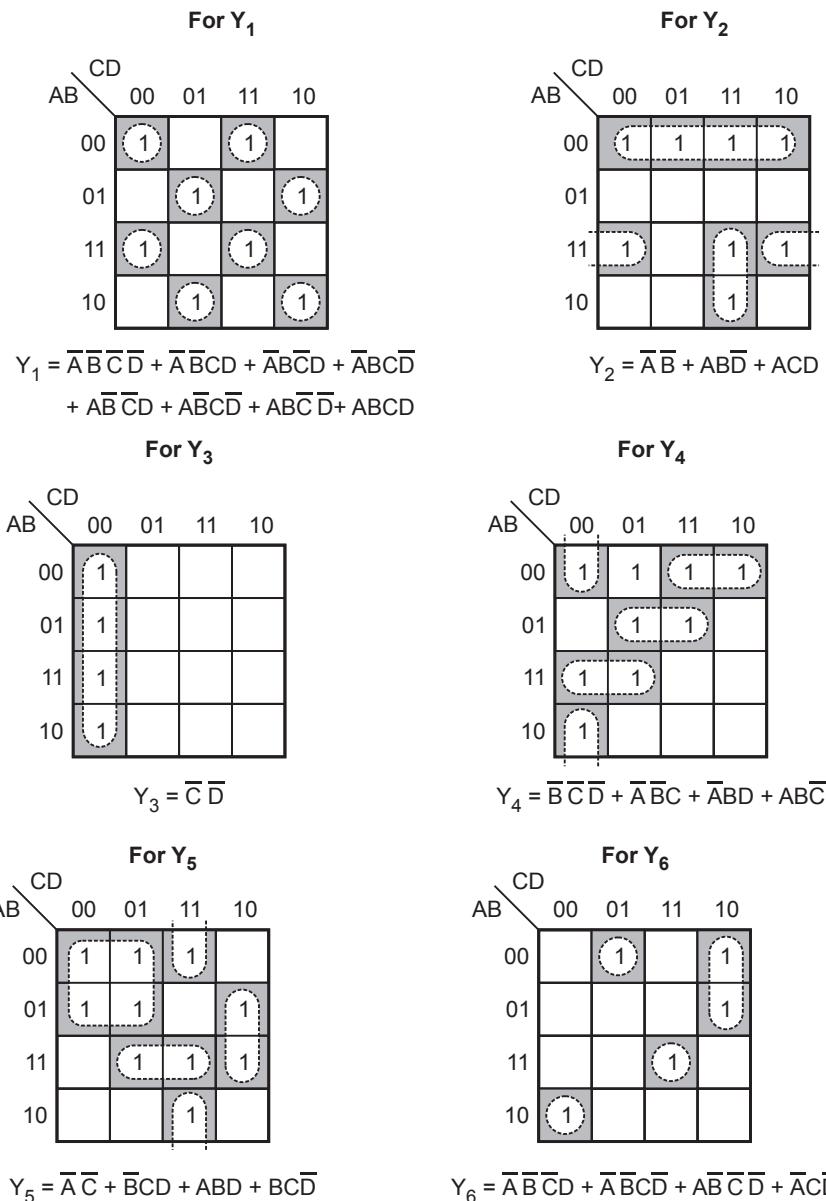


Fig. 9.3.21

After simplification for output functions we have realized that there are 23 product terms which are greater than possible 16 minterms.

To get minimal combinational circuit it is better to implement all 16 possible minterms as product terms as shown in the Fig. 9.3.22.

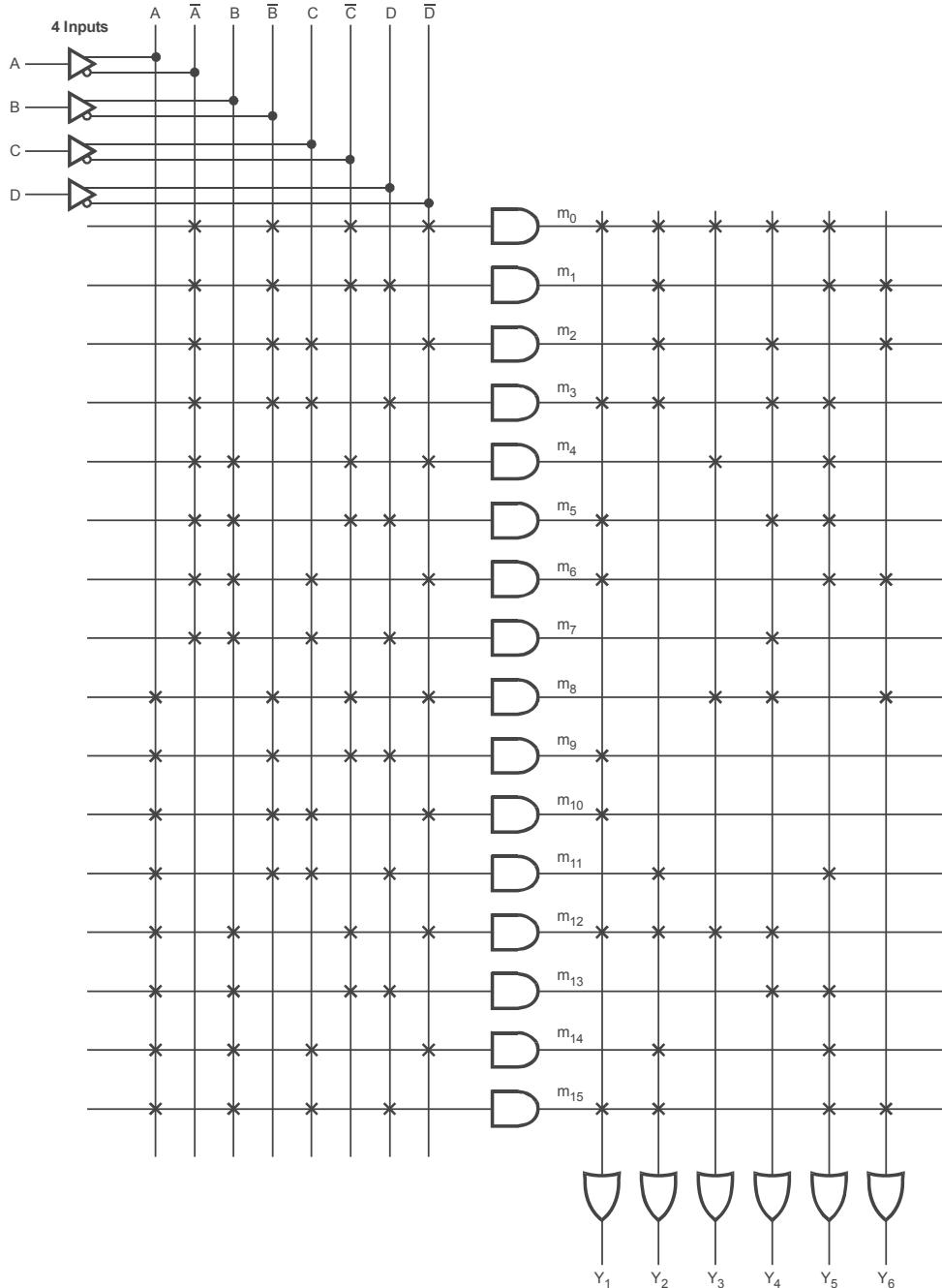


Fig. 9.3.22

Example 9.3.11 Implement the following functions using PLA :

$$F_1(A,B,C) = \sum m(1,2,4,6)$$

$$F_2(A,B,C) = \sum m(0,1,6,7)$$

SPPU : Dec.-19, Marks 6

Solution :

K-map simplification :

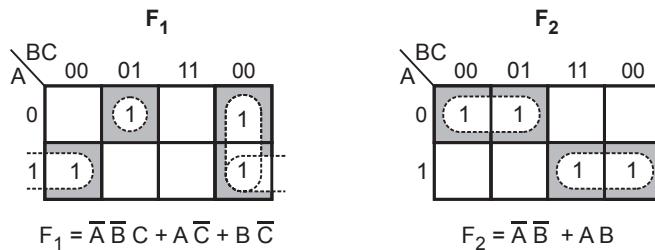


Fig. 9.3.23

Implementation :

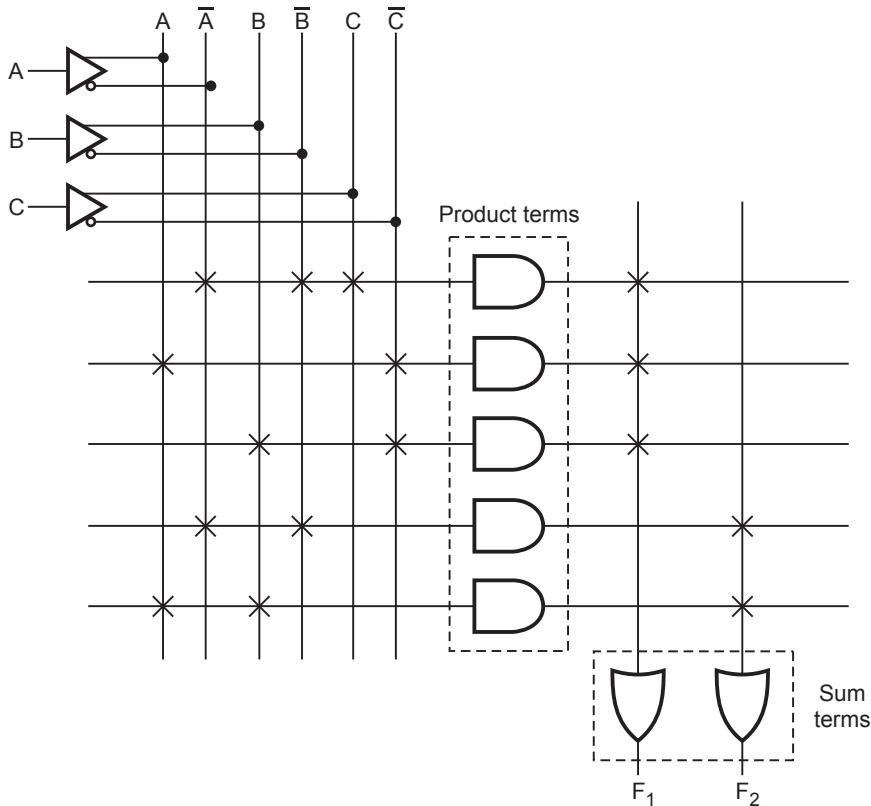


Fig. 9.3.24

Review Questions

1. Explain in brief the internal architecture of a PLA.

SPPU : Dec.-05, May-17, Marks 4

2. Explain PLA.

SPPU : Dec.-08, Marks 2

3. Explain in brief design model of PLA for any code conversion example.

SPPU : Dec.-10, Marks 8

4. Implement the following functions using PLA :

$$F_1(A,B,C) = \Sigma m(1, 2, 4, 6)$$

$$F_2(A,B,C) = \Sigma m(0, 1, 6, 7)$$

SPPU : May-13, Marks 8

5. Design 3 : 8 decoder using PLD.

SPPU : Dec.-13, Marks 4

6. What is PLA ? Explain the input buffer AND and OR matrix in PLA.

SPPU : May-14, Marks 6

7. A combinational circuit is defined by the function :

$$F_1 = \Sigma m(3, 5, 7)$$

$$F_2 = \Sigma m(4, 5, 7)$$

Implement the circuit with PLA having 3 input and 3 product term with 2 output.

SPPU : May-14, Marks 6

8. A combinational circuit is defined by following functions :

$$F_1(A,B,C) = \sum m(0, 2, 4, 5), \quad F_2(A,B,C) = \sum m(1, 3, 6, 7)$$

Implement this circuit using PLA.

SPPU : May-19, Marks 6

9.4 PAL (Programmable Array Logic)

SPPU : May-05, 11, 14, Dec.-08, 12, 13, 14, 18

We have seen that PLA is a device with a programmable AND array and programmable OR array. However, PAL programmable array logic is a programmable logic device with a fixed OR array and a programmable AND array. Because only AND gates are programmable, the PAL is easier to program, but is not as flexible as the PLA. Fig. 9.4.1 shows the array logic of a typical PAL. It has four inputs and four outputs. Each input has buffer and an inverter gate. It is important to note that two gates are shown with one composite graphic symbol with normal and complement outputs. There are four sections. Each section has three programmable AND gates and one fixed OR gate. The output of section 1 is connected to a buffer-inverter gate and then fed back into the inputs of the AND gates, through fuses. This allows the logic designer to feed an output function back as an input variable to create a new function. Such PALs are referred to as **Programmable I/O PALs**.

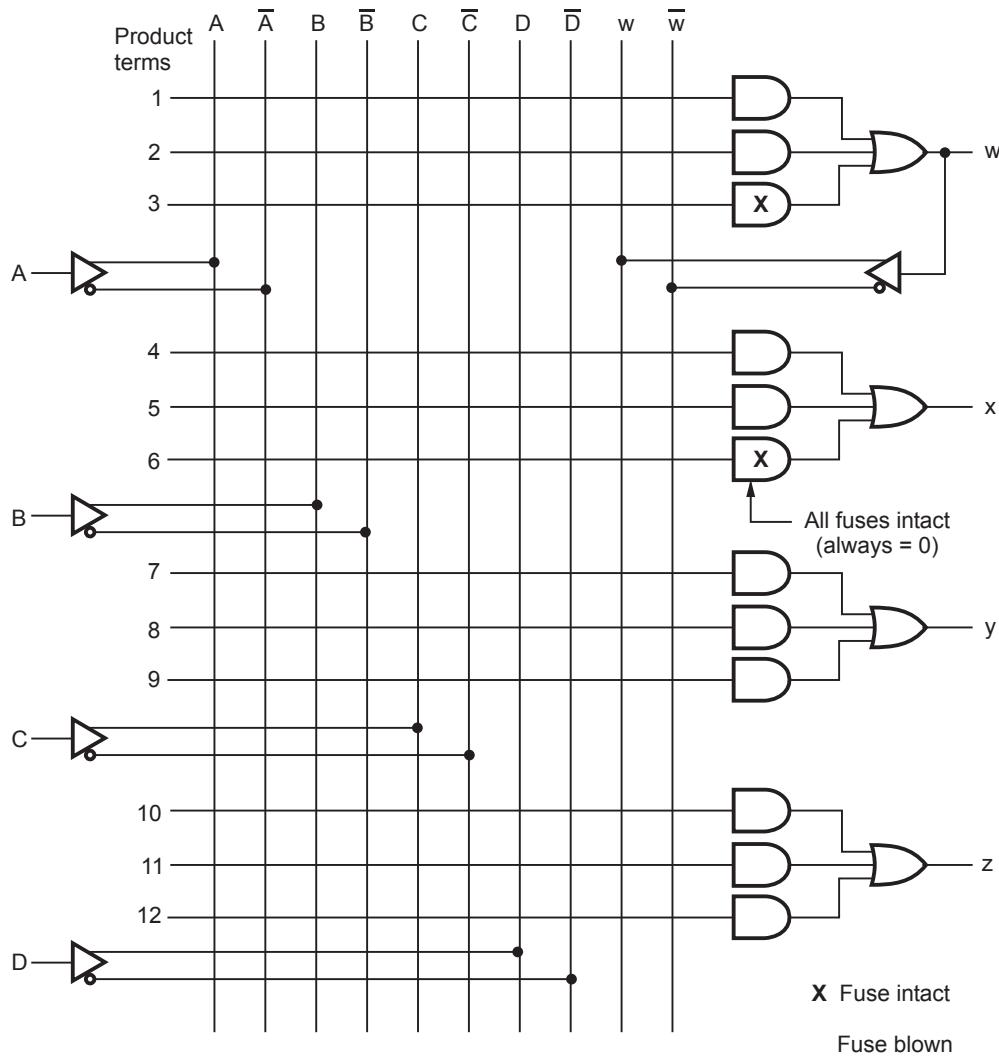


Fig. 9.4.1 Array logic for typical PAL

The commercial PAL devices have more gates than the one shown in Fig. 9.4.1. A typical PAL integrated circuit may have eight inputs, eight outputs, and eight sections, each consisting of an eight wide AND-OR array.

9.4.1 Implementation of Combinational Logic Circuit using PAL

Let us see the implementation of a combinational circuit using PAL with the help of examples.

Illustrative Examples

Example 9.4.1 Implement the following Boolean functions using PAL.

$$w(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9, 12, 13)$$

$$x(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9, 12, 13, 14)$$

$$y(A, B, C, D) = \sum m(2, 3, 8, 9, 10, 12, 13)$$

$$z(A, B, C, D) = \sum m(1, 3, 4, 6, 9, 12, 14)$$

SPPU : May-14, Marks 7

Solution : Step 1 : Simplify the four functions

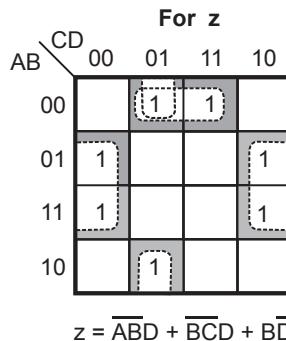
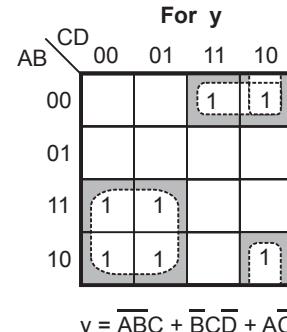
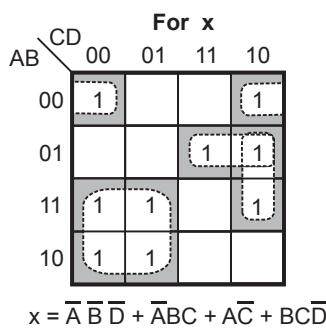
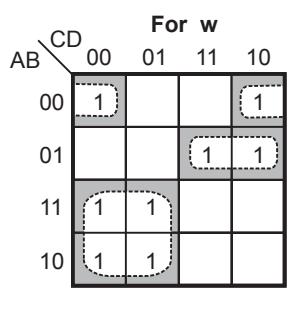


Fig. 9.4.2 K-map simplification

Note that function x has four product terms. Three of them are equal to w. Therefore we can write $x = w + BC\bar{D}$.

Step 2 : Implementation

In the last section we have seen the PLA program table. The program table for PAL is similar to PLA program table. Table 9.4.1 shows PAL program table with product terms, AND inputs and outputs.

Product term	AND Inputs					Outputs
	A	B	C	D	w	
1	0	0	-	0	-	$w = \bar{A} \bar{B} \bar{D} + \bar{A} BC + A \bar{C}$
2	0	1	1	-	-	
3	1	-	0	-	-	
4	-	-	-	-	1	$x = w + BCD$
5	-	1	1	0	-	
6	-	-	-	-	-	
7	0	0	1	-	-	$y = \bar{A} \bar{B} C + \bar{B} C \bar{D} + A \bar{C}$
8	-	0	1	0	-	
9	1	-	0	-	-	
10	0	0	-	1	-	$z = \bar{A} \bar{B} D + \bar{B} \bar{C} D + B \bar{D}$
11	-	0	0	1	-	
12	-	1	-	0	-	

Table 9.4.1 PAL program table

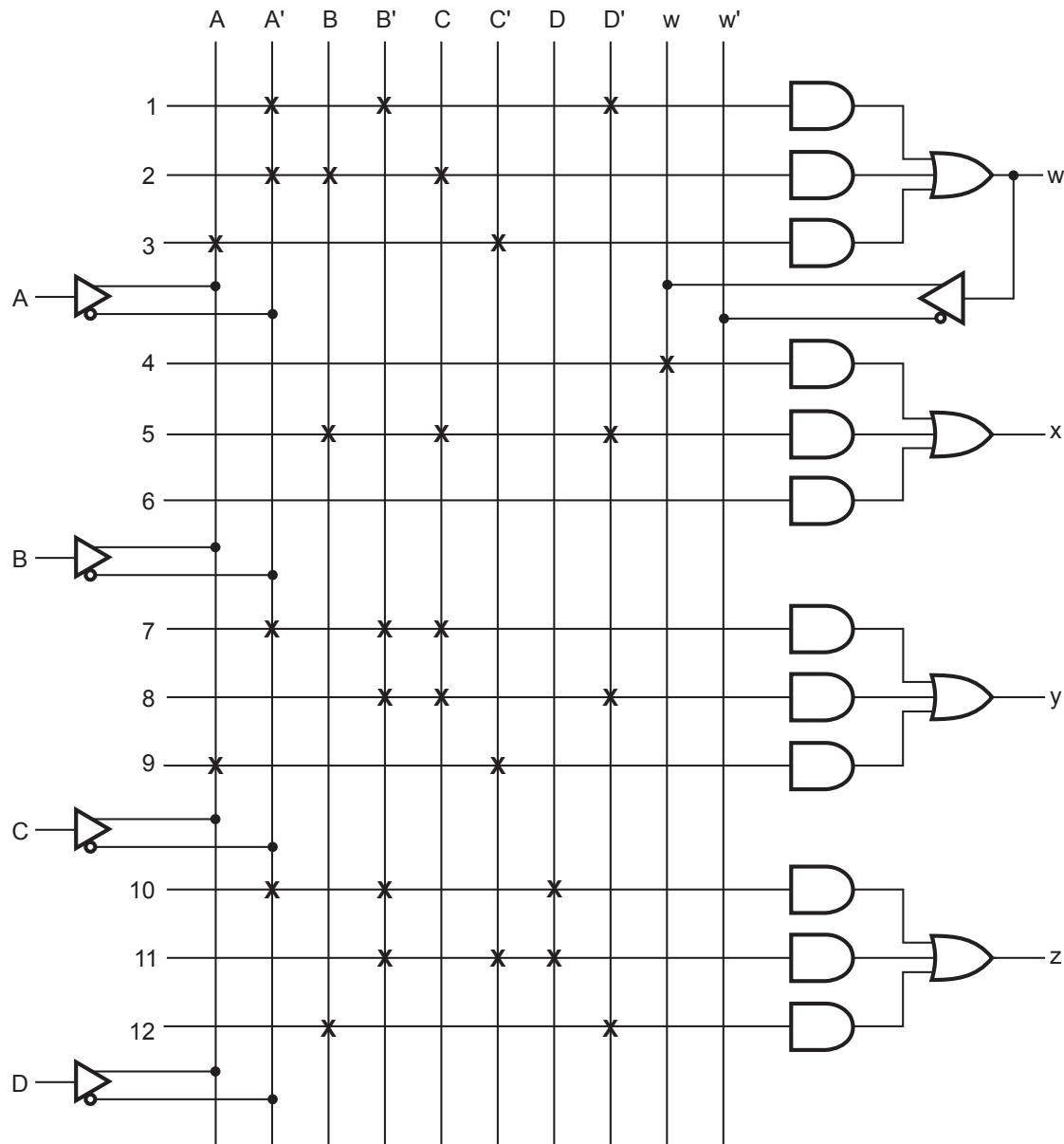


Fig. 9.4.3 Logic diagram

Example 9.4.2 Design BCD to Excess-3 converter using PAL.

Solution :

Step 1 : Derive the truth table of BCD to Excess-3 converter

Decimal	BCD code				Excess-3 code			
	B ₃	B ₂	B ₁	B ₀	E ₃	E ₂	E ₁	E ₀
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table 9.4.2 Truth table for BCD to Excess-3 code converter

Step 2 : Simplify the Boolean functions for Excess-3 code outputs.

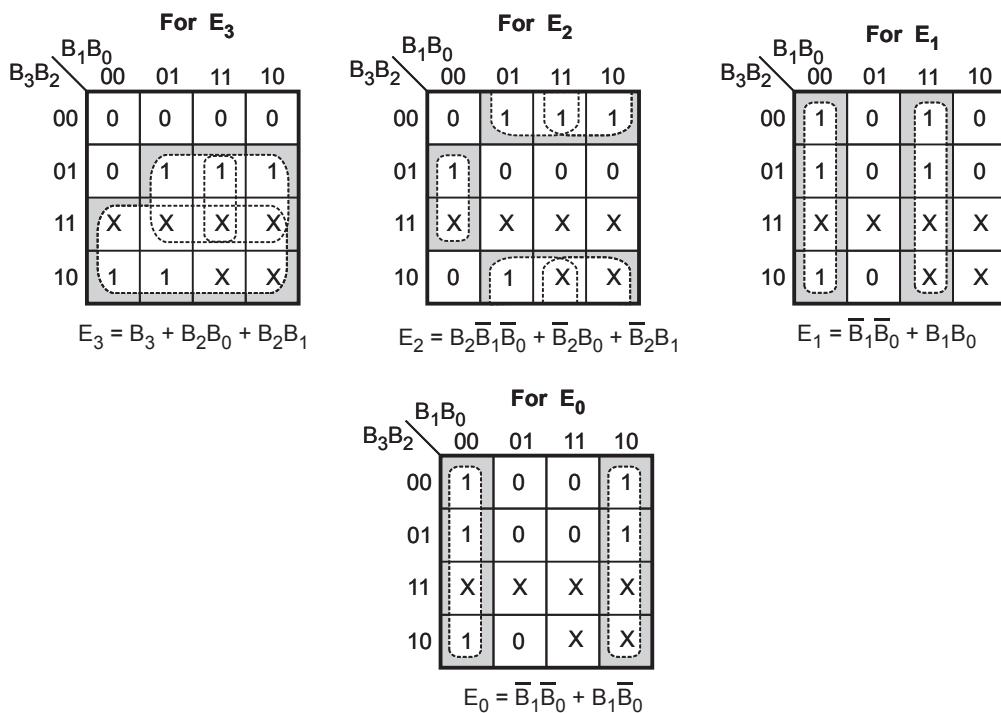
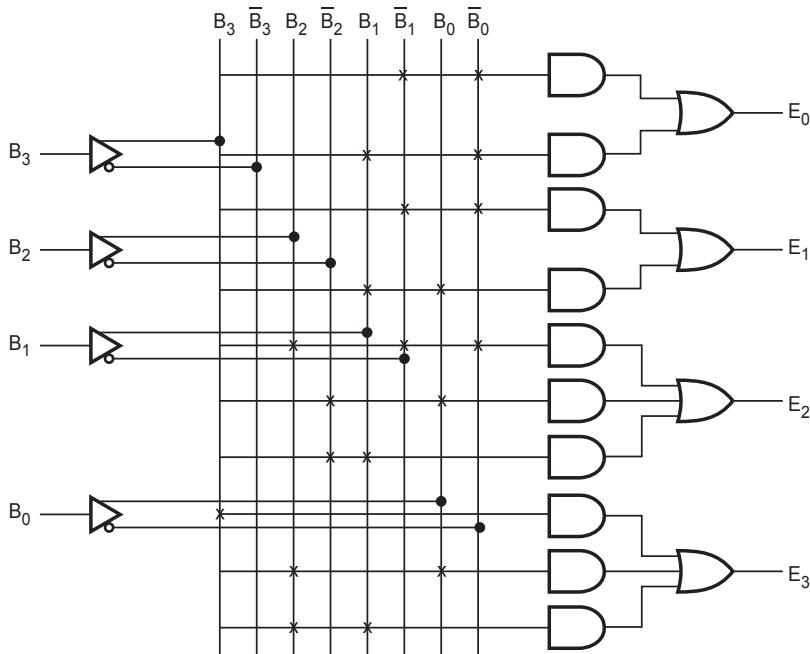


Fig. 9.4.4 K-map simplification

Step 3 : Implementation

Product terms	Inputs				Outputs			
	B ₃	B ₂	B ₁	B ₀	E ₃	E ₂	E ₁	E ₀
B ₃	1	1	—	—	—	1	—	—
B ₂ B ₀	2	—	1	—	1	1	—	—
B ₂ B ₁	3	—	1	1	—	1	—	—
B ₂ B ₁ B ₀	4	—	1	0	0	—	1	—
B ₂ B ₀	5	—	0	—	1	—	1	—
B ₂ B ₁	6	—	0	1	—	—	1	—
B ₁ B ₀	7	—	—	0	0	—	—	1
B ₁ B ₀	8	—	—	1	1	—	—	1
B ₁ B ₀	9	—	—	1	0	—	—	1
					T	T	T	T
								T/C

Table 9.4.3 PAL program table**Fig. 9.4.5 Logic diagram**

Example 9.4.3 Generate the following Boolean functions with a PAL with 4 inputs and 4 outputs.

$$Y_3 = \overline{AB}\overline{CD} + \overline{ABC}\overline{D} + A\overline{BC}\overline{D}, \quad Y_2 = \overline{ABC}\overline{D} + \overline{ABCD} + ABCD$$

$$Y_1 = \overline{ABC} + \overline{ABC} + A\overline{BC} + A\overline{BC}, \quad Y_0 = ABCD$$

Solution : Step 1 : Simplify the Boolean functions

$$\begin{aligned} Y_3 &= \overline{A} B \overline{C} D + \overline{A} B C \overline{D} + A B \overline{C} D = (\overline{A} + A) B \overline{C} D + \overline{A} B C \overline{D} \\ &= B \overline{C} D + \overline{A} B C \overline{D} \end{aligned}$$

$$\begin{aligned} Y_2 &= \overline{A} B C \overline{D} + \overline{A} B C D + A B C D = \overline{A} B C (\overline{D} + D) + (\overline{A} + A) (B C D) \\ &= \overline{A} B C + B C D \end{aligned}$$

$$\begin{aligned} Y_1 &= \overline{A} B \overline{C} + \overline{A} B C + A \overline{B} C + A B \overline{C} \\ &= \overline{A} B (\overline{C} + C) + A \overline{B} C + (\overline{A} + A) B \overline{C} = \overline{A} B + A \overline{B} C + B \overline{C} \end{aligned}$$

$$Y_0 = A B C D$$

Step 2 : Implementation

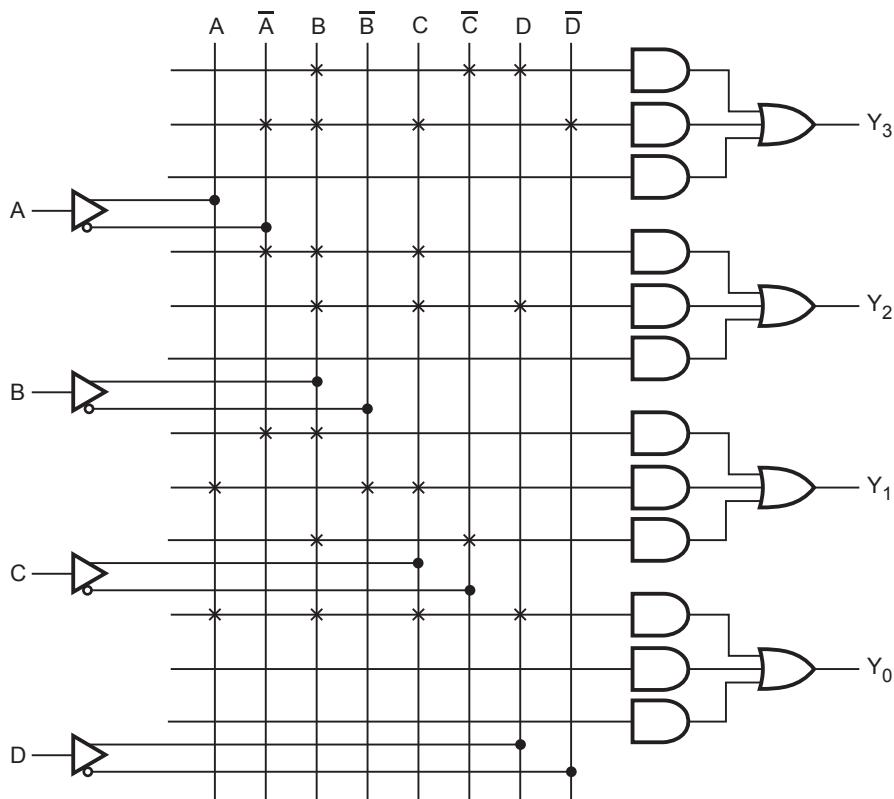


Fig. 9.4.6 Logic diagram

Example 9.4.4 Implement 4 : 1 multiplexer using PAL.

SPPU : Dec.-12,14, Marks 8

Solution : The Boolean function for 4 : 1 multiplexer is

$$Y = E \overline{S}_1 \overline{S}_0 D_0 + E \overline{S}_1 S_0 D_1 + E S_1 \overline{S}_0 D_2 + E S_1 S_0 D_3$$

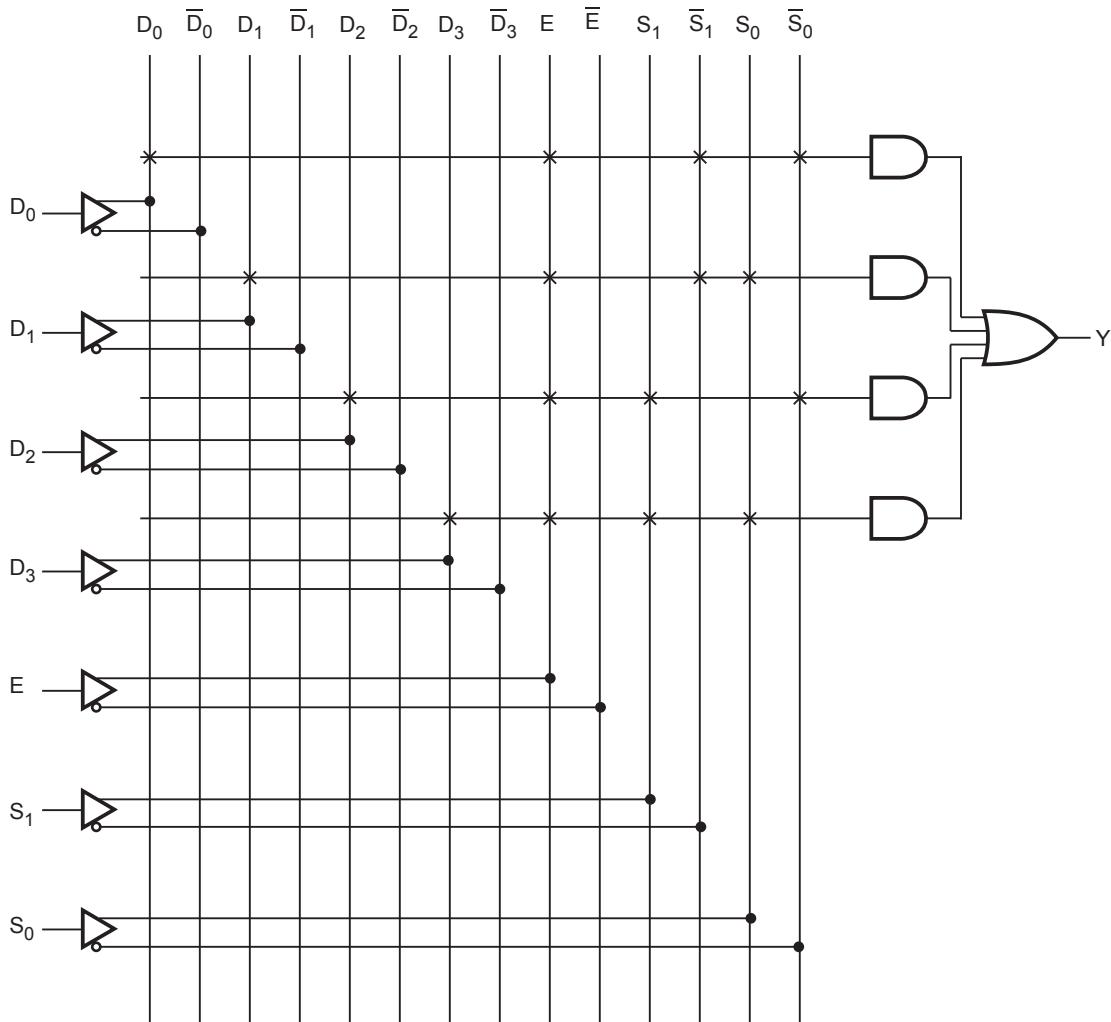


Fig. 9.4.7

Example 9.4.5 A combinational circuit is defined by the function

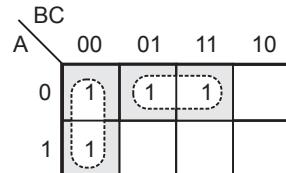
$$F_1(A, B, C) = \sum m(0, 1, 3, 4)$$

Implement this circuit with PAL.

SPPU : Dec.-13, Marks 8

Solution : K-map simplification

$$\therefore F_1 = \bar{B}\bar{C} + \bar{A}C$$



Implementation

Product term	AND inputs			Outputs
	A	B	C	
1	-	0	0	
2	0	-	1	

Table PAL program table

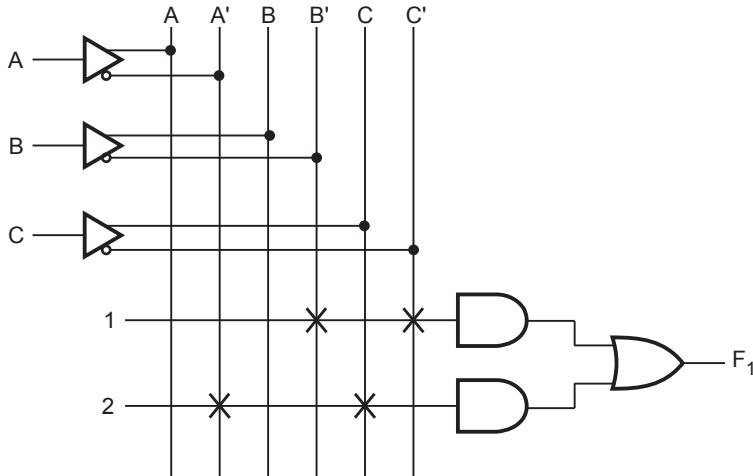


Fig. 9.4.8 Logic diagram

Example 9.4.6 Implement the following function using PAL :

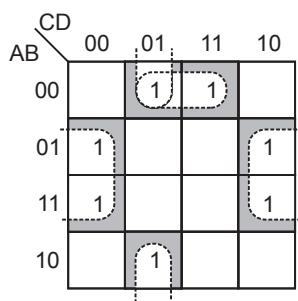
$$F1(A, B, C, D) = \sum m(1, 3, 4, 6, 9, 12, 14)$$

$$F2(A, B, C, D) = \sum m(1, 2, 3, 7, 12, 15).$$

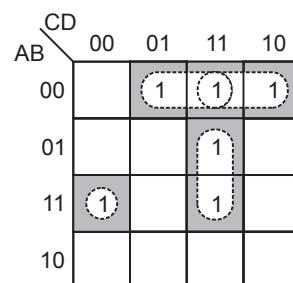
SPPU : Dec.-18, Marks 4

Solution :

Step 1 : K-map simplification



$$F1 = B\bar{D} + \bar{A}\bar{B}D + \bar{B}\bar{C}D$$



$$F2 = AB\bar{C}\bar{D} + \bar{A}\bar{B}D + \bar{A}\bar{B}C + BCD$$

Fig. 9.4.9

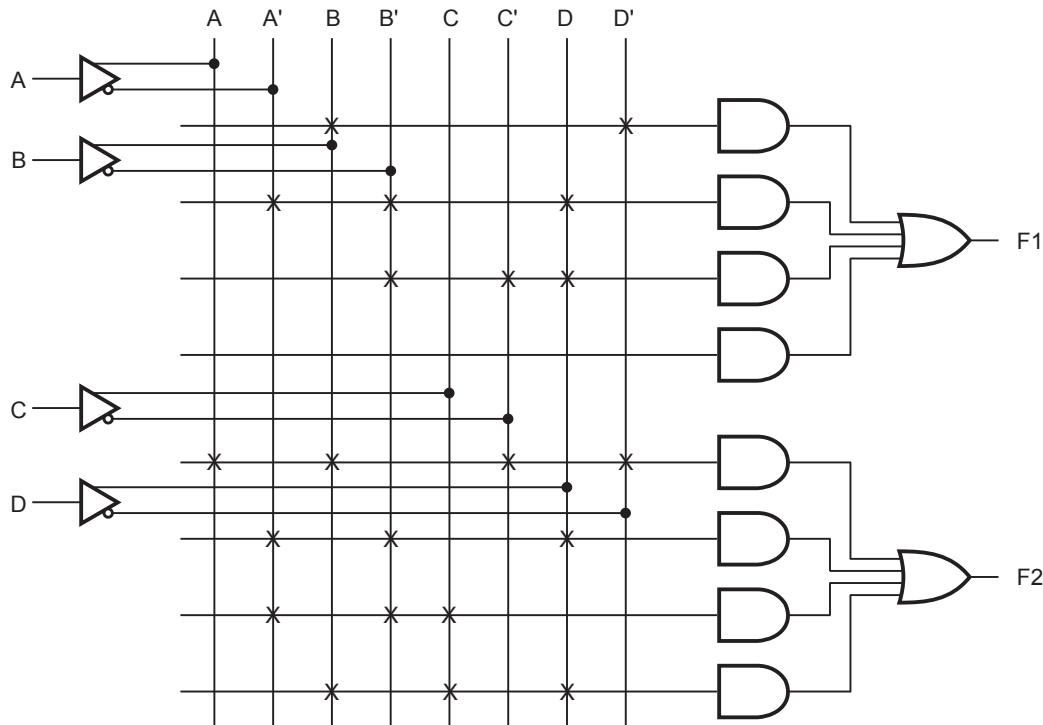


Fig. 9.4.9 (a)

Example for Practice

Example 9.4.6 : A combinational logic circuit is defined by the following function.
 $f_1(a, b, c) = \Sigma(0, 1, 6, 7)$, $f_2(a, b, c) = \Sigma(2, 3, 5, 7)$. Implement the circuit with a PAL having three inputs, three product terms and two outputs.

Review Questions

- What is PAL ?
- Explain PAL. SPPU : Dec.-08, Marks 2
- Give the array logic symbol for eight input AND gate.
- With the help of suitable example explain the basic fixed OR and programmable AND logic of PAL. SPPU : May-05, Marks 4
- Explain the design model of PAL. SPPU : May-11, Marks 8
- Implement the following Boolean function using PAL :

$$W(A, B, C, D) = \Sigma m(0, 2, 6, 7, 8, 9, 12, 13)$$

$$x(A, B, C, D) = \Sigma m(0, 2, 6, 7, 8, 9, 12, 13, 14)$$

$$y(A, B, C, D) = \Sigma m(2, 3, 8, 9, 10, 12, 13)$$

$$z(A, B, C, D) = \Sigma m(1, 3, 4, 6, 9, 12, 14)$$
SPPU : May-14, Marks 7

9.5 Comparison between PROM, PLA and PAL

SPPU : Dec.-05,06,07, May-06,07,08,15,16

Sr. No.	PROM	PLA	PAL
1.	AND array is fixed and OR array is programmable.	Both AND and OR arrays are programmable.	OR array is fixed and AND array is programmable.
2.	Cheaper and simple to use.	Costliest and complex than PAL and PROMs.	Cheaper and simpler.
3.	All minterms are decoded.	AND array can be programmed to get desired minterms.	AND array can be programmed to get desired minterms.
4.	Only Boolean functions in standard SOP form can be implemented using PROM.	Any Boolean functions in SOP form can be implemented using PLA.	Any Boolean functions in SOP form can be implemented using PLA.

Review Question

- What is the difference between PLA, PAL and PROM ?

SPPU : Dec.-05,06,07, May-06,07,08,15,16, Marks 4



Notes

UNIT - V

10

Logic Families

Syllabus

Classification of logic families : Unipolar and Bipolar Logic Families, Characteristics of Digital ICs : Fan-in, Fan-out, Current and voltage parameters, Noise immunity, Propagation Delay, Power Dissipation, Figure of Merits, Operating Temperature Range, power supply requirements.

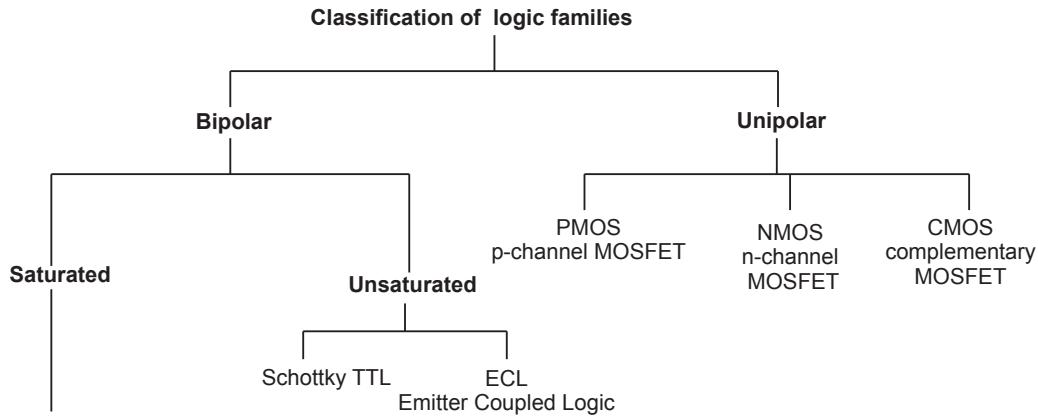
Transistor-Transistor Logic : Operation of TTL NAND Gate (Two input), TTL with active pull up, TTL with open collector output, Wired AND Connection, Tristate TTL Devices, TTL characteristics. CMOS : CMOS Inverter, CMOS characteristics, CMOS configurations- Wired Logic, Open drain outputs.

Contents

10.1	Classification of Logic Families	Dec.-08,14,16,17, May-10,14,18,	Marks 4
10.2	Characteristics of Digital ICs	Dec.-08,18, May-17 Marks 4
10.3	Transistor-Transistor Logic (TTL).	May-05,06,07,08,10,12,13,14,15, 16,17,18,19, Dec.-04,05,06,07,08,10, 12,13,15,17,18,19 Marks 8
10.4	Complementary MOS (CMOS)	May-05,06,07,08,11,12,13,14,15, 16,17,18, Dec.-04,05,06,07,08,10,11, Dec.-12,14,17,18,19 Marks 10
10.5	Interfacing TTL and CMOS Families	Dec.-18, Marks 8
10.6	Comparison between TTL and CMOS Families	May-15,16,19, Dec.-13,19 Marks 6

10.1 Classification of Logic Families SPPU : Dec.-08,14,16,17, May-10,14,18

A digital logic family is a group of compatible devices with the same logic levels and supply voltages. According to components used in the logic family, digital logic families are classified as shown in the Fig. 10.1.1.



- **RTL** : Register Transistor Logic
- **DTL** : Diode Transistor Logic
- **DCTL** : Direct Coupled Transistor Logic
- **I²L** : Integrated Injection Logic
- **HTL** : High Threshold Logic
- **TTL** : Transistor Transistor Logic

Fig. 10.1.1 Classification of logic families

- Logic families are basically classified into two categories. They are
 1. Bipolar logic families
 2. Unipolar logic families
- **Bipolar logic families** : The main elements of a bipolar logic families are diodes and transistors. These are further divided into two types based on BJT operating mode. They are :
 1. Saturated
 2. Non-saturated
- In saturated logic, the transistors are driven to saturation mode, the saturated bipolar logic families are :

1. Resistor-Transistor Logic (RTL)	2. Diode Transistor Logic (DTL)
3. Direct Coupled Transistor Logic (DCTL)	4. Integrated Injection Logic (IIL)
5. High Threshold Logic (HTL)	6. Transistor Transistor Logic (TTL)
- The non saturated bipolar logic families are :
 1. Schottky TTL
 2. Emitter Coupled Logic (ECL)
- **Unipolar logic families** : MOS devices are unipolar devices and only MOSFETs are employed in these MOS logic circuits. The MOS logic families are :
 1. PMOS
 2. NMOS
 3. CMOS

Review Question

1. What is logic family ? Give the classification of logic families.

SPPU : May-10,14,18, Dec.-08,14,16,17, Marks 4

10.2 Characteristics of Digital ICs

SPPU : Dec.-08,18, May-17

Propagation Delay : The propagation delay of a gate is basically the time interval between the application of an input pulse and the occurrence of the resulting output pulse. The propagation delay is a very important characteristic of logic circuits because it limits the speed at which they can operate. The shorter the propagation delay, the higher the speed of the circuit and vice-versa.

Power Dissipation : The amount of power that an IC dissipates is determined by the average supply current, I_{CC} , that it draws from the V_{CC} supply. It is the product of I_{CC} and V_{CC} .

Current and Voltage Parameter

$V_{IH(min)}$ - High-Level Input Voltage : It is the minimum voltage level required for a logical 1 at an input. Any voltage below this level will not be accepted as a HIGH by the logic circuit.

$V_{IL(max)}$ - Low-Level Input Voltage : It is the maximum voltage level required for a logic 0 at an input. Any voltage above this level will not be accepted as a LOW by the logic circuit.

$V_{OH(min)}$ - High-Level Output Voltage : It is the minimum voltage level at a logic circuit output in the logical 1 state under defined load conditions.

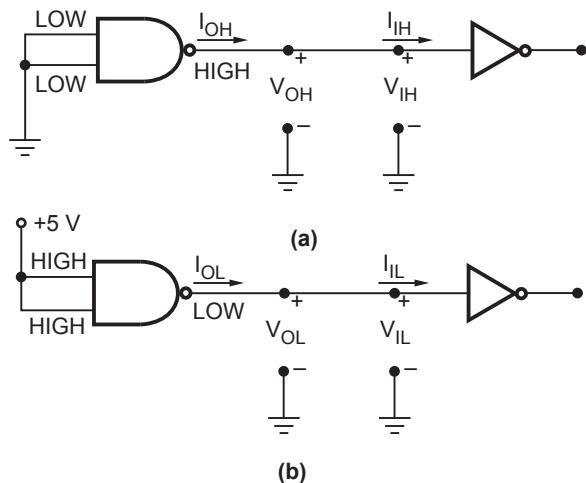


Fig. 10.2.1 Currents and voltages in the two logic states

V_{OL(max)} - Low-Level Output Voltage : It is the maximum voltage level at a logic circuit output in the logical 0 state under defined load conditions.

I_{IH} - High-Level Input Current : It is the current that flows into an input when a specified high-level voltage is applied to that input.

I_{IL} - Low-Level Input Current : It is the current that flows into an input when a specified low-level voltage is applied to that input.

I_{OH} - High-Level Output Current : It is the current that flows from an output in the logical 1 state under specified load conditions.

I_{OL} - Low-Level Output Current : It is the current that flows from an output in the logical 0 state under specified load conditions.

Noise Margin : The **noise immunity** of a logic circuit refers to the circuit's ability to tolerate the noise without causing spurious changes in the output voltage. To avoid this problem due to noise, voltage level $V_{IH(min)}$ is kept at a few fraction of volts below $V_{OH(min)}$ and voltage level $V_{IL(max)}$ is kept above $V_{OL(max)}$, at the design time.

V_{NH} is the difference between the lowest possible HIGH output, $V_{OH(min)}$ and the minimum voltage, $V_{IH(min)}$ required for a HIGH input. This voltage difference, V_{NH} is called high-state noise margin. Similarly, we have low-state noise margin. It is the voltage difference between the largest possible low output, $V_{OL(max)}$ and the maximum voltage, $V_{IL(max)}$ required for a LOW input.

In short we can write as,

$$V_{NH} = V_{OH(min)} - V_{IH(min)} \quad \text{and}$$

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

Fan-in and Fan-out : The maximum number of inputs of several gates that can be driven by the output of a logic gate is decided by the parameter called **fan-out**. In general, the fan-out is defined as the maximum number of inputs of the same IC family that the gate can drive maintaining its output levels within the specified limits.

The **fan-in** of a digital logic gate refers to the number of inputs.

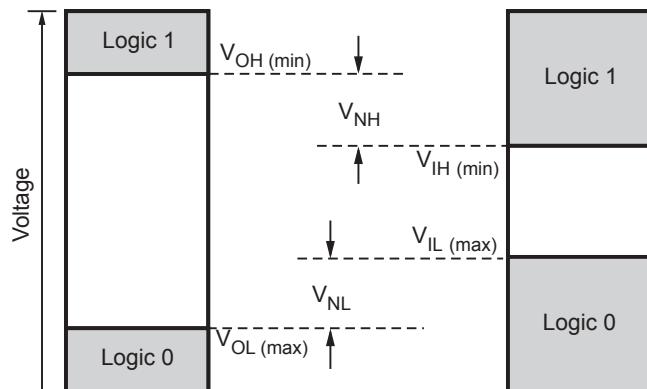


Fig. 10.2.2 Noise margins

Speed Power Product (Figure of Merit)

- In general, for any digital IC, it is desirable to have shorter propagation delays (higher speed) and lower values of power dissipation. There is usually a trade-off between switching speed and power dissipation in the design of a logic circuit i.e. speed is gained at the expense of increased power dissipation. Therefore, a common means for measuring and comparing the overall performance of an IC family is the **Speed-Power Product (SPP)**. It is also called **Figure of Merit**.

Current Sinking

- A device output is said to **sink current** when current flows from the power supply, through the load and through the device output to ground. This is illustrated in Fig. 10.2.3 (a).

Current Sourcing

- A device output is said to **source current** when current flows from the power supply, out of the device output and through the load to ground. This is illustrated in Fig. 10.2.3 (b).

Operating Temperature Range

- It is the temperature range specified by the logic family within which devices are guaranteed to work reliably.

Power Supply Requirements

- Power supply requirements differ from logic family to family. For example, it is 5V for TTL family and 3-15 volts for CMOS family. Further more, power supply tolerance also depends on logic family. For example, for 74 series TTL family it is ± 0.25 V and for 54 series TTL family it is ± 0.5 V.

Review Questions

- State and explain any four characteristics of digital ICs. **SPPU : May-17, Dec.-08,18, Marks 4**
- Define 1) V_{OH} , V_{IL} 2) V_{OL} , V_{IL} 3) Noise margin.
- Define fan-out.
- Define fan-in.
- What is propagation delay ?
- What is noise margin ?
- Define current sinking and current sourcing.
- What is speed power product ?

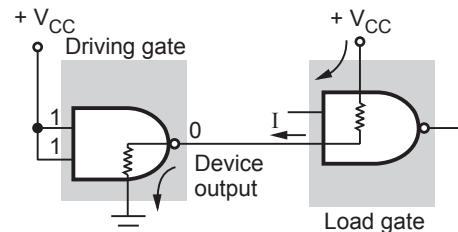


Fig. 10.2.3 (a) Current sinking

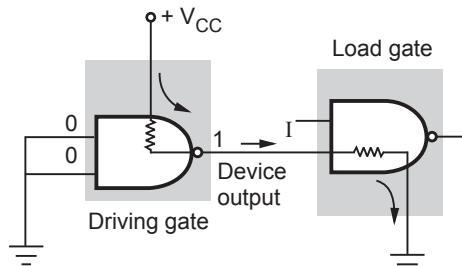


Fig. 10.2.3 (b) Current sourcing

10.3 Transistor-Transistor Logic (TTL)

**SPPU : May-05,06,07,08,10,12,13,14,15,16,17,18,19,
Dec.-04,05,06,07,08,10,12,13,15,17,18,19**

- Transistor-transistor logic, TTL, is named for its dependence on transistors alone to perform basic logic operations. The first version, which is now known as standard TTL, was developed in 1965 and is rarely used in today's systems. Through the years, the basic design has been modified to improve its performance in several respects and as a consequence, a number of subfamilies have evolved. In this section we are going to study the basic transistor configurations in TTL and its subfamily circuits along with their characteristics.

10.3.1 2-Input TTL NAND Gates

- The Fig. 10.3.1 (a) shows the circuit diagram of 2-input NAND gate. Its input structure consists of multiple-emitter transistor and output structure consists of totem-pole output. Here, Q_1 is an NPN transistor having two emitters, one for each input to the gate. Although this circuit looks complex, we can simplify its analysis by using the diode equivalent of the multiple-emitter transistor Q_1 , as shown in Fig. 10.3.1 (b). Diodes D_2 and D_3 represent the two E-B junctions of Q_1 and D_4 is the collector-base (C-B) junction.

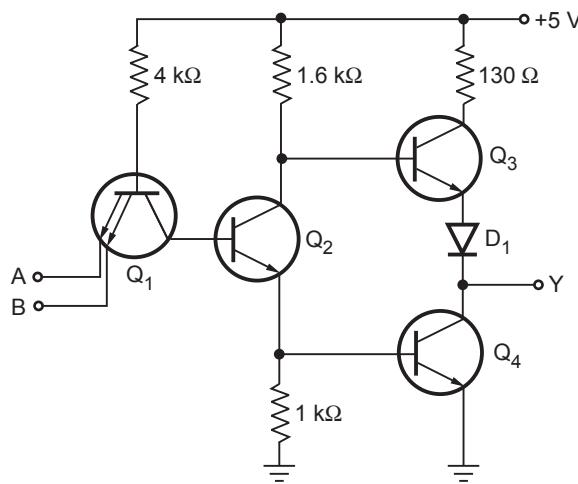


Fig. 10.3.1 (a) Two input TTL NAND gate

- The input voltages A and B are either LOW (ideally grounded) or HIGH (ideally + 5 volts). If either A or B or both are low, the corresponding diode conducts and the base of Q_1 is pulled down to approximately 0.7 V. This reduces the base voltage of Q_2 to almost zero. Therefore, Q_2 cuts off. With Q_2 open, Q_4 goes into cut-off and the Q_3 base is pulled HIGH. Since Q_3 acts as an emitter follower, the Y output is pulled up to a HIGH voltage. On the other hand, when A and B both are HIGH, the emitter diode of Q_1 are reversed biased making them off. This causes the collector diode D_4 to go into forward conduction. This forces Q_2 base to go HIGH. In turn, Q_4 goes into saturation, producing a low output. Table 10.3.1 summarizes all input and output conditions.

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

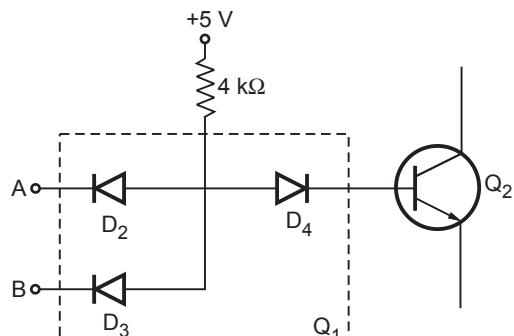
Fig. 10.3.1 (b) Diode equivalent for Q₁

Table 10.3.1 Truth table for 2-input NAND gate

- Without diode D₁ in the circuit, Q₃ will conduct slightly when the output is low. To prevent this, the diode is inserted; its voltage drop keeps the base-emitter diode of Q₃ reverse-biased. In this way, only Q₄ conducts when the output is low.

10.3.2 Totem-Pole Output / Active Pull-Up

- Fig. 10.3.2 shows an highlighted output configuration. Transistor Q₃ and Q₄ form a totem-pole. Such a configuration is known as **active pull-up** or **totem pole** output.

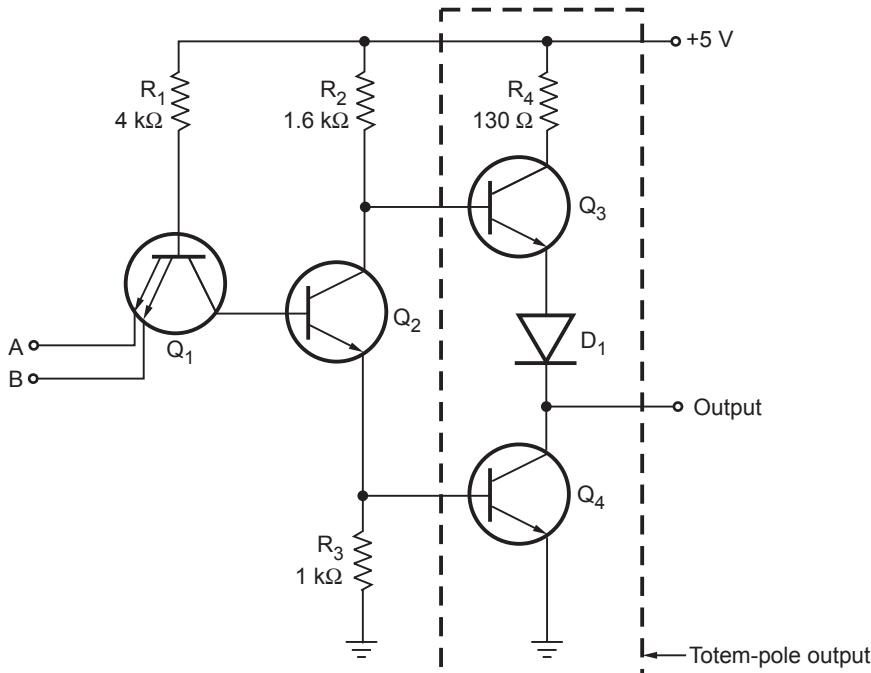


Fig. 10.3.2 Two input NAND gate with totem-pole output

- The active pull-up formed by Q_3 and Q_4 has specific advantage. Totem-pole transistors are used because they produce a LOW output impedance.
- Either Q_3 acts as an emitter follower (HIGH output), or Q_4 is saturated (LOW output).
- When Q_3 is conducting, the output impedance is approximately $70\ \Omega$; when Q_4 is saturated, the output impedance is only $12\ \Omega$. Either way, the output impedance is low. This means that the output voltage can change quickly from one state to the other because any stray output capacitance is rapidly charged or discharged through the low output impedance. Thus the propagation delay is low in totem-pole TTL logic.

10.3.3 Wired Logic - Open Collector Output

- One problem with totem pole output is that two outputs cannot be tied together. See Fig. 10.3.3, where the totem pole outputs of two separate gates are connected together at point X.
- Suppose that the output of gate A is high (Q_{3A} ON and Q_{4A} OFF) and the output of gate B is low (Q_{3B} OFF and Q_{4B} ON). In this situation transistor Q_{4B} acts as a load for Q_{3A} . Since Q_{4B} is a low resistance load, it draws high current around 55 mA . This current might not damage Q_{3A} or Q_{4B} immediately, but over a period of time can cause overheating and deterioration in performance and eventual device failure.

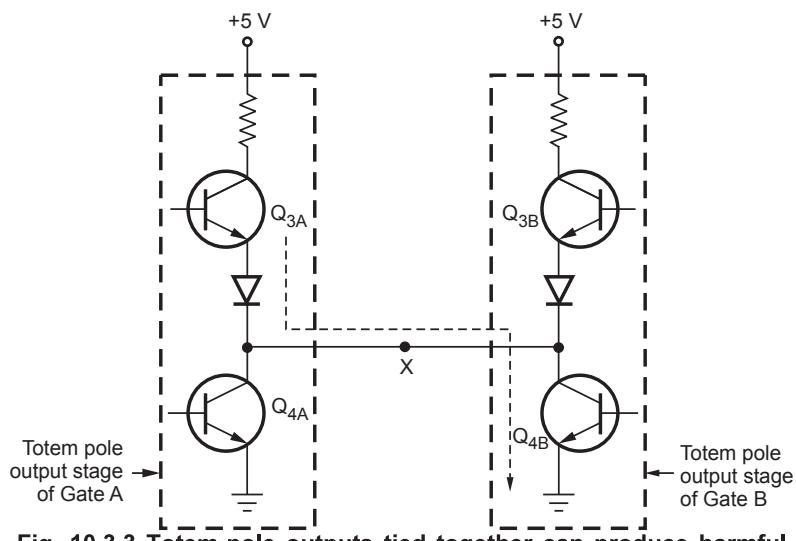


Fig. 10.3.3 Totem-pole outputs tied together can produce harmful current

- Some TTL devices provide another type of output called **open collector output**. The outputs of two different gates with open collector output can be tied together. This is known as **wired logic**.

- Fig. 10.3.4 shows a 2-input NAND gate with an open-collector output which eliminates the pull-up transistor Q_3 , D_1 and R_4 . The output is taken from the open collector terminal of transistor Q_4 .
- Because the collector of Q_4 is open, a gate like this will not work properly until you connect an external pull-up resistor, as shown in Fig. 10.3.5.

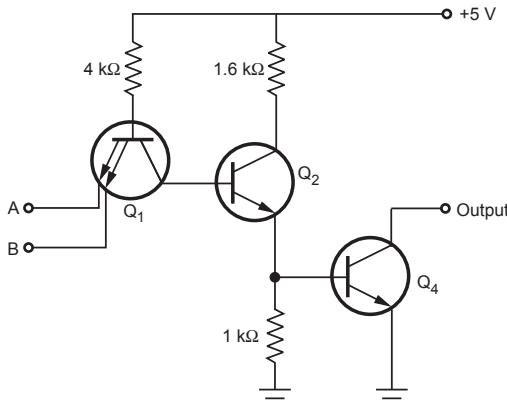


Fig. 10.3.4 Open collector 2-input TTL NAND gate

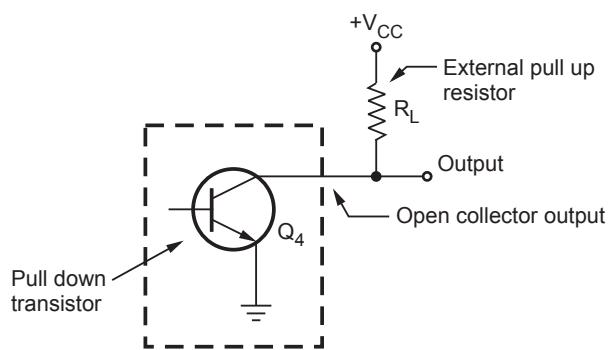
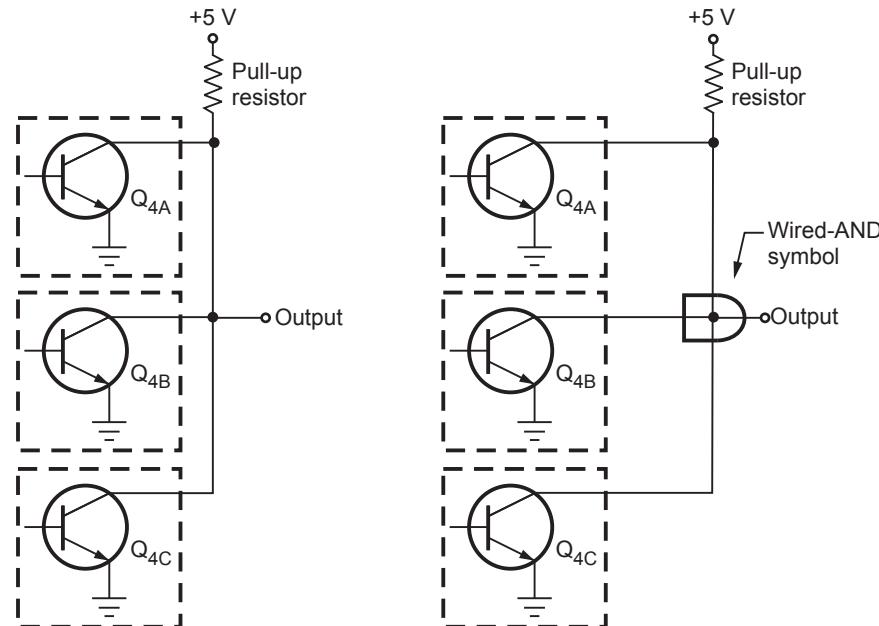


Fig. 10.3.5 Open collector output with pull-up resistor

- When Q_4 is ON, output is low and when Q_4 is OFF output is tied to V_{CC} through an external pull up resistor.



(a) Open collector outputs connected together

(b) Wired-AND output with special AND gate symbol

Fig. 10.3.6

- As mentioned earlier, the open collector outputs of two or more gates can be connected together, as shown in the Fig. 10.3.6 (a). The connection is called a **wired-AND** and represented schematically by the special AND gate symbol as shown in Fig. 10.3.6 (b).

10.3.4 Comparison between Totem-Pole and Open-Collector Outputs

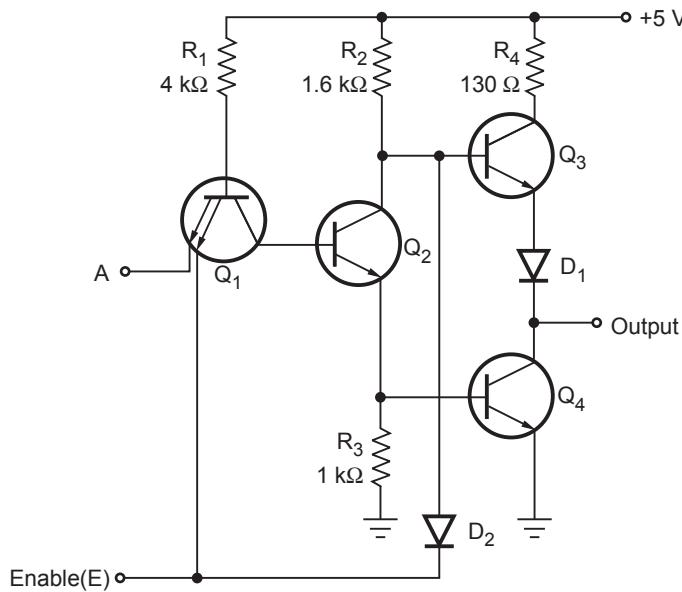
Table 10.3.2 summarizes the difference between totem-pole and open collector outputs.

Sr. No.	Totem-pole	Open collector
1.	Output stage consists of pull-up transistor (Q_3), diode resistor and pull-down transistor (Q_4).	Output stage consists of only pull-down transistor.
2.	External pull-up resistor is not required.	External pull-up resistor is required for proper operation of gate.
3.	Output of two gates cannot be tied together.	Output of two gates can be tied together using wired AND technique.
4.	Operating speed is high.	Operating speed is low.

Table 10.3.2 Comparison of totem-pole and open collector output

10.3.5 Tri-state TTL Inverter

- The tristate configuration is a third type of TTL output configuration. It utilizes the high-speed operation of the totem-pole arrangement while permitting outputs to be wired-ANDED (connected together). It is called tristate TTL because it allows three possible output stages : **HIGH**, **LOW** and **high-impedance**.
- We know that transistor Q_3 is ON when output is HIGH and Q_4 is ON when output is LOW. In the high impedance state both transistors, transistors Q_3 and Q_4 in the totem-pole arrangement are turned OFF. As a result, the output is open or floating, it is neither LOW nor HIGH.
- Fig. 10.3.7 shows the simplified circuit for tristate inverter. It has two inputs A and E.
- A is the normal logic input whereas E is an ENABLE input. When ENABLE input is HIGH, the circuit works as a normal inverter. Because when E is HIGH, the state of the transistor Q_1 (either ON or OFF) depends on the logic input A, and the additional component diode is open circuited as its cathode is at logic HIGH.

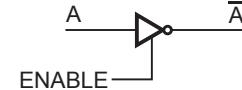


A	E	Output
0	1	1
1	1	0
X	0	Floating

Truth table

Fig. 10.3.7 Tristate TTL inverter

- When ENABLE input is LOW, regardless of the state of logic input A, the base-emitter junction of Q₁ is forward biased and as a result it turns ON. This shunts the current through R₁ away from Q₂ making it OFF.
- As Q₂ is OFF, there is no sufficient drive for Q₄ to conduct and hence Q₄ turns OFF.
- The LOW at ENABLE input also forward-biases diode D₂, which shunt the current away from the base of Q₃, making it OFF. In this way, when ENABLE input is LOW, both transistors are OFF and output is at high impedance state.
- Fig. 10.3.8 shows the logic symbols for tristate inverter. In above case circuit operation is enabled when ENABLE input is HIGH. Therefore, ENABLE input is active high. The logic symbol for active high enable input is shown in Fig. 10.3.8 (a). In some circuits ENABLE input can be active LOW, i.e. circuit operates when ENABLE input is LOW. The logic symbol for active low ENABLE input is shown in the Fig. 10.3.8 (b).



(a) Logic symbol for active high enable input



(b) Logic symbol for active low enable input

Fig. 10.3.8

10.3.6 Tri-state TTL NAND Gate

- Fig. 10.3.9 shows the tri-state 2-input TTL NAND gate. As shown in Fig. 10.3.9, the Q_1 has three inputs. Two of them are the inputs of NAND gate and the remaining input is an enable input (E).

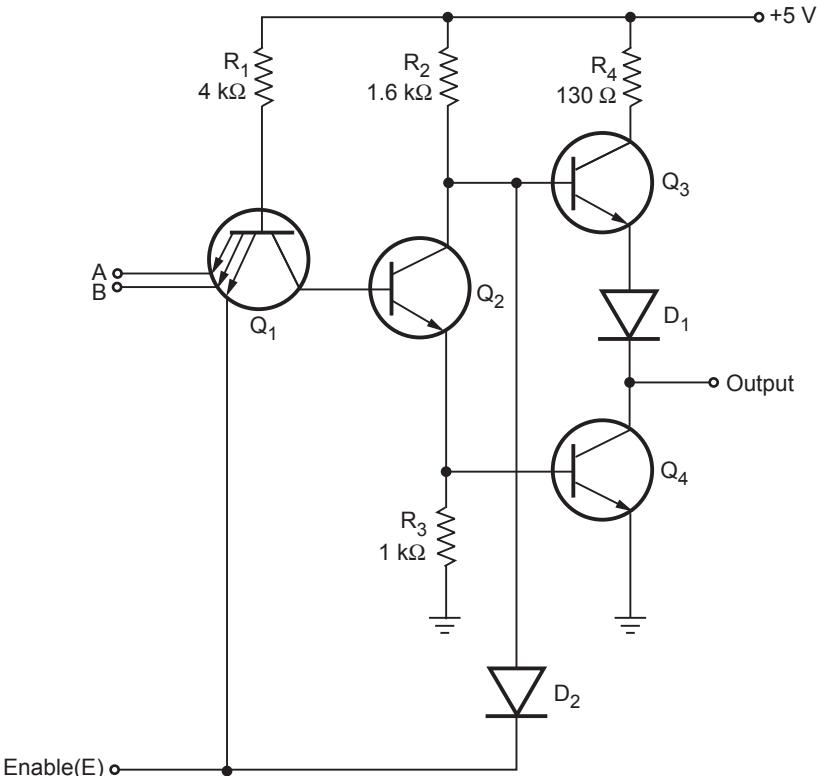


Fig. 10.3.9 Tri-state 2-input TTL NAND gate

- The function of enable input E and diode D_2 is same as that of tri-state inverter.
- When E input is HIGH, the state of transistor Q_1 (either ON or OFF) depends on the two inputs of NAND gate and the circuit will operate as a two-input NAND gate.
- When E input is low, regardless of the state of logic inputs A and B. The base-emitter junction of Q_1 is forward biased and as a result it turns ON. This shunts the current through R_1 away from Q_2 making it OFF. As Q_2 is OFF, there is no sufficient drive for Q_4 to conduct and hence Q_4 turns off. The LOW at ENABLE input also forward-biases diode D_2 , which shunt the current away from

the base of Q_3 , making it OFF. In this way, when ENABLE input is LOW, both transistors are OFF and output is at high impedance state.

10.3.7 Standard TTL Characteristics

- In 1964 Texas Instruments Corporation introduced the standard TTL ICs, 54/74 series. There are several series/ subfamilies in the TTL family of logic devices. Let us see the characteristics of standard TTL family.

Supply voltage and temperature range

- Both the 74 series and 54 series operate on supply voltage of 5 V. The 74 series works reliably over the range 4.75 V to 5.25 V, while the 54 series can tolerate a supply variation of 4.5 to 5.5 V.
- The 74 series devices are guaranteed to work reliably over a temperature range of 0 to $^{\circ}\text{C}$ where as 54 series devices can handle temperature variations from -55 to $^{\circ}\text{C}$.

Voltage levels and noise margin

- Table 10.3.3 shows the input and output logic voltage levels for the standard 74 series.
- Looking at Table 10.3.3 we can say that, in the worst case, there is difference of 0.4 V between the driver output voltages and the required load input voltages. For instance, the worst-case low values are

$$V_{OL(max)} = 0.4 \text{ V driver output}$$

$$V_{IL(max)} = 0.8 \text{ V load input}$$

Similarly, the worst-case high values are

$$V_{OH(min)} = 2.4 \text{ V driver output}$$

$$V_{IH(min)} = 2 \text{ V load input}$$

- In either case, the difference is 0.4 V. This difference is called *noise margin*. For TTL, Low state noise margin, V_{NL} and high state noise margin, V_{NH} both are equal and 0.4 V. This is illustrated in Fig. 10.3.10.

Voltages	Minimum	Typical	Maximum
V_{OL}	—	0.2	0.4
V_{OH}	2.4	3.4	—
V_{IL}	—	—	0.8
V_{IH}	2.0	—	—

Table 10.3.3 Voltage levels

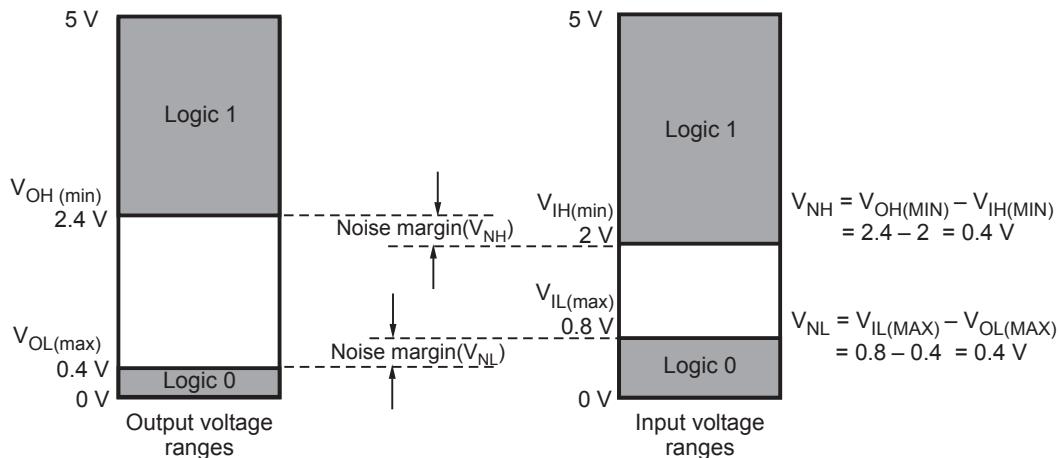


Fig. 10.3.10 TTL logic levels and noise margin

Power dissipation and propagation delay

- A standard TTL gate has an average power dissipation of about 10 mW.
- The propagation delay time of a TTL gate is approximately 10 nanoseconds.

Fan-out

- A standard TTL output can typically drive 10 standard TTL inputs. Therefore, standard TTL has fan-out 10.
- Table 10.3.4 summarizes the characteristics of standard TTL.

Characteristics	Values
Supply voltage	For 74 series - (4.75 to 5.25) units For 54 series - (4.5 to 5.5) units
Temperature range	For 74 series - (0 °C to 70 °C) For 54 series - (- 55 °C to 125 °C)
Voltage levels	$V_{OL(\text{max})}$ - 0.4 V $V_{OH(\text{min})}$ - 2.4 V $V_{IL(\text{max})}$ - 0.8 V $V_{IH(\text{min})}$ - 2.0 V
Noise margin	0.4 V
Power dissipation	10 mW per gate
Propagation delay	Typically 10 ns
Fan-out	10

Table 10.3.4 Standard TTL characteristics

10.3.8 Advantages and Disadvantages of TTL Family**Advantages of TTL**

1. High speed operation. Fastest among the saturated logic families. The propagation delay time is about 10 ns.

2. Moderate power dissipation.
3. Available in commercial and military versions.
4. Available for wide range of functions.
5. Low cost.
6. Moderate packaging density.

Disadvantages of TTL

1. Higher power dissipation than CMOS.
2. Lower noise immunity than CMOS.
3. Less fan-out than CMOS.

Review Questions

1. With neat circuit diagram explain the operation of two-input TTL NAND gates.

SPPU : May-06,10,13,17, Dec.-07,17, Marks 8

OR

Draw 2-input standard TTL NAND gate with totem pole. Explain operation of transistor (ON/OFF) with suitable input conditions and truth table.

SPPU : May-10,19, Dec.-12

OR

Explain the working of two input TTL NAND gate with active pull up. Consider various input, output states for explanation.

SPPU : May-12, Marks 8

2. What is totem-pole output ? Explain with the help of circuit diagram.

SPPU : Dec.-05, 06, May-05,08, Marks 6

3 Explain the wired logic output of TTL with neat diagram.

SPPU : Dec.-05,06,07, May-14, Marks 3

4. Draw and explain wired AND gate in detail.

SPPU : Dec.-18, Marks 6

5. Compare different types of output configurations in case of TTL family.

SPPU : Dec.-05, May-07, Marks 4

6. Give the advantages and disadvantages of totem-pole output stage arrangement.

SPPU : May-05, Dec.-07, Marks 4

7. Explain TTL open collector logic.

SPPU : May-18, Marks 5

8. Explain the advantages of open collector output.

SPPU : Dec.-06,15,17, May-08, Marks 4

9. What is tri-state ? What is the use of tri-state buffers ? Explain with suitable circuit diagram.

SPPU : May-07, Marks 4

10. Draw and explain tri-state TTL inverter.

SPPU : Dec.-05, 07, May-17,19, Marks 8

11. What do you mean by tri-state buffer ?

SPPU : May-05, 07, May-19, Marks 4

12. Explain the following characteristics of TTL logic families :

- i) Power dissipation ii) Noise margin iii) Propagation delay iv) Fan out.

OR

Explain standard TTL characteristics in detail.

SPPU : Dec.-08,10,12,13,17,19, May-10,13,15,16,19, Marks 8

10.4 Complementary MOS (CMOS)

SPPU : May-05,06,07,08,11,12,13,14,15,16,17,18, Dec.-04,05,06,07,08,10,11,12,14,17,18,19

10.4.1 CMOS Inverter

- Fig. 10.4.1 shows the basic CMOS inverter circuit. It consists of two MOSFETs in series in such a way that the P-channel device has its source connected to $+V_{DD}$ (a positive voltage) and the N-channel device has its source connected to ground. The gates of the two devices are connected together as the common input and the drains are connected together as the common output.

1. When input is HIGH, the gate of Q_1 (P-channel) is at 0 V relative to the source of Q_1 i.e. $V_{gs1} = 0$ V. Thus, Q_1 is OFF. On the other hand, the gate of Q_2 (N-channel) is at $+V_{DD}$ relative to its source i.e. $V_{gs2} = +V_{DD}$. Thus, Q_2 is ON. This will produce $V_{OUT} \approx 0$ V, as shown in the Fig. 10.4.2 (a).

2. When input is LOW, the gate of Q_1 (P-channel) is at a negative potential relative to its source while Q_2 has $V_{gs} = 0$ V. Thus, Q_1 is ON and Q_2 is OFF. This produces output voltage approximately $+V_{DD}$, as shown in the Fig. 10.4.2 (b).

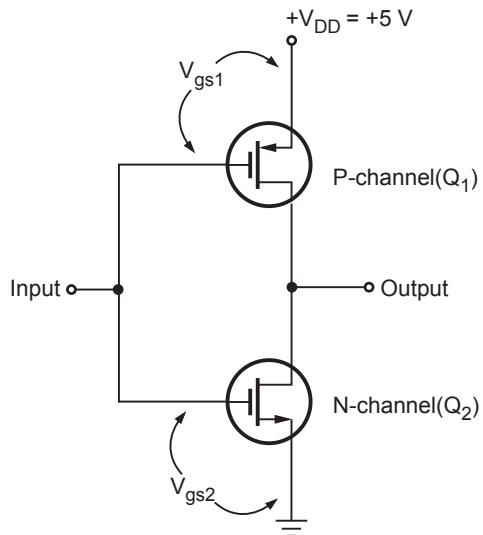


Fig. 10.4.1 CMOS inverter circuit

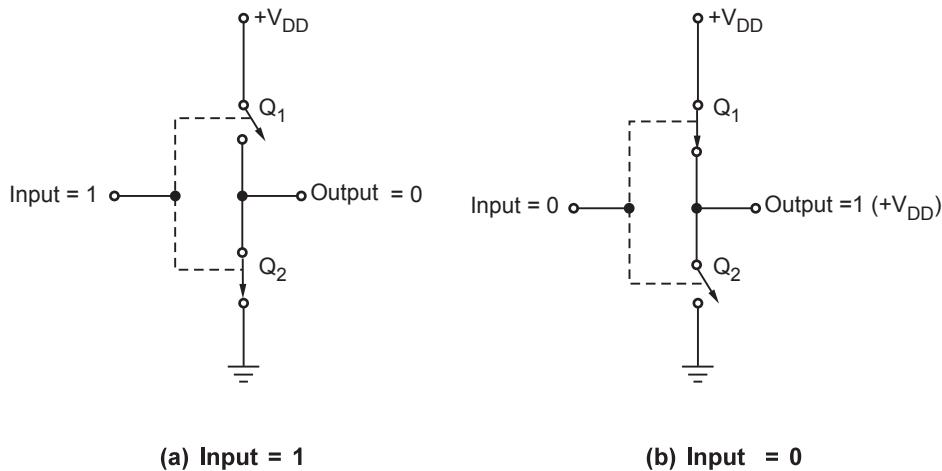


Fig. 10.4.2 Operation of CMOS inverter for both input conditions

- Table 10.4.1 summarizes the operation of CMOS inverter circuit

A	Q₁	Q₂	Output
0	ON	OFF	1
1	OFF	ON	0

Table 10.4-1 Truth table of inverter

- Fig. 10.4.3 shows, different symbols used for the p-channel and n-channel transistors to reflect their logical behaviour. The n-channel transistor (Q_2) is switched 'ON' when a HIGH voltage is applied at the input. The p-channel transistor (Q_1) has the opposite behaviour, it is switched ON when a LOW voltage is applied at the input. It is indicated by placing bubble in the symbol.

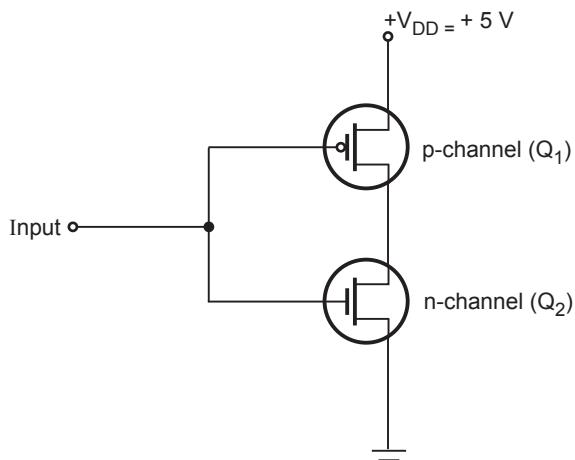


Fig. 10.4.3 The CMOS inverter

10.4.2 CMOS NAND Gate

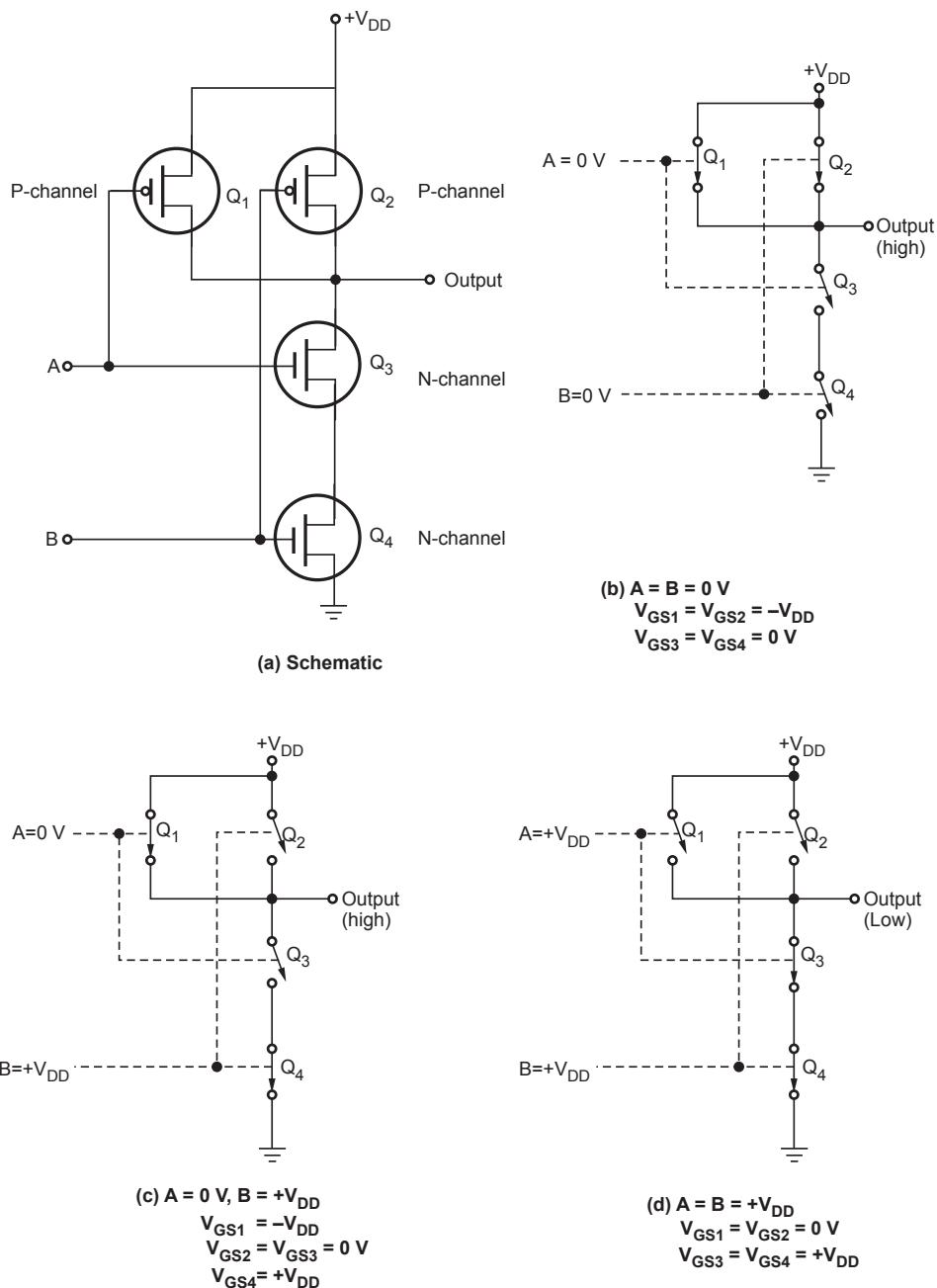


Fig. 10.4.4 CMOS NAND gate

P-channel MOSFET is ON when its gate voltage is negative with respect to its source whereas N-channel MOSFET is ON when its gate voltage is positive with respect to its source

- Fig. 10.4.4 shows CMOS 2-input NAND gate. It consists of two P-channel MOSFETs, Q_1 and Q_2 , connected in parallel and two N-channel MOSFETs, Q_3 and Q_4 connected in series.
- Fig. 10.4.4 (a) shows the equivalent switching circuit when both inputs are low. Here, the gates of both P-channel MOSFETs are negative with respect to their sources, since the sources are connected to $+V_{DD}$. Thus, Q_1 and Q_2 are both ON. Since the gate - to - source voltages of Q_3 and Q_4 (N-channel MOSFETs) are both 0 V, those MOSFETs are OFF. The output is therefore connected to $+V_{DD}$ (HIGH) through Q_1 and Q_2 and is disconnected from ground, as shown in the Fig. 10.4.4 (b).
- Fig. 10.4.4 (c) shows the equivalent switching circuit when $A = 0$ and $B = +V_{DD}$. In this case, Q_1 is on because $V_{GS1} = -V_{DD}$ and Q_4 is ON because $V_{GS4} = +V_{DD}$. MOSFETs Q_2 and Q_3 are off because their gate-to-source voltages are 0 V. Since Q_1 is ON and Q_3 is OFF, the output is connected to $+V_{DD}$ and it is disconnected from ground. When $A = +V_{DD}$ and $B = 0$ V, the situation is similar (not shown); the output is connected to $+V_{DD}$ through Q_2 and it is disconnected from ground because Q_4 is OFF.
- Finally, when both inputs are high ($A = B = +V_{DD}$), MOSFETs Q_1 and Q_2 are both OFF and Q_3 and Q_4 are both ON. Thus, the output is connected to the ground through Q_3 and Q_4 and it is disconnected from $+V_{DD}$.
- Table 10.4.2 summarizes the operation of 2-input CMOS NAND gate.

A	B	Q_1	Q_2	Q_3	Q_4	Output
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

Table 10.4.2 Truth table of NAND gate

- Fig. 10.4.5 shows the circuit diagram, function table and logic symbol of CMOS 3-input NAND gate.

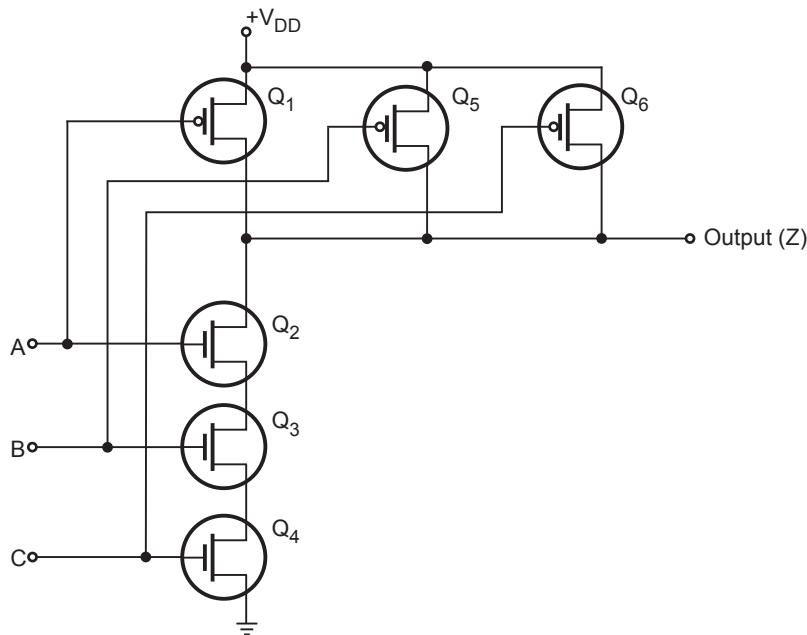


Fig. 10.4.5 (a) Circuit diagram for 3 input NAND gate

A	B	C	Q ₁	Q ₂	Q ₃	Q ₄	Q ₅	Q ₆	Z
0	0	0	ON	OFF	OFF	OFF	ON	ON	1
0	0	1	ON	OFF	OFF	ON	ON	OFF	1
0	1	0	ON	OFF	ON	OFF	OFF	ON	1
0	1	1	ON	OFF	ON	ON	OFF	OFF	1
1	0	0	OFF	ON	OFF	OFF	ON	ON	1
1	0	1	OFF	ON	OFF	ON	ON	OFF	1
1	1	0	OFF	ON	ON	OFF	OFF	ON	1
1	1	1	OFF	ON	ON	ON	OFF	OFF	0

Fig. 10.4.5 (b) Function table

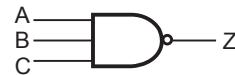
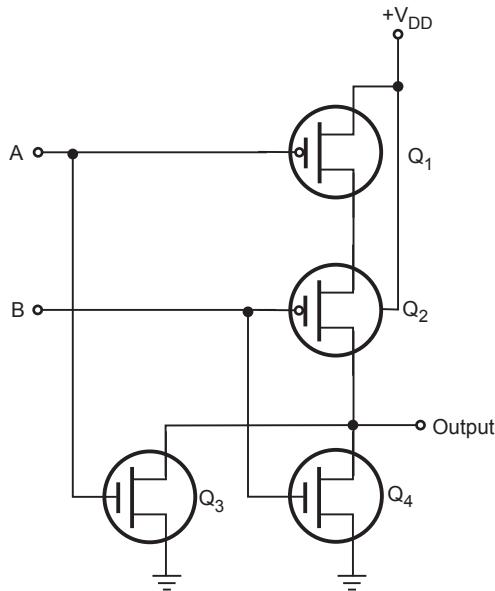


Fig. 10.4.5 (c) Logic symbol

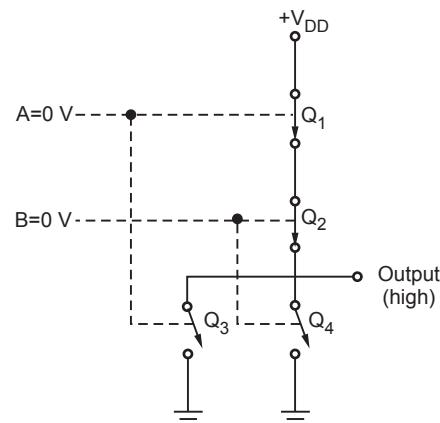
10.4.3 CMOS NOR Gate

- Fig. 10.4.6 (a) shows 2-input CMOS NOR gate. Here, P-channel MOSFETs Q₁ and Q₂ are connected in series and N-channel MOSFETs Q₃ and Q₄ are connected in parallel.

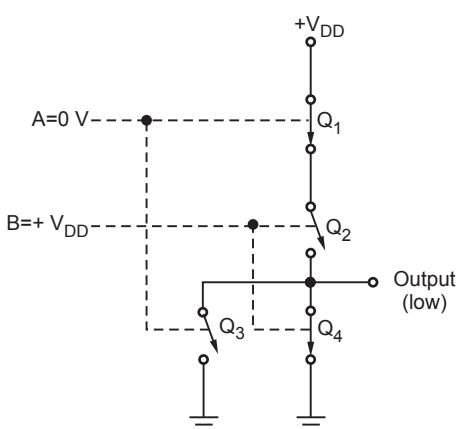
- Like NAND circuit, this circuit can be analyzed by realizing that a LOW at any input turns ON its corresponding P-channel MOSFET and turns OFF its corresponding N-channel MOSFET, and vice versa for a HIGH input. This is illustrated in Fig. 10.4.6. Table 10.4.3 summarizes the operation of 2-input NOR gate.



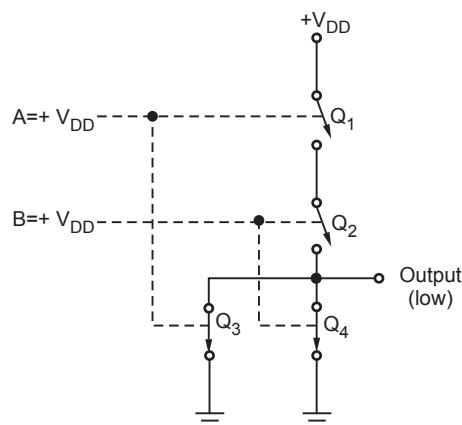
(a) Schematic



(b) $A = B = 0 \text{ V}$
 $V_{GS1} = V_{GS2} = -V_{DD}$
 $V_{GS3} = V_{GS4} = 0 \text{ V}$



(c) $A = 0 \text{ V}, B = +V_{DD}$
 $V_{GS1} = -V_{DD}$
 $V_{GS2} = V_{GS3} = 0 \text{ V}$
 $V_{GS4} = +V_{DD}$



(d) $A = B = +V_{DD}$
 $V_{GS1} = V_{GS2} = 0 \text{ V}$
 $V_{GS3} = V_{GS4} = +V_{DD}$

Fig. 10.4.6 CMOS NOR gate

A	B	Q ₁	Q ₂	Q ₃	Q ₄	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

Table 10.4.3 Truth table for NOR gate

10.4.4 CMOS Characteristics

- Operating Speed :** Slower than TTL series. Approximately 25 to 100 ns depending on the subfamily of CMOS. It also depends on the power supply voltage.
- Voltage Levels and Noise Margins :** The voltage levels for CMOS varies according to their subfamilies. These are listed in Table 10.4.4.

Parameter	CMOS series				
	4000 B	74 HC	74 HCT	74 AC	74 ACT
V _{IH(min)}	3.5	3.5	2.0	3.5	2.0
V _{IL(max)}	1.5	1.0	0.8	1.5	0.8
V _{OH(min)}	4.95	4.9	4.9	4.9	4.9
V _{OL(max)}	0.05	0.1	0.1	0.1	0.1
V _{NH}	1.45	1.4	2.9	1.4	2.9
V _{NL}	1.45	0.9	0.7	1.4	0.7

Table 10.4.4

- Noise margins in table are calculated as follows.
 $V_{NH} = V_{OH(min)} - V_{IH(min)}$
 $V_{NL} = V_{IL(max)} - V_{OL(max)}$
- Fan-out :** Typically, each CMOS load increases the driving circuit's propagation delay by 3 ns. Thus, fan-out for CMOS depends on the permissible maximum propagation delay.
- Typically, CMOS outputs are limited to a fan-out of 50.
- Power Dissipation (P_D) :** The power dissipation of a CMOS IC is very low as long as it is in a d.c. condition. Unfortunately, power dissipation of CMOS IC increases in proportion to the frequency at which the circuits are switching states. For example, a CMOS NAND gate that has P_D = 10 nW under d.c. conditions will have P_D = 0.1 mW at a frequency of 100 kHz and 1 mW at 1 MHz.

Propagation Delay : The propagation delay in CMOS is the sum of delay due to internal capacitance and due to load capacitance. The delay due to internal capacitance is called the **intrinsic propagation delay**. Typically, the propagation delay of CMOS is 70 ns.

Unused Inputs : CMOS inputs should never be left disconnected. All CMOS inputs have to be tied either to a fixed voltage level (0 V or V_{DD}) or to another input. An unused CMOS input is susceptible to noise and static charges.

10.4.5 | Wired Logic

Fig. 10.4.7 shows two CMOS inverters with their outputs connected together. This circuit does not work properly when B input is logic 0 and A input is logic 1. In this situation, Q_3 and Q_2 are ON and large current flows through Q_3 and Q_2 damaging these transistors. Therefore, wired-logic must not be used for CMOS logic circuits.

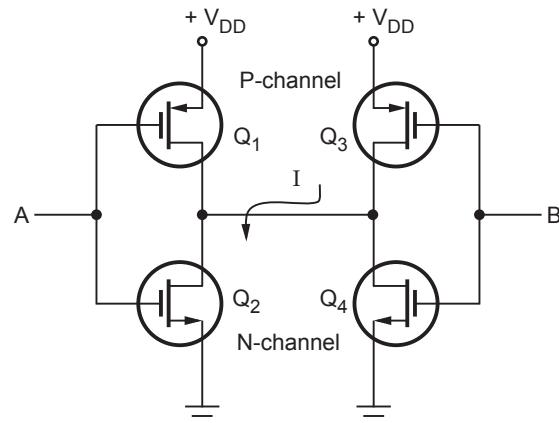


Fig. 10.4.7

10.4.6 | Open Drain Outputs

CMOS gates are available with open drain outputs, as shown in Fig. 10.4.8. In open drain outputs, PMOS transistor is replaced by a diode D_1 which provides protection from electrostatic discharge. Open drain gates can be used with external pull-up resistors to perform wired-AND operation, as discussed in TTL logic.

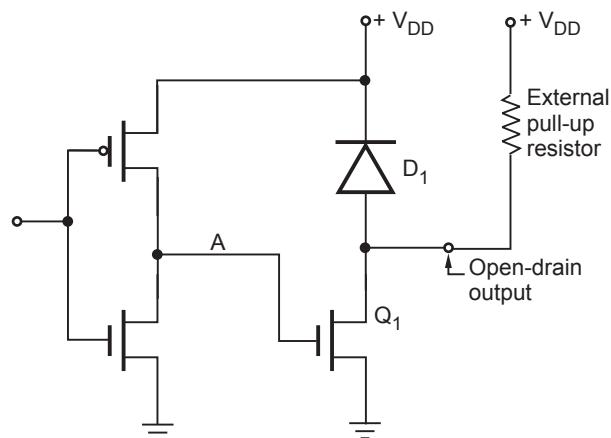


Fig. 10.4.8

10.4.7 | Advantages and Disadvantages of CMOS Family

Advantages

1. Consumes less power.
2. Can be operated at high voltages, resulting in improved noise immunity.
3. Fan-out is more.
4. Better noise margin.

Disadvantages

1. Susceptible to static charge.
2. Switching speed low.
3. Greater propagation delay.

Review Questions

1. Draw the structure of CMOS inverter gate. Explain its working.

SPPU : Dec.-07,17,19, Marks 3; May-12,17,18 Marks 4

2. Explain with neat diagram two input CMOS NAND gate.

SPPU : Dec.-12,14, May-14, Marks 8

3. Explain with neat diagram two input CMOS NOR gate.

SPPU : Dec.-10,18, May-13, Marks 4

4. Define the following parameters and give typical values of these parameters w.r.t. CMOS logic family : i) Speed of operation ii) Power dissipation

iii) Sourcing current iv) Fan out v) Sinking current.

SPPU : May-06, Marks 10

5. Explain, why wired-logic is not used for CMOS logic circuits.

SPPU : May-17, Marks 4

6. What do you mean by open drain output ? Where is it used ?

7. List differences between open drain and wired logic CMOS.

SPPU : May-12, Marks 4

8. State merits and demerits of CMOS logic family.

SPPU : Dec.-07, Marks 3

9. Explain with a neat diagram interfacing of TTL gate driving CMOS gates and vice-versa.

SPPU : May-05,07,13, Dec.-05, Marks 6; Dec.-08,11, Marks 8

10. Which parameters are significant while interfacing TTL and CMOS ? Draw and explain TTL driving CMOS gate.

SPPU : May-12, Marks 8

11. Compare TTL and CMOS logic families w.r.t. :

i) Power dissipation per gate ii) Propagation delay iii) Figure of merit iv) Fan-out.

SPPU : Dec.-04,05,06,12, May-08, Marks 6

12. List differences between CMOS and TTL.

SPPU : Dec.-11, May-15,16, Marks 4

13. Explain OR gate using CMOS logic.

SPPU : May-11, Marks 6

[Hints : Connect CMOS inverter at the output of CMOS NOR gate]

14. Why is it necessary to interface between TTL and CMOS ?

SPPU : May-13, Marks 2

10.5 Interfacing TTL and CMOS Families

SPPU : Dec.-18

- Interfacing means connecting the output(s) of one circuit or system to the input(s) of another circuit or system that may have different electrical characteristics. When two circuits have different electrical characteristics direct connection cannot be made. In such cases driver and load circuits are connected through interface

circuit. Its function is to take the driver output signal and condition it so that it is compatible with requirements of the load.

- One must consider following important points while interfacing two circuits or systems.
- The driver output must satisfy the voltage and current requirements of the load circuit.
- The driver and load circuit may require different power supplies. In such cases the output of both circuit must swing between its specified voltage ranges.

10.5.1 TTL Driving CMOS

- Here, TTL is a driver circuit and CMOS is a load circuit. The two circuits are from different families with different electrical characteristics. Therefore, we must check that the driving device can meet the current and voltage requirements of the load device.

	CMOS		TTL		
	4000B	74HC/HCT	74	74LS	74AS
I _{IH} (max)	1 μ A	1 μ A	40 μ A	20 μ A	200 μ A
I _{IL} (max)	1 μ A	1 μ A	1.6 mA	0.4 mA	2 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

Table 10.5.1. Input/output currents for standard devices with supply voltage of 5 V

- Table 10.5.1 indicates that the input current values for CMOS are extremely low compared with the output current capabilities of any TTL series. Thus, TTL has no problem meeting the CMOS input current requirements.
- But when we compare the TTL output voltages with the CMOS input voltage requirements we find that :
- $V_{OH}(\min)$ for TTL $\ll V_{IH}(\min)$ for CMOS for these situations TTL output must be raised to an acceptable level for CMOS. This can be done by connecting pull-up resistor at the output of TTL, as shown in the Fig. 10.5.1. The pull-up resistor causes the TTL output to rise to approximately 5 V in the HIGH state, thereby providing an adequate CMOS input voltage level.

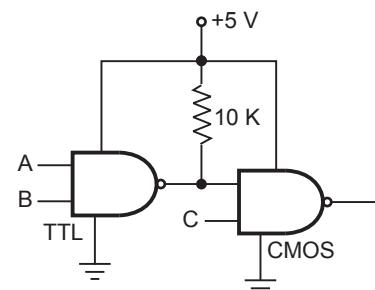


Fig. 10.5.1 TTL driving CMOS using external pull-up resistor

- TTL Driving HIGH Voltage CMOS :** When output CMOS circuit is operating with V_{DD} greater than 5 V, the situation becomes more difficult. The outputs of many TTL devices cannot be operated at more than 5 V. In such cases some alternative arrangements are made. Two of them are discussed below :
 - When the TTL output cannot be pulled up to V_{DD} , one can use open collector buffer as an interface between totem-pole TTL output and CMOS operating at $V_{DD} > 5$ V, as shown in the Fig. 10.5.2.

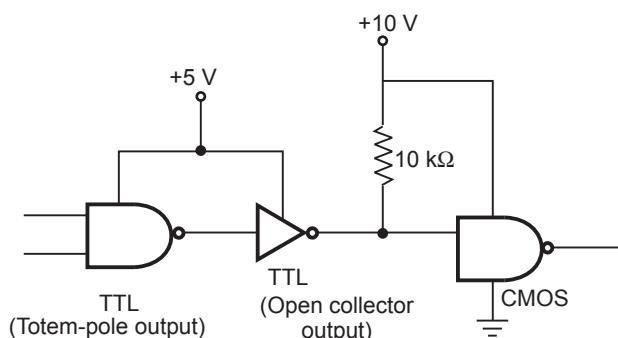


Fig. 10.5.2 Open collector buffer used as interface circuit

- The second alternative is to use **level translator** circuit, such as the 40104. This is a CMOS chip that is designed to take a low-voltage input (e.g. from TTL) and translate it to high voltage output for CMOS. Fig. 10.5.3 shows the circuit arrangement.

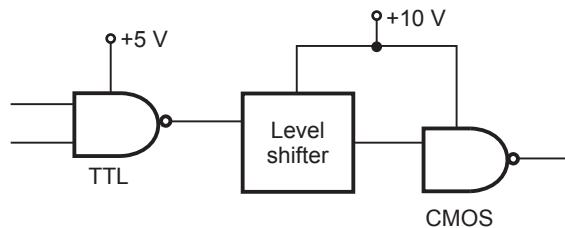


Fig. 10.5.3 Level shifter used as interface circuit

10.5.2 CMOS Driving TTL

- Before we consider the problem of interfacing CMOS outputs to TTL inputs, it will be helpful to review the CMOS output and TTL input characteristics for the two logic states.
- CMOS Driving TTL in the HIGH state :** Above voltage parameters show that CMOS outputs can easily supply enough voltage (V_{OH}) to satisfy the TTL input requirement in the HIGH state (V_{IH}). The parameters also show that CMOS outputs can supply more than enough current (I_{OH}) to meet the TTL input current requirements (I_{IH}). Thus no special consideration is required for CMOS driving TTL in the HIGH state.
- CMOS Driving TTL in the LOW state :** The parameters in the Table 10.5.2 show that CMOS output voltage (V_{OL}) satisfies TTL input requirement in the LOW state (V_{IL}). However, the current requirements in the LOW state are not satisfied. The TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state (I_{OL}) is not sufficient to drive even one input

of the TTL. In such situations some type of interface circuit is needed between the CMOS and TTL devices. Fig. 10.5.4 shows the possible interface circuit.

- In Fig. 10.5.4 the CMOS 4050B, non-inverting buffer is used as an interfacing circuit. It has an output current rating of $I_{OL(\max)} = 3 \text{ mA}$ which satisfies the TTL input current requirement.
- HIGH Voltage CMOS Driving TTL :** Some IC manufacturers have provided several 74LS TTL devices that can withstand input voltages as high as 15 V. These devices can be driven directly from CMOS outputs operating at $V_{DD} = 15 \text{ V}$. However, most TTL inputs cannot handle more than 7 V and so interface is necessary if they are to be driven from high-voltage CMOS. In such situations **voltage level translators** are used. They convert the high-voltage input to a 5 V output that can be connected to TTL.
- Fig. 10.5.5 shows how the 4050B can be used to perform this level translation between 15 V and 5 V.

For CMOS (4000B)	For TTL
$V_{OH(\min)} : 4.95 \text{ V}$	$V_{IH(\min)} : 2.0 \text{ V}$
$V_{OL(\max)} : 0.05 \text{ V}$	$V_{IL(\max)} : 0.8 \text{ V}$
$I_{OH(\max)} : 0.4 \text{ mA}$	$I_{IH(\max)} : 40 \mu\text{A}$
$I_{OL(\max)} : 0.4 \text{ mA}$	$I_{IL(\max)} : 1.6 \text{ mA}$

Table 10.5.2

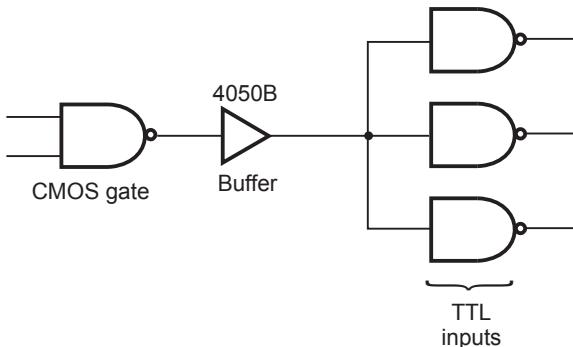


Fig. 10.5.4 CMOS driving TTL in LOW state using buffer

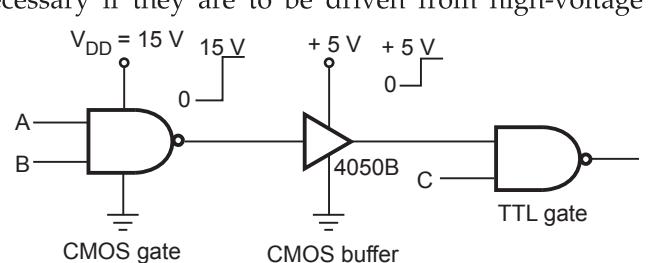


Fig. 10.5.5 Level translation using CMOS buffer

Review Questions

- Explain the interfacing of TTL and CMOS :
 - CMOS driving TTL
 - TTL driving CMOS

SPPU : Dec.-18, Marks 8

10.6 Comparison between TTL and CMOS Families

SPPU : May-15,16,19, Dec.-13,19

Sr. No.	Parameter	CMOS	TTL
1.	Device used	n-channel and p-channel MOSFET	Bipolar junction transistor
2.	$V_{IH(min)}$	3.5 V	2 V
3.	$V_{IL(max)}$	1.5 V	0.8 V
4.	$V_{OH(min)}$	4.95 V	2.7 V
5.	$V_{OL(max)}$	0.005 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	70 ns	10 ns
10.	Switching speed	Less than TTL	Faster than CMOS
11.	Power dissipation per gate	0.1 mW	10 mW
12.	Speed power product	0.7 pJ	100 pJ
13.	Fan-out	50	10
14.	Power supply voltage	3 - 15 V	Fixed 5 V
15.	Power dissipation	Increase with frequency	Increase with frequency
16.	Application	Portable instrument where battery supply is used.	Laboratory instruments.

Table 10.6.1 Comparison between TTL and CMOS families

Review Questions

1. Differentiate between standard TTL and CMOS logic circuit w.r.t.

i) Propagation delay ii) FANOUT iii) Figure of merit

SPPU : Dec.-13, Marks 6

2. List the differences between CMOS and TTL.

SPPU : May-15,16,19, Dec.-19, Marks 6



UNIT - VI

11

Introduction to Computer Architecture

Syllabus

Introduction to Ideal Microprocessor - Data Bus, Address Bus, Control Bus. Microprocessor based Systems - Basic Operation, Microprocessor operation, Block Diagram of Microprocessor.

Functional Units of Microprocessor - ALU using IC 74181, Basic Arithmetic operations using ALU IC 74181, 4-bit Multiplier circuit using ALU and shift registers. Memory Organization and Operations, digital circuit using decoder and registers for memory operations.

Contents

- 11.1 Introduction to Ideal Microprocessor
- 11.2 Data Bus, Address Bus and Control Bus
- 11.3 Microprocessor Based Systems - Basic Operation
- 11.4 Microprocessor Operation
- 11.5 Block Diagram of Microprocessor with its Functional Units
- 11.6 ALU using IC 74181 - Basic Arithmetic operations
- 11.7 The 4-bit Multiplier Circuit using ALU and Shift Registers
- 11.8 Memory Organization
- 11.9 Memory Operation

11.1 Introduction to Ideal Microprocessor

- A microprocessor is an important part of a computer architecture without which you will not be able to perform anything on your computer.
- It is a programmable device that takes in input, performs some arithmetic and logical operations over it and produces desired output.
- In simple words, a microprocessor is a digital device on a chip which can fetch instructions from memory, decode and execute them and give results.
- There is nothing like an ideal microprocessor. However, to understand the function of the microprocessor we have introduced this hypothetical device.
- Fig. 11.1.1 shows an ideal microprocessor with **n** inputs and **m** outputs.
- Input signals are applied from input devices. The input devices may include keyboard, mouse, sensors, switches, etc. These signals are in the binary form and are applied at the input terminals of the microprocessor.
- Microprocessor processes these input signals according to the sequence of operations stored in the memory. The sequence of operations referred to as program is in the binary form.
- The processed output in the binary form is fed to the output device. The output device may include indicators, displays, alarms, actuators etc.
- Microprocessor also called **Central Processing Unit (CPU)** consists of an ALU, control unit and register array.
 - The **ALU** performs arithmetic and logical operations on the data received from an input device or memory.
 - Control unit** controls the instructions and flow of data within the computer.

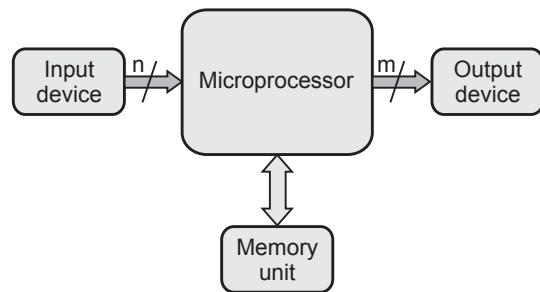


Fig. 11.1.1 An ideal microprocessor

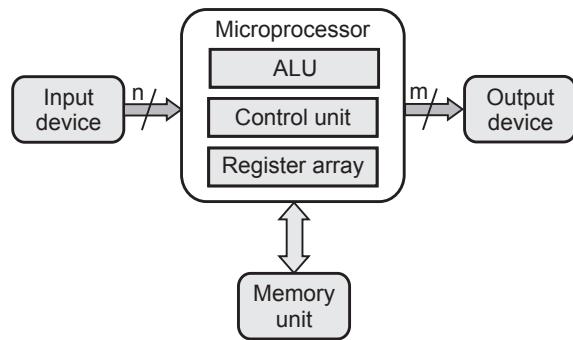


Fig. 11.1.2 Ideal microprocessor with ALU, Control unit and Register array

- **Register array** consists of registers identified by letters like B, C, D, E, H, L, and accumulator.
- Principal components of a CPU include the Arithmetic Logic Unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that orchestrates the fetching (from memory) and execution of instructions by directing the coordinated operations of the ALU, registers and other components.

Review Questions

1. What is microprocessor ?
2. Write a note on ideal microprocessor.

11.2 Data Bus, Address Bus and Control Bus

- The central processing unit, memory unit and I/O unit are the hardware components/modules of microprocessor based systems. They work together with communicating each other and have paths for connecting the modules together. The collection of paths connecting the various modules is called the **interconnection structure**.
- The design of this interconnection structure will depend on the exchanges that must be made between modules.
- A group of wires, called **bus** is used to provide necessary signals for communication between modules.
- A bus that connects major microprocessor based system components/modules (CPU, memory, I/O) is called a **system bus**. The system bus is a set of conductors that connects the CPU, memory and I/O modules. Usually, the system bus is separated into three functional groups :
 - Address bus
 - Data bus
 - Control bus

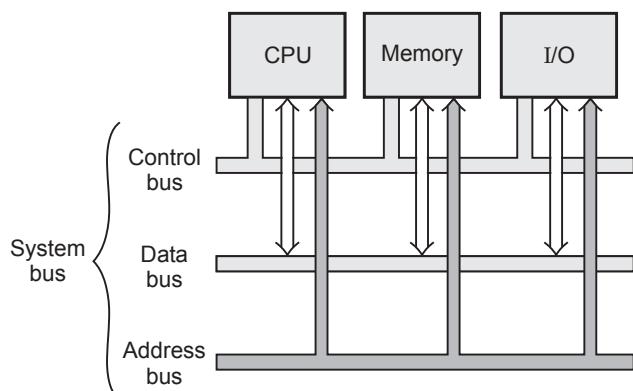


Fig. 11.2.1 Interconnection structure

11.2.1 Address Bus

- The memory which is external to the microprocessor consists of many millions of storage cells. Each cell is capable of storing 1-bit information having value 0 or 1. A single bit represents a very small amount of information. For this reason, the memory is organized so that a group of n-bits can be stored or retrieved in a single, basic operation. Each group of n bits is referred to as a **word** of information, and n is called the **word length**.

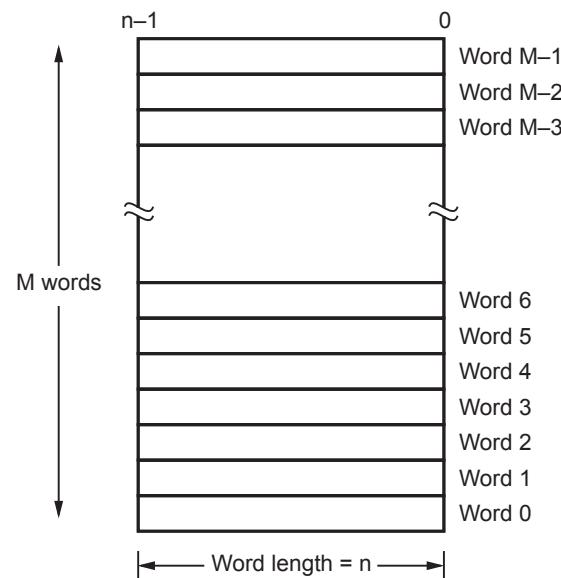


Fig. 11.2.2 Memory organization

- Fig. 11.2.2 shows the computer memory in the form of collection of words. In most of the memories word length is 8-bit, i.e., **byte**. Looking at the figure, we can say that memory is a array of M number of n-bit words.
- Each word of memory has an unique address. The address is used to identify a particular word in the memory or **memory location**. Thus the number and address lines provided by the processor decides the addressing capability of the microprocessor.
- For example, if the microprocessor has 16 address lines it can address up to 2^{16} (65536) memory locations. Table 2.3.1 shows the relation between memory capacity and the address lines.

Memory capacity	Address lines required
1 K = 1024 memory locations	10
2 K = 2048 memory locations	11
4K = 4096 memory locations	12
8K = 8192 memory locations	13
16K = 16384 memory locations	14
32K = 32768 memory locations	15
64K = 65536 memory locations	16

Table 11.2.1

- Usually the address bus of microprocessor consists of 16, 20, 24 or more parallel signal lines. On these lines the CPU sends out the address of the memory location or I/O port that is to be written to or read from.
- Here, the communication is one way, the address is send from CPU to memory and I/O port and hence these lines are **unidirectional**.

11.2.2 Data Bus

- The data bus consists of 8, 16, 32 or more parallel signal lines. These lines are used to send data to memory and output ports, and to receive data from memory and input port. Therefore, data bus lines are **bi-directional**.
- This means that CPU can read data on these lines from memory or from a port, as well as send data out on these lines to a memory location or to a port.
- The data bus is connected in parallel to all peripherals. The communication between peripheral and CPU is activated by giving output enable pulse to the peripheral. Outputs of peripherals are floated when they are not in use.
- The data transfer between the memory and microprocessor takes place through the use of two internal registers, usually called **MAR** (Memory Address Register) or simply **AR** (Address Register) and **MDR** (Memory Data Register) or simply **DR** (Data Register). This is illustrated in Fig. 11.2.3. If MAR is k -bit long and MDR is n bit long, it is possible to access up to 2^k memory locations, and during one memory cycle it is possible to transfer n -bit data.

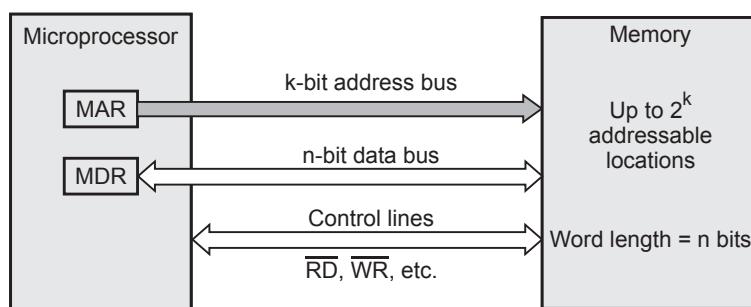


Fig. 11.2.3 Connection between memory and microprocessor

11.2.3 Control Bus

- The control lines from the microprocessor decide the memory operation. In case of read operation (**RD**) signal is activated. It is used to enable the active low output enable signal of the memory. In case of write operation (**WR**) signal is activated to indicate the write operation.

- The control lines regulate the activity on the bus. The CPU sends signals on the control bus to enable the outputs of addressed memory devices or I/O devices.
- At a time, microprocessor can communicate either with memory or I/O device. To choose between them microprocessor provides control signal called IO/M . When this signal is 1, the address meant for I/O devices. On the other hand, when this signal is 0, the address is for memory.

11.2.4 Multiplexed Address and Data Bus

- The main reason of multiplexing address and data bus is to reduce the number of pins for address and data and dedicate those pins for other several functions of microprocessor.
- These multiplexed set of lines used to carry the address as well as data at different time frames.
- In such cases, the external hardware called **latch** is used to de-multiplex address and data lines. This is illustrated in Fig. 11.2.4
- The input is transferred to the output only when clock is high. This clock signal is driven by **Address Latch Enable** (ALE) signal from microprocessor.
- During the first phase of machine cycle, address is put on the multiplexed address and data bus and the ALE signal is activated. This latches the address sent by microprocessor at the output of latch.
- In the remaining part of the machine cycle, ALE signal is disabled so output of the latch remains unchanged. During this time multiplexed address and data bus is used as a data bus.

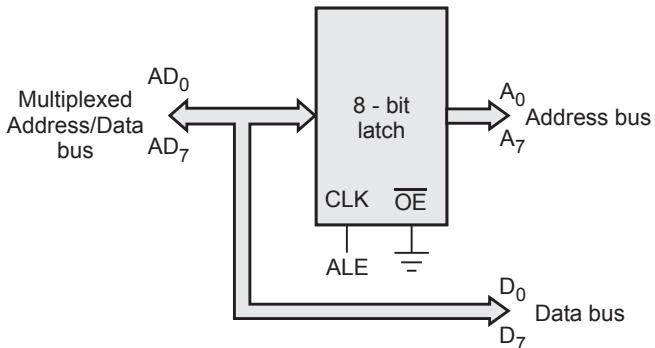


Fig. 11.2.4 De-multiplexing of address and data bus

11.2.5 Address Bus and Data Bus Drivers

- Bus drivers, buffers are used to increase the driving capacity of the microprocessor buses.

- **Unidirectional buffers** : As we know, the address bus is unidirectional, unidirectional buffer, is used to buffer address bus. Fig. 11.2.5 shows the logic diagram of 8-bit unidirectional buffer. It consists of eight non-inverting buffers with tri-state outputs. The enabling and disabling of these groups are controlled by (**Enable**) Signal.

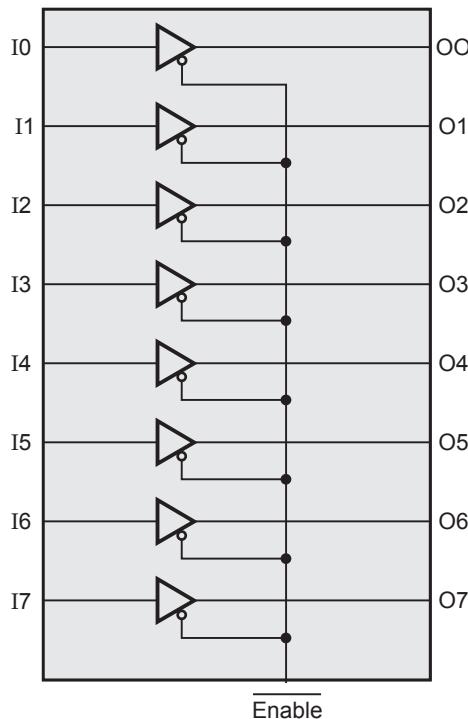


Fig. 11.2.5 8-bit unidirectional buffer

- To increase the driving capacity of data bus, bi-directional buffer is used. Fig. 11.2.6 shows the logic diagram of the 8-bit bi-directional buffer, also called an **octal bus transceivers**.
- It consists of sixteen non-inverting buffers, eight for each direction, with tri-state output. The direction of data flow is controlled by the pin DIR. When DIR is high, data flows from the A bus to the B bus; when it is low, data flows from B to A.
- For microprocessor, the number of address lines and data lines are in multiples of 8 (8, 16, 24, ...). In such case multiple number of buffer ICs are used to drive address bus and data bus.

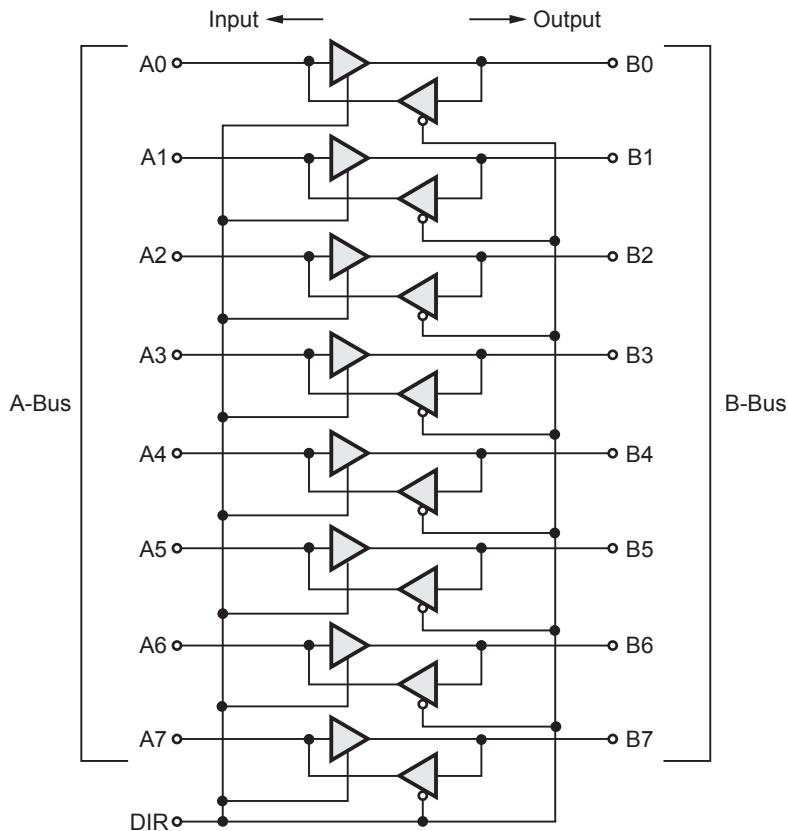


Fig. 11.2.6 8-bit bi-directional buffer

11.2.6 Sharing of Address, Data and Control Bus with DMA

- To increase the speed of data transfer between memory and I/O, the **hardware controlled I/O** is used. It is commonly referred to as Direct Memory Access (DMA). The hardware which controls this data transfer is commonly known as **DMA controller**.
- The DMA controller sends a **HOLD** signal to the microprocessor to initiate data transfer. In response to HOLD signal, microprocessor sends **HLDA** signal as an acknowledgement and releases its data, address and control buses to the DMA controller.
- Then the data transfer is controlled at high speed by the DMA controller without the intervention of the microprocessor. After data transfer, DMA controller sends low on the HOLD pin, which gives the control of data, address and control buses back to the microprocessor. This type of data transfer is used for large data transfers.

11.2.7 Control Signals to Provide Facility of Interrupt Driven I/O

- The interrupt driven I/O approach, provides the facility of interrupting the execution of the normal sequence of the program. To provide this facility, there is control input called **INTR** (Interrupt Request). When a peripheral wants to interrupt the microprocessor for some reason, it sends logic 1 on INTR control line. In response, microprocessor completes execution of the current instruction and sends out logic 0 on the **(INTA)** (Interrupt Acknowledge), output control line, acknowledging the request of the interrupting device. It then services the interrupt by transferring the program control to an interrupt service routine. This interrupt service routine performs the desire task and after completion of the task, it returns control to the main program at the point it was interrupted.

Review Questions

1. *What is bus ?*
2. *What do you mean by system bus ?*
3. *Draw the interconnection structure to connect memory and I/O devices with microprocessor.*
4. *Draw and explain the connections between memory and microprocessor.*
5. *What is the role of address bus ?*
6. *Write a short note on address bus.*
7. *Why is data bus bidirectional ?*
8. *What is the role of data bus ?*
9. *What is multiplexed address and data bus ?*
10. *How is address and data bus de-multiplexed ?*
11. *How are address and data lines de-multiplexed ?*
12. *Write a short note on address and data bus drivers.*
13. *What is DMA ?*
14. *What are the functions of HOLD and HLDA signals ?*
15. *State the control lines required to support interrupt driven I/O.*
16. *What is an interrupt driven I/O ?*

11.3 Microprocessor Based Systems - Basic Operation

- A microprocessor based system has two principle components : **hardware** and **software**. The hardware is the circuitry and physical devices, and the software is the collection of programs which controls and operates the hardware to get the desired output.
- The electronic components used in the microprocessor based system, as a whole, are referred to as **hardware**.

- The block diagram of a simple microprocessor based will give general layout of the hardware. Fig. 11.3.1 shows the block diagram of a simple microprocessor based. The major components of microprocessor based are Central Processing Unit (CPU), memory and input and output circuitry or I/O ports.
- These components of microprocessor based are connected using three sets of parallel lines called buses. The three buses are **address bus**, **data bus** and **control bus**.

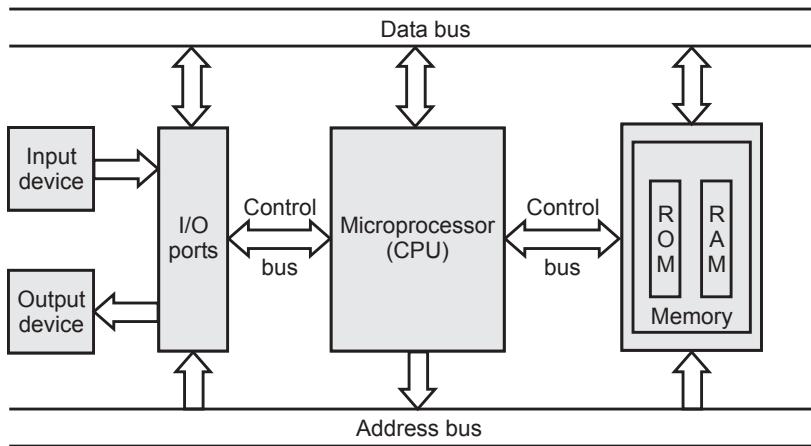


Fig. 11.3.1 Block diagram of microprocessor based system

- Central Processing Unit (CPU) :** Central Processing Unit is a microprocessor and is often referred as Microprocessor Unit (MPU). Its purpose is to fetch binary coded instructions from memory, decode the fetched instructions and generate the control signals required to execute the instructions. The CPU contains an Arithmetic Logic Unit (ALU), which can perform add, subtract, OR, AND, invert and exclusive - OR operations on binary words. The CPU also contains a program counter register which is used to hold the address of the next instruction or data to be fetched from memory, general - purpose registers which are used for temporary storage of binary data, and circuitry which generates the control signals.
- Input / Output Module :** Input/Output system consists of a variety of devices for communicating with the external world. It consists of input devices, output devices and control circuitry. Each port has a unique address. Microprocessor can read data from input device like keyboard, analog to digital converter, card readers and paper tape readers through the input port. Output ports are used to send data to output devices such as printer, video display, digital to analog converter, plotter and card punches. Physically the port is often just a set of parallel D flip flops which usually act as a buffer for input and as a latch for output.

- Memory Module :** The memory module is used to store the binary codes for the sequences of instructions we want the microprocessor to carry out. This sequence of instructions or program is stored as binary numbers in successive memory locations. It is also used to store the binary coded data.
- Typically, memory is implemented with both, ROM (Read Only Memory) and RAM ICs. RAM and ROM memories consist of an array of registers, in which each register has unique address.
- ROM :** It is a read only memory. We can't write data in this memory. It is a non volatile memory i.e. it can hold data even if power is turned off. Generally, ROM is used to store the binary codes for the sequence of instructions you want the microprocessor to carry out and data such as lookup tables. This is because this type of information does not change.
- RAM :** Unlike ROM, we can read from or write into the RAM, so it is often called read/write memory. The numerical and character data that are to be processed by the microprocessor change frequently. These data must be stored in type of memory from which they can be read by the microprocessor, modified through processing, and written back for storage. For this reason, they are stored in RAM instead of ROM.
- Program is a series of instructions that can be executed in order to perform a **specified** task.
- Let us consider a simple task to understand the functioning of the components of a microprocessor. The task involves following three operations :
 1. Read a value from a keyboard connected to the port at address 01H.
 2. Add 30H to the value read from keyboard.
 3. Send the result to a display connected to the port at address 02H.

Table 11.3.1 shows a program to accomplish the given task. It consists of three instructions.

Memory Address		Contents (binary)										Contents (Hex)	Operation	
0	0	0	0	H	1	1	0	1	1	0	1	1	DB	Input from port 01H
0	0	0	1	H	0	0	0	0	0	0	0	1	01	
0	0	0	2	H	1	1	0	0	0	1	1	0	C6	Add immediate data 30h
0	0	0	3	H	0	0	1	1	0	0	0	0	30H	
0	0	0	4	H	1	1	0	1	0	0	1	1	D3	Output to port 02H
0	0	0	5	H	0	0	0	0	0	0	1	0	02	

Table 11.3.1 Memory addresses and memory contents for a three instruction program

- We assume that the microprocessor fetches instructions and data from memory 1 byte at a time.
- An **opcode** is the first byte of an instruction in machine language which tells the hardware what operation needs to be performed with this instruction. Every microprocessor has its own set of opcodes (operational codes) defined in its architecture.
- We also assume that the instructions are in sequential memory locations starting at address 0000H as shown in Table 11.3.1.
- Fig. 11.3.2 shows the hardware components and Table 11.3.2 shows the sequence of actions that the microprocessor will perform to execute these three instructions.

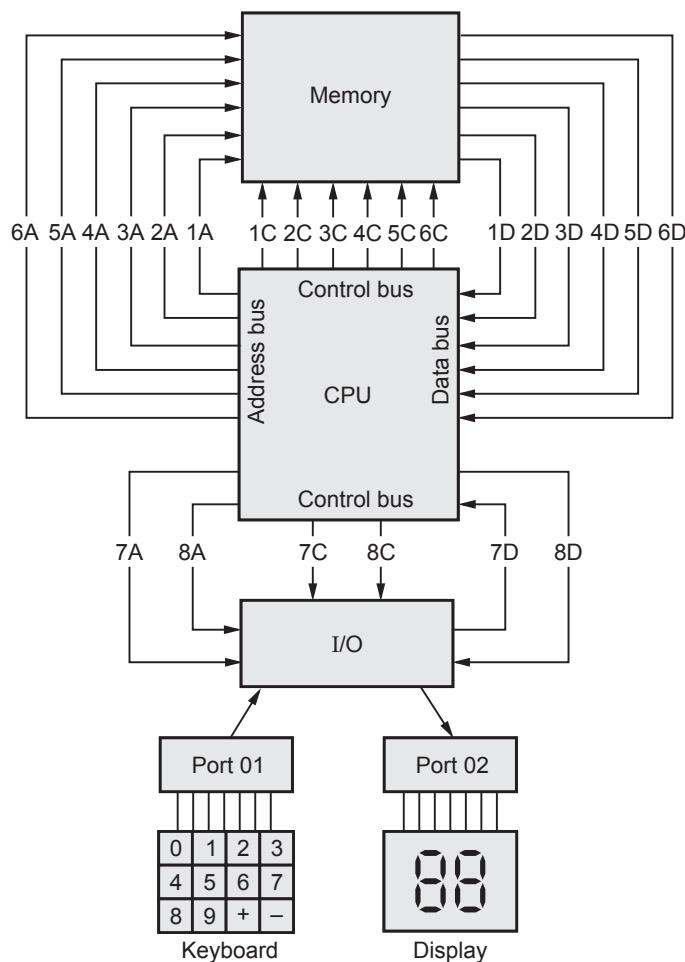


Fig. 11.3.2 Hardware components for execution of three-instruction program

Instruction		Action
1	1A	Microprocessor sends the address of the first instruction to the memory. Address Bus : 0000H
	1C	Microprocessor sends memory read control signal to enable memory.
	1D	Microprocessor reads first instruction byte (DBH) sent from memory on data bus and decodes it. By decoding it determines that the code read represents the INPUT instruction and it needs more information to carry out the instruction.
	2A	Microprocessor sends the address of the next memory location to get the remaining instruction. Address Bus : 0001H
	2C	Microprocessor sends memory read control signal to enable memory.
	2D	Microprocessor reads port address byte (01H) sent from memory on data bus
	7A	To execute instruction Microprocessor sends the port address (01H) on the address bus.
	7C	Microprocessor sends I/O read control signal to enable the addressed input port.
	7D	Input device puts a byte data on the data bus. Microprocessor reads the data byte from the data bus and stores it in the internal register. This completes execution of the first instruction.
2	3A	Microprocessor sends the address of the next (second) instruction (0002H) on the address bus.
	3C	Microprocessor sends memory read control signal to enable memory.
	3D	Microprocessor reads instruction byte (C6H) sent from memory on data bus and decodes it. By decoding it determines that it supposed to add some number to the number stored in the internal register. It also determines that it must get next byte of instruction from memory.
	4A	Microprocessor sends next sequential address (0003H) on the address bus.
	4C	Microprocessor sends memory read control signal to enable memory.
	4D	Microprocessor reads the data byte (30H) from the data bus and adds it to the contents of internal register. This completes execution of the second instruction.

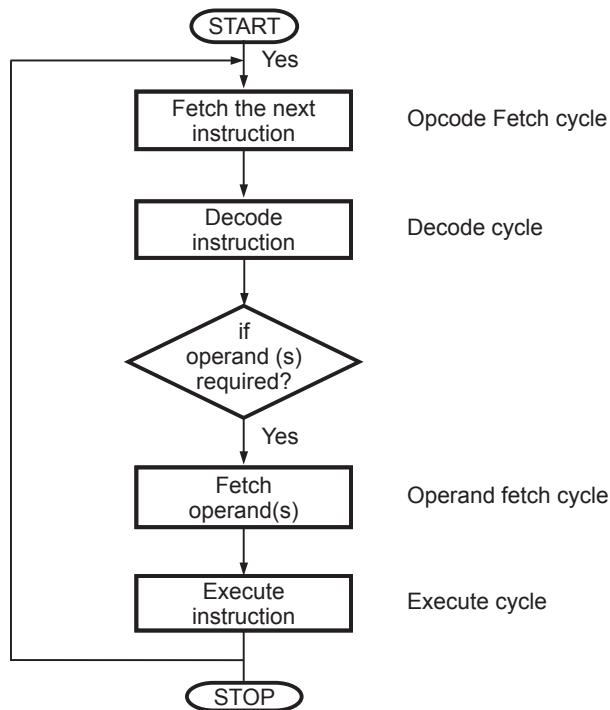
3.	5A	Microprocessor sends the address of the next (third) instruction (00004H) on the address bus.
	5C	Microprocessor sends memory read control signal to enable memory.
	5D	Microprocessor reads instruction byte (D3H) sent from memory on data bus and decodes it. By decoding it determines that it is supposed to perform output operation. It also determines that it must get address of the output port from memory.
	6A	Microprocessor sends next sequential address (0005H) on the address bus.
	6C	Microprocessor sends memory read control signal to enable memory.
	6D	Microprocessor reads output port address byte (02H) sent from memory on data bus.
	8A	Microprocessor sends the output port address (02H) on the address bus.
	8D	Microprocessor sends the data byte from the internal register on the data bus.
	8C	Microprocessor sends I/O write signal on the control bus to enable the addressed output port so that the data from the data bus can pass through it to LED displays. This completes the execution of the third instruction.

Table 11.3.2 Sequence of actions during execution of three instruction program**Review Questions**

1. Draw and explain the block diagram of microprocessor based system.
2. Write a note on memory module of a microprocessor based system.
3. Explain the execution of three-instruction program.

11.4 Microprocessor Operation

- The primary function of a microprocessor is to execute sequence of instructions stored in a memory, which is external to the microprocessor. The sequence of operations involved in processing an instruction constitutes an instruction cycle.
- From the above discussion it is cleared that, the complete instruction cycle involves three operations : **fetch, decode and execution.**
(See Fig. 11.4.1 on next page)
- **Fetch :**
 - Microprocessor sends the address of the instruction to the memory.
 - Microprocessor also sends memory read control signal to enable memory.
 - Microprocessor reads instruction byte (opcode) sent from memory on data bus and places it in the Instruction Register (IR) in the microprocessor.

**Fig. 11.4.1 Basic instruction cycle**

- When the fetch cycle is used to read instruction it is known as **instruction fetch cycle**. On the other hand, when the fetch cycle is used to read operand it is known as **operand fetch cycle**.
- **Decode :**
 - Microprocessor decodes the opcode of the instruction stored in the instruction register to determine which operation is to be performed.
- **Execution :**
 - Microprocessor performs the specified operation. This often involves performing an arithmetic or logical operation and storing the result in the destination location.

Review Questions

1. Write a note on microprocessor operations
2. Explain the phases in the instruction cycle.

11.5 Block Diagram of Microprocessor with its Functional Units

- Fig. 11.5.1 shows the simplified block diagram of microprocessor. As shown in Fig. 11.5.1 includes three major logic devices
 - ALU
 - Several registers
 - Control unit
- The internal data bus is used to transmit data between these logic devices.

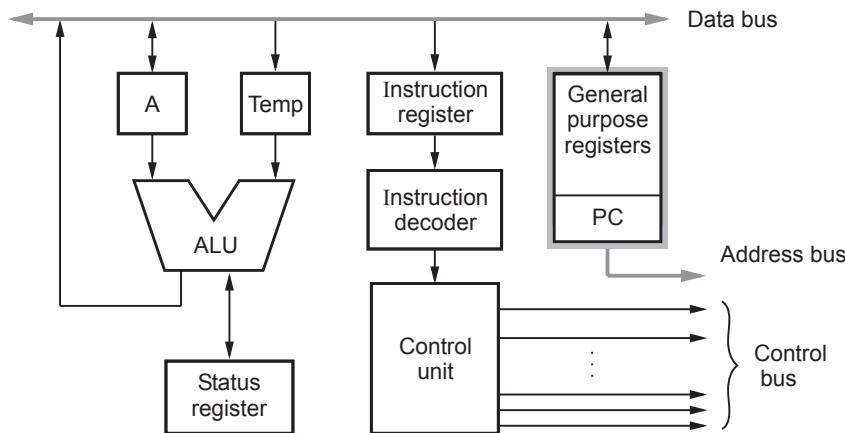


Fig. 11.5.1 Simplified block diagram of a microprocessor

11.5.1 Arithmetic and Logic Unit (ALU)

- One of the microprocessor's major logic devices is the arithmetic logic unit (ALU). It contains the microprocessor's data processing logic. It has two inputs and an output.
- The internal data bus of microprocessor is connected to the two inputs of ALU through the temporary register and the accumulator.
- The ALU's single output is connected to the internal data bus so that can be sent to any device connected to the bus. In most of the microprocessors **register A** gives data for the ALU and after performing the operation, the resulting data word is sent to the register A and stored there. This special register, where the result is accumulated is commonly known as **accumulator**.
- The ALU works on either one or two data words, depending on the kind of operation. The ALU uses input ports as necessary. For example, addition operation uses both ALU inputs while complementing data operation uses only one input.

To complement the data word, all the bits of the word's that are logic 1 are set to logic 0, and all the bits of the word at logic 0 are set to logic 1.

Table 11.5.1 lists some of the functions performed by the ALU in most of the microprocessors.

Add	Subtract	AND	OR	Exclusive OR
Complement	Shift right	Shift left	Increment	Decrement

Table 11.5.1 Functions performed by ALU

11.5.2 Registers

- Registers are a prominent part of the block diagram and the programming model of any microprocessor. The basic registers found in most of the microprocessors are :
 - Accumulator
 - Program Counter (PC)
 - Status Register (Flag Register)
 - General purpose registers
 - Memory address register
 - Instruction register and
 - Temporary data registers.
 - Stack Pointer (SP)

Accumulator

- The accumulator is the major working register of microprocessor. Most of the time it is used to hold the data for manipulation.
- Whenever the operation processes two words, whether arithmetically or logically, the accumulator contains one of the words. The other word may be present in another register or in a memory location.
- Most of the times the result of an arithmetic or logical operation is placed in the accumulator. In such cases, after execution of instruction original contents of accumulator are lost because they are overwritten.
- The accumulator is also used for data transfer between an I/O port and a memory location, or between one memory location and another.

Program Counter (PC)

- The Program Counter is one of the most important registers in the microprocessor. As mentioned earlier, a program is a series of instructions stored in the memory. These instructions tell the microprocessor exactly how to solve a problem. It is

important that these instructions must be executed in a proper order to get the correct result. This sequence of instruction execution is monitored by the program counter. It keeps track of which instruction is being used and what the next instruction will be.

- The program counter gives the address of memory location from where the next instruction is to be fetched. Due to this the length of the program counter decides the maximum program length in bytes. For example, microprocessor that has 16 bit program counter, can address bytes (64 K) of memory.
- Before the microprocessor can start executing a program, the program counter has to be loaded with valid memory address. This memory location must contain the opcode of first instruction in the program. In most of the microprocessors this location is fixed. For example, memory address (0000H) for 16 bit program counter. The fixed address is loaded into the program counter by resetting the microprocessor.

Status Register

- The status register is used to store the results of certain condition when certain operations are performed during execution of the program.
- The status register is also referred to as **flag register**. ALU operations and certain register operations may set or reset one or more bits in the status register.
- Status bits lead to a new set of microprocessor instructions. These instructions permit the execution of a program to change flow on the basis of the condition of bits in the status register. So the condition bits in the status register can be used to take logical decisions within the program. Some of the common status register bits are :
- **Carry/Borrow** : The carry bit is set when the summation of two 8 bit numbers is greater than 1111 1111 (FFH). A borrow is generated when a large number is subtracted from a smaller number.
- **Zero** : The zero bit is set when the contents of register are zero after any operation. This happens not only when you decrement the register, but also when any arithmetic or logical operation causes the contents of register to be zero.
- **Negative or sign** : In 2's complement arithmetic, the most significant bit is a sign bit. If this bit is logic 1, the number is negative number, otherwise a positive number. The negative bit or sign bit is set when any arithmetic or logical operation gives a negative result.
- **Auxiliary Carry** : The auxiliary carry bit of status register is set when an addition in the first 4 bits causes a carry into the fifth bit. This is often referred as half carry or intermediate carry. This is used in the BCD arithmetic.

- **Overflow Flag :** In 2's complement arithmetic, most significant bit is used to represent sign and remaining bits are used to represent magnitude of a number. This flag is set if the result of a signed operation is too large to fit in the number of bits available (7-bits for 8-bit number) to represent it.
- For example, if you add the 8-bit signed number 01110110 (+118 decimal) and the 8-bit signed number 00110110 (+ 54 decimal). The result will be 10101100 (+ 172 decimal), which is the correct binary result, but in this case it is too large to fit in the 7-bits allowed for the magnitude in an 8-bit signed number. The overflow flag will be set after this operation to indicate that the result of the addition has overflowed into the sign bit.
- **Parity :** When the result of an operation leave the indicated register with an even number of 1s, parity bit is set.

General Purpose Registers

- In addition to the basic registers, most microprocessors have other registers called **general purpose registers**.
- The general purpose registers are used as simple storage area, mainly these are used to store intermediate results of the operation. Getting the operand from the general purpose registers is more faster than from memory so it is better to have sufficient number of general purpose register in the microprocessor.
- The microprocessor used in this chapter has six general purpose registers (Refer Fig. 11.5.1) called the B, C, D, E, H, and L registers. These registers individually can operate as 8 bit registers. Together, the BC, DE, and HL registers can operate as 16 bit register pairs.

Memory Address Register

- The memory address register gives the address of memory location that the microprocessor wants to use. That is, memory address register holds 16-bit binary number. The output of the memory address register drives the 16-bit address bus. This output is used to select a memory location.

Instruction Register (IR)

- The instruction register holds the operation code (opcode) of the instruction the microprocessor is currently executing. The instruction register is loaded during the opcode fetch cycle. The contents of the instruction register are used to drive the part of the control logic known as the **instruction decoder**.

Temporary Data Register

- The need for the temporary data registers arises because the ALU has no storage of its own. The ALU has two inputs. One input is supplied by accumulator and

other from temporary data register. The programmer cannot access this temporary data register and, therefore it is not a part of programming model.

Stack Pointer

- The **stack** is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data. The Stack Pointer register holds the address of the top location of the stack.

11.5.3 Instruction Decoder

- It decodes the opcode from instruction register and generates the decoded output. This decoded output is used by control logic to generate control signals.

11.5.4 Control Logic

- The control logic is a important block in the microprocessor. The control logic is responsible for working of all other parts of the microprocessor together.
- It maintains the synchronization in operation of different parts in the microprocessor. The synchronization is achieved with the help of one of the control logic's major external inputs, microprocessor's clock. The clock is a signal which is the basis of all the timings inside the microprocessor.
- The control logic receives the signal from instruction decoder and generates the control signals necessary to execute the instruction.
- The control logic does a few other special functions. It looks after the microprocessor power-up sequence. It also processes interrupts.

11.5.5 Internal Data Bus

- The internal data bus connects the different parts of microprocessor together and it enables the communication between these parts. The data transfer through this internal data bus is controlled by control logic.
- Microprocessor's internal data bus usually connected to an external data bus. Due to this microprocessor can communicate with external memory or I/O devices. Usually the internal data bus is connected to the external data bus by logic called a **bi-directional bus** (transceiver).

11.5.6 Subroutine, Stack and Stack Pointer

- As said earlier, the instructions must be executed in a proper order to get the correct result. This does not mean that every instruction must follow the last instruction in the memory. But it must follow the logical sequence of the instructions.

- In some situations, it is better to execute part of a program that is not in sequence (don't confuse with logical sequence) with the main program.
- For example, there may be a part of a program that must be repeated many times during the execution of the entire program. Rather than writing repeated part of the program again and again, the programmer can write that part only once. This part is written separately. The part of the program which is written separately is called **subroutine**.
- Fig. 11.5.2 shows how the main and subroutine programs are executed.
- The program counter does the major role in subroutine execution as it can be loaded with required memory address.
- With the help of instruction it is possible to load any memory address in the program counter. When subroutine is to be executed, the program counter is loaded with the memory address of the first instruction in the subroutine.
- After execution of the subroutine, the program counter is loaded with the memory address of the next instruction from where the program control was transferred to the subroutine program.
- During subroutine operation, before transferring the program control to the subroutine the return address is kept in a special memory area called the **stack**.
- After the execution of subroutine the return address is popped off from the stack and loaded into the program counter.
- The memory address of the stack area is given by a special register called the **stack pointer**.
- Like the Program Counter, the Stack Pointer automatically points to the next available location in the memory. In most microprocessors, the stack pointer decrements (points to the next lower memory address) when data is pushed on the stack. This allows the programmer to build the stack down in memory as shown in the Fig. 11.5.3.

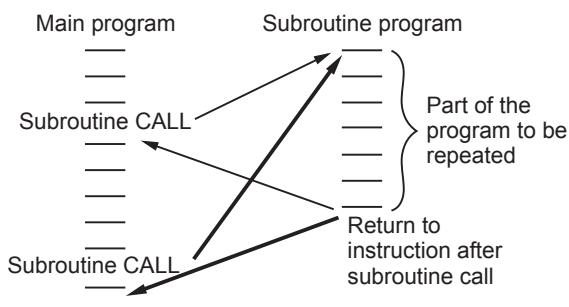


Fig. 11.5.2 Execution of subroutine programs

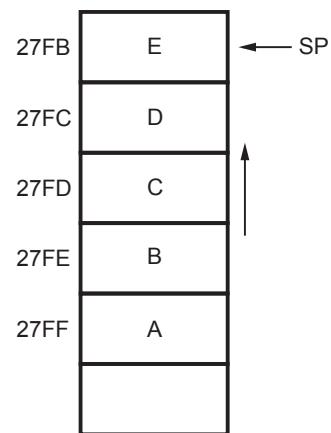


Fig. 11.5.3 Stack operation

- It is important to note that as you go on storing (pushing) data on the stack, the stack pointer always points the last data placed on the stack and when you try to remove (pop) data you always get the last data placed on the stack. This kind of stack operation is called **LIFO (Last In First Out) operation**.

Review Questions

- Draw and explain the block diagram of microprocessor.*
- Explain various functions of ALU.*
- What is accumulator ?*
- What is program counter ?*
- What do you mean by general purpose registers ?*
- List various flags in the status register.*
- What is status register ? Explain its use*
- State the functions of instruction register and instruction decoder.*
- What is subroutine ?*
- Explain the execution of subroutine program.*
- What is stack and stack pointer ?*
- Explain the operation of stack.*

11.6 ALU using IC 74181 - Basic Arithmetic operations

- In this section we study the very popular ALU IC, IC 74LS181. It is a 4-bit Arithmetic Logic Unit (ALU). Its features are as given below :
- Features :**
 - Provides 16 arithmetic operations : add, subtract, compare, double, plus twelve other arithmetic operations.
 - Provides all 16 logic operations of two variables : exclusive-OR, compare, AND, NAND, OR, NOR, plus ten other logic operations.
 - Full lookahead for high speed arithmetic operation on long words.
- Fig. 11.6.1 and Fig. 11.6.2 show the block diagram and connection diagram for IC74LS181.
- As shown in the Fig. 11.6.1, 74LS181 has two four bit operands ($A_0 - A_3$ and $B_0 - B_3$). Its mode select input selects one of the two modes : arithmetic or logic, and four function select inputs select a particular function from the selected mode.

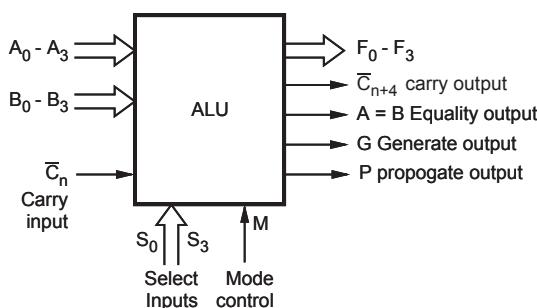


Fig. 11.6.1 Block diagram

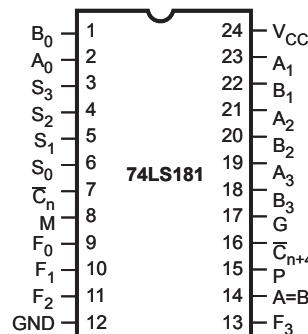


Fig. 11.6.2 Connection diagram

- Table 11.6.1 gives the pin description for IC 74LS181 and Table 11.6.2 gives the function table for IC 74LS181.

Pin Names	Description
$A_0 - A_3$	Operand Inputs
$B_0 - B_3$	Operand Inputs
$S_0 - S_3$	Function Select Inputs
M	Mode Control Input
\bar{C}_n	Carry Input (Active LOW)
$F_0 - F_3$	Function Outputs
$A = B$	Comparator Output
G	Carry Generate Output
P	Carry Propagate Output
\bar{C}_{n+4}	Carry Output (Active LOW)

Table 11.6.1 Pin description of IC 74LS181

Mode Select Inputs				Active HIGH Operands and F_n Outputs	
S_3	S_2	S_1	S_0	Logic ($M = 1$)	Arithmetic (Note 2) ($M = 0$) ($\bar{C}_n = 1$)
0	0	0	0	$F = \bar{A}$	$F = A$
0	0	0	1	$F = \bar{A} + \bar{B}$	$F = A + B$
0	0	1	0	$F = \bar{A}B$	$F = A + \bar{B}$

0	0	1	1	F = Logic 0	F = minus 1
0	1	0	0	F = \overline{AB}	F = A plus \overline{AB}
0	1	0	1	F = \overline{B}	F = (A + B) plus \overline{AB}
0	1	1	0	F = A \oplus B	F = A minus B minus 1
0	1	1	1	F = \overline{AB}	F = AB minus 1
1	0	0	0	F = $\overline{A} + B$	F = A plus AB
1	0	0	1	F = $\overline{A} \oplus \overline{B}$	F = A plus B
1	0	1	0	F = B	F = (A + \overline{B}) plus AB
1	0	1	1	F = AB	F = AB minus 1
1	1	0	0	F = Logic 1	F = A plus A (Note 1)
1	1	0	1	F = A + \overline{B}	F = (A + B) plus A
1	1	1	0	F = A + B	F = (A + \overline{B}) plus A
1	1	1	1	F = A	F = A minus 1

Table 11.6.2 (a) Function table for IC 74LS181

Note 1 : Each bit is shifted to the next most significant position.

Note 2 : Arithmetic operations expressed in 2's complement notation.

Mode Select Inputs				Active LOW Operands and F_n Outputs	
				Logic (M = 1)	Arithmetic (Note 2) (M = 0) ($\overline{C}_n = 0$)
S_3	S_2	S_1	S_0		
0	0	0	0	F = \overline{A}	F = A minus 1
0	0	0	1	F = \overline{AB}	F = AB minus 1
0	0	1	0	F = $\overline{A} + \overline{B}$	F = $A\overline{B}$ minus 1
0	0	1	1	F = Logic 1	F = minus 1
0	1	0	0	F = $\overline{A} + \overline{B}$	F = A plus ($A + \overline{B}$)
0	1	0	1	F = \overline{B}	F = AB plus ($A + \overline{B}$)
0	1	1	0	F = $\overline{A} \oplus \overline{B}$	F = A minus B minus 1
0	1	1	1	F = $A + \overline{B}$	F = $A + \overline{B}$
1	0	0	0	F = $\overline{A} B$	F = A plus (A + B)
1	0	0	1	F = A \oplus B	F = A plus B

1	0	1	0	$F = B$	$F = \bar{AB}$ plus $(A + B)$
1	0	1	1	$F = A + B$	$F = A + B$
1	1	0	0	$F = \text{Logic 0}$	$F = A$ plus A (Note 1)
1	1	0	1	$F = \bar{AB}$	$F = AB$ plus A
1	1	1	0	$F = AB$	$F = \bar{AB}$ minus A
1	1	1	1	$F = A$	$F = A$

Table 11.6.2 (b) Function table for IC 74LS181

Note 1 : Each bit is shifted to the next most significant position.

Note 2 : Arithmetic operations expressed in 2's complement notation.

Functional Description

- The 74LS181 is a 4-bit high speed parallel Arithmetic Logic Unit (ALU), controlled by the four Function Select inputs (S_0-S_3) and the Mode Control input (M), it can perform all the 16 possible logic operations or 16 different arithmetic operations on active HIGH or active LOW operands.
- When the Mode Control input (M) is HIGH, all internal carries are inhibited and the device performs logic operations on the individual bits as listed.
- When the Mode Control input is LOW, the carries are enabled and the device performs arithmetic operations on the two 4-bit words.
- The device incorporates full internal carry lookahead and provides for either ripple carry between devices using the $\bar{C}_n + 4$ output, or for carry lookahead between packages using the signals P (Carry Propagate) and G (Carry Generate).
- In the ADD mode, P indicates that F is 15 or more, while G indicates that F is 16 or more.
- In the SUBTRACT mode, P indicates that F is zero or less, while G indicates that F is less than zero. P and G are not affected by carry in. When speed requirements are not stringent, it can be used in a simple ripple carry mode by connecting the Carry output (\bar{C}_{n+4}) signal to the Carry input (C_n) of the next unit.
- The $A = B$ output from the device goes HIGH when all four F outputs are HIGH and can be used to indicate logic equivalence over four bits when the unit is in the subtract mode. The $A = B$ output is open-collector and can be wired - AND with other $A = B$ out puts to give a comparison for more than four bits. The $A = B$ signal can also be used with the \bar{C}_{n+4} signal to indicate $A > B$ and $A < B$.
- The Function Table lists the arithmetic operations that are performed without a carry in. An incoming carry adds one to each operation. Thus, select code 0110

generates A minus B minus 1 (2's complement notation) without a carry in and generates A minus B when a carry is applied.

- Because subtraction is actually performed by complementary addition (1's complement), a carry out means borrow; thus a carry is generated when there is no underflow and no carry is generated when there is underflow.
- As indicated, this device can be used with either active LOW inputs producing active LOW outputs or with active HIGH inputs producing active HIGH outputs. For either case the table lists the operations that are performed to the operands labeled inside the logic symbol.

Example 11.6.1 Design 8 bit ALU circuit using two 74LS181 ICs.

Solution : Fig. 11.6.3 shows the cascaded connection of two 74LS181 ICs.

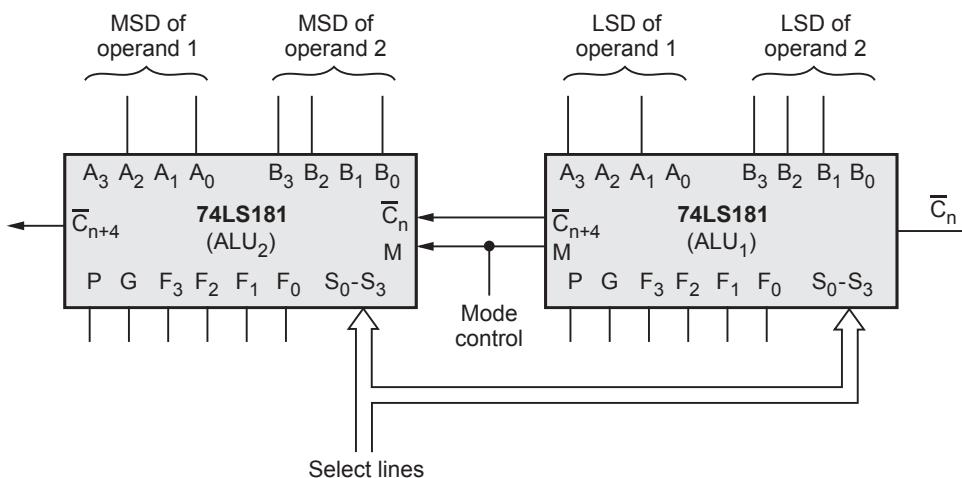


Fig. 11.6.3 8-bit ALU using two 74LS181 ICs

- As shown in the Fig. 11.6.3 mode and select inputs for both ICs are connected in parallel so that they operate in same mode and perform the same function. For ALU₁, carry out (C_{out}) is connected to C_n input of the ALU₂, i.e. carry of one stage is propagated to next stage. The C_{n+4} output of ALU₂ gives the final carry.

Example 11.6.2 Show how the circuit designed in the previous example works for following operations.

1. A - B
 2. A + B
 3. A XOR B
 4. A AND B
- Assume : A = 56₁₀ = 0011 1000₂ and B = 45₁₀ = 0010 1100₂

Solution : A = 56₁₀ = 0011 1000₂ and B = 45₁₀ = 0010 1100₂

Subtraction

0	0	1	0	1	1	0	0
1	1	0	1	0	0	1	1
+							1
1	1	0	1	0	1	0	0

B**1's complement of B****Add 1****2's complement of B****Addition**

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

Carry**A****B****Result**

1	1	1					
0	0	1	1	1	0	0	0
+	1	1	0	1	0	1	0

Carry**A****2's complement of B**

Ignore carry **Result**
0 0 0 0 1 1 0 0

Operation	Mode Control	Select Inputs	Most Significant ALU (ALU ₂)				Least Significant ALU (ALU ₁)				Output			
			Inputs		Outputs		Inputs		Outputs					
	M	S ₃ S ₂ S ₁ S ₀	A	B	\bar{C}_n	S	\bar{C}_{n+4}	A	B	\bar{C}_n	S	\bar{C}_{n+4}	Binary	Decimal
1. A - B	0	0 1 1 0	0011	0010	1	0000	0	1000	1100	0	1100	1	0000 1100	12
2. A + B	0	1 0 0 1	0011	0010	0	0110	1	1000	1100	1	0100	0	0110 0100	100

XOR operation

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

A**B****Result****AND operation**

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

A**B****Result**

Operation	Mode Control	Select Inputs	Most Significant ALU (ALU ₂)				Least Significant ALU (ALU ₁)				Output	
			Inputs		Outputs		Inputs		Outputs			
	M	S ₃ S ₂ S ₁ S ₀	A	B	S	A	B	S	Binary	Decimal		
3. A XOR B	1	0 1 1 0	0011	0010	0001	1000	1100	0100	0001 0100	10		
4. A AND B	0	1 0 1 1	0011	0010	0010	1000	1100	1000	0010 1000	100		

Review Questions

- Draw the block diagram of ALU IC 74181 and explain the function of all important pins.
- Write a short note on IC 74181.
- Explain any four arithmetic and four logic functions of ALU IC 74181.
- Implement an 8-bit ALU using two 74181 ICs.

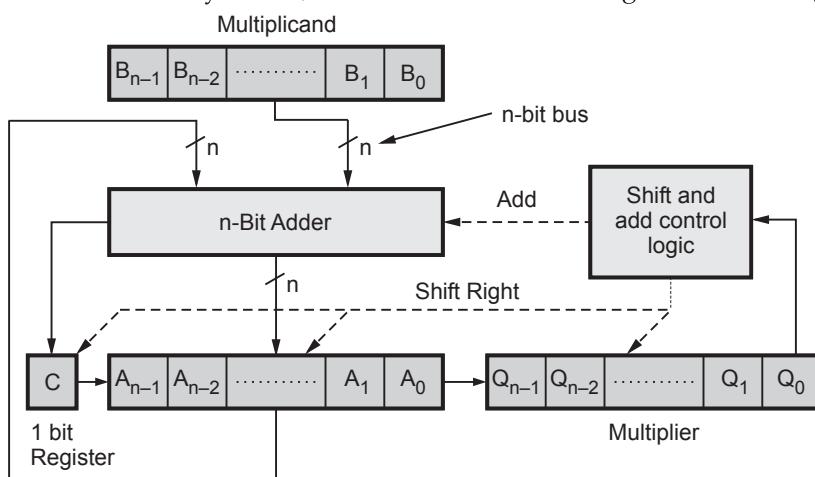
11.7 The 4-bit Multiplier Circuit using ALU and Shift Registers

- The multiplication is a complex operation than addition and subtraction. It can be performed in **hardware** or **software**.
- Fig. 11.7.1 shows the usual algorithm for multiplying positive numbers by hand.
- Looking at this algorithms we can note following points :
 - Multiplication process involves generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.
 - In the binary system the partial products are easily defined. When the multiplier bit is 0, the partial product is 0, and when the multiplier is 1, the partial product is the multiplicand.
 - The final product is produced by summing the partial products. Before summing operation each successive partial product is shifted one position to the left relative to the preceding partial product, as shown in the Fig. 11.7.1.
 - The product of two n-digit numbers can be accommodated in $2n$ digits, so the product of the two 4-bit numbers in fits into 8-bits.
- Fig. 11.7.2 shows the implementation of manual multiplication approach. It consists of n-bit binary adder, shift and add control logic and four registers, A, B, C.

$$\begin{array}{r}
 1101 \quad (13) \quad \text{Multiplicand} \\
 \times 1001 \quad (9) \quad \text{Multiplier} \\
 \hline
 1101 \\
 0000 \\
 0000 \\
 1101 \\
 \hline
 1110101 \quad \text{Final product (117)}
 \end{array}$$

} Partial products

Fig. 11.7.1 Manual multiplication algorithm



Note : Dotted lines indicate control signals

Fig. 11.7.2 Hardware implementation of unsigned binary multiplication

C and Q. As shown in the Fig. 11.7.2 multiplier and multiplicand are loaded into register Q and register B, respectively, and C are initially set to 0.

Multiplication Operation Steps

1. Bit 0 of multiplier operand (Q_0 of Q register) is checked.
 2. If bit 0 (Q_0) is one then multiplicand and partial product are added and all bits of C, A and Q registers are shifted to the right one bit, so that the C bit goes into A_{n-1} , A_0 goes into Q_{n-1} , and Q_0 is lost. If bit 0 (Q_0) is 0, then no addition is performed, only shift operation is carried out.
 3. Steps 1 and 2 are repeated n times to get the desired result in the A and Q registers.
- A flowchart for multiplication operation is shown in Fig. 11.7.3.

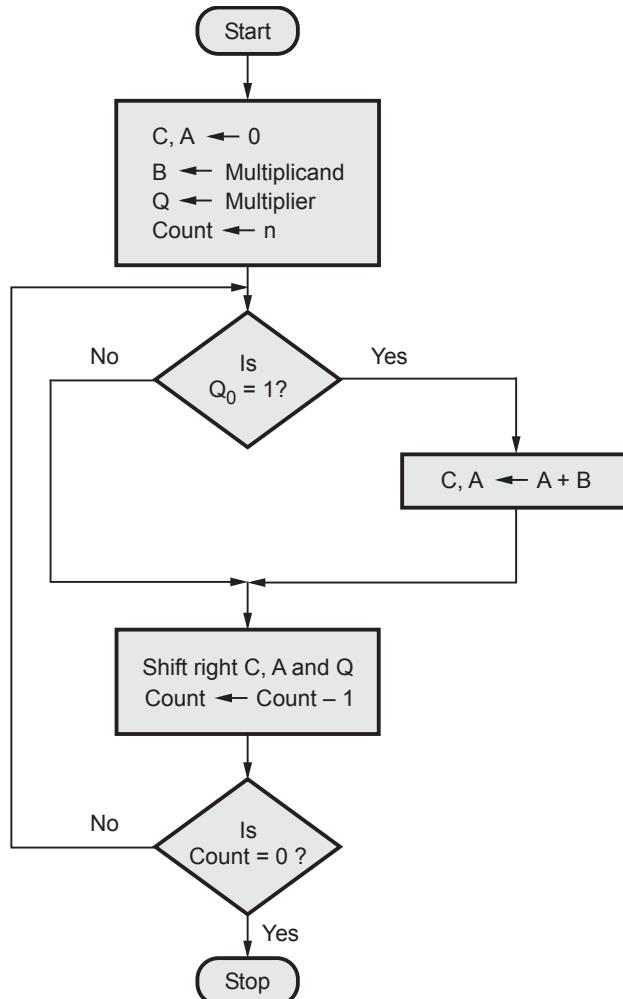


Fig. 11.7.3 Flowchart for multiplication operation

- Let us see one example.

Consider 4-bit multiplier and multiplicand :

$$\text{Multiplicand} = 1 \ 1 \ 0 \ 1 \quad \text{and} \quad \text{Multiplier} = 1 \ 0 \ 1 \ 1$$

Fig. 11.7.4 shows operations involved and their results in the multiplication process.

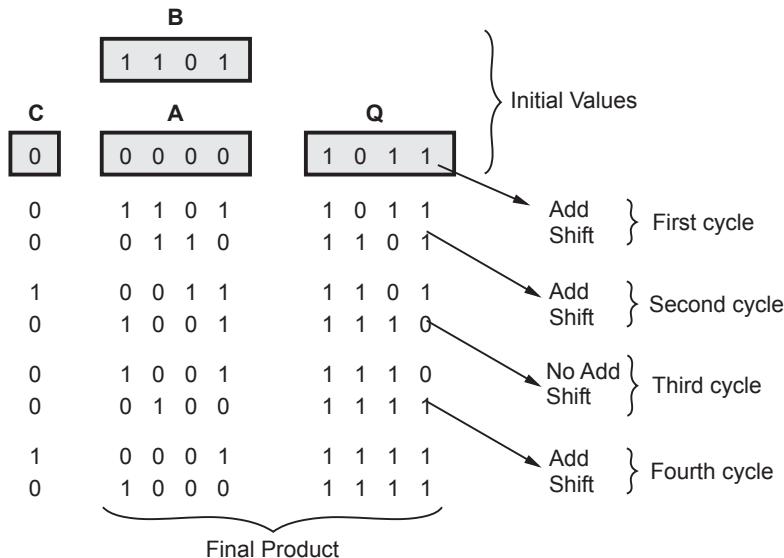


Fig. 11.7.4 Multiplication process

Review Questions

- Explain an algorithm to multiply two positive numbers. Also discuss the realization of a multiplier to implement the same.
- Design a multiplier that multiplies two 4-bit numbers.
- Explain the operation of sequential circuit binary multiplier with
Multiplicand 1101
Multiplier 1011.
- Draw flowchart hardware multiplication algorithm and explain it.

11.8 Memory Organization

- Memories are made up of registers. Each register in the memory is one storage location. Each location is identified by an **address** called **memory address**. The number of storage locations can vary from a few in some memories to hundreds of thousand in others.

- Each location can accommodate one or more bits. Generally, the total number of bits that a memory can store is its capacity. Most of the types the capacity is specified in terms of bytes (group of eight-bits).
- Each register consists of storage elements (flip-flops or capacitors in semiconductor memories and magnetic domain in magnetic storage), each of which stores one-bit of data. A storage element is called a **cell**.
- The data stored in a memory by a process called **writing** and are retrieved from the memory by a process called **reading**. Fig. 11.8.1 illustrates in a very simplified way the concept of write, read, address and storage capacity for a generalized memory.

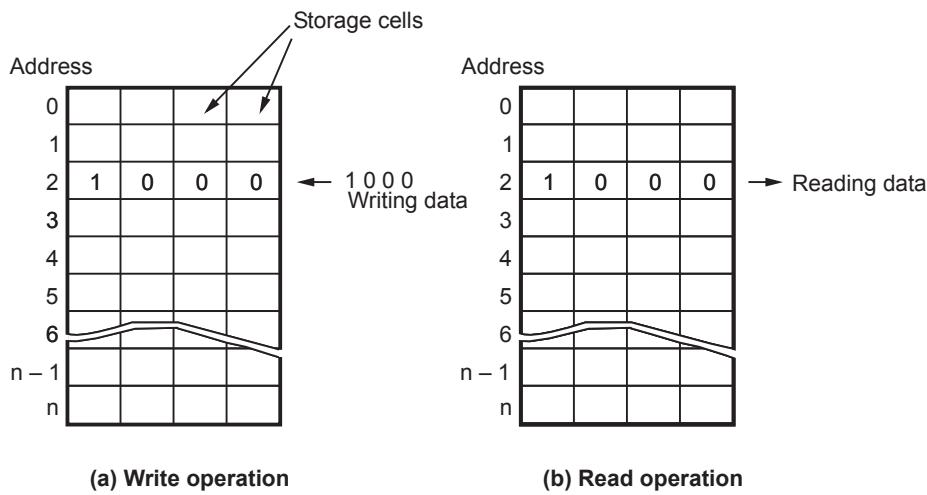


Fig. 11.8.1

- As shown in the Fig. 11.8.1, a memory unit stores binary information in groups of bits called **words**. A word in memory is an entity of bits that moves in and out of storage as a unit. A word having group of 8-bits is called a **byte**. Most computer memories use words that are multiples of eight-bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is made of 4 bytes.
- The communication between a memory and its environment is achieved through data lines, address selection lines and control lines that specify the direction of transfer.

11.8.1 Block Diagram of Memory Unit

- Fig. 11.8.2 shows the block diagram of memory unit. The n data lines provide the information to be stored in memory and the k address lines specify the particular word chosen among the many available. The control input R/W (read and write) specify the direction transfer.

- When R/W signal is HIGH, read operation is activated and data bus will act as input for memory.
- When R/W signal is LOW, write operation is activated and data bus will act as output for memory.
- Another control signal, chip select (CS) or Chip Enable (CE) is used to enable the memory chip for read/write operation. The CS input selects the specified memory chip in a multichip memory system.

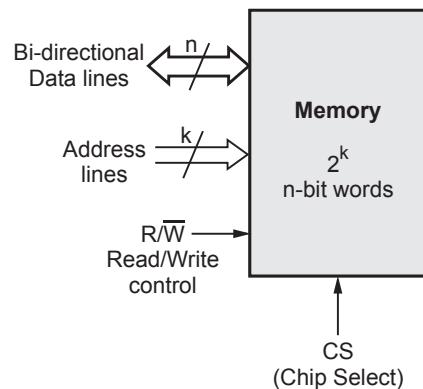


Fig. 11.8.2 Block diagram of memory unit

11.8.2 Digital Circuit using Decoder and Registers for Memory Operations

- Fig. 11.8.3 shows the internal organization of 16 X 4 memory chip. Here, the organization is shown with detail connections of address lines, data lines and control lines.

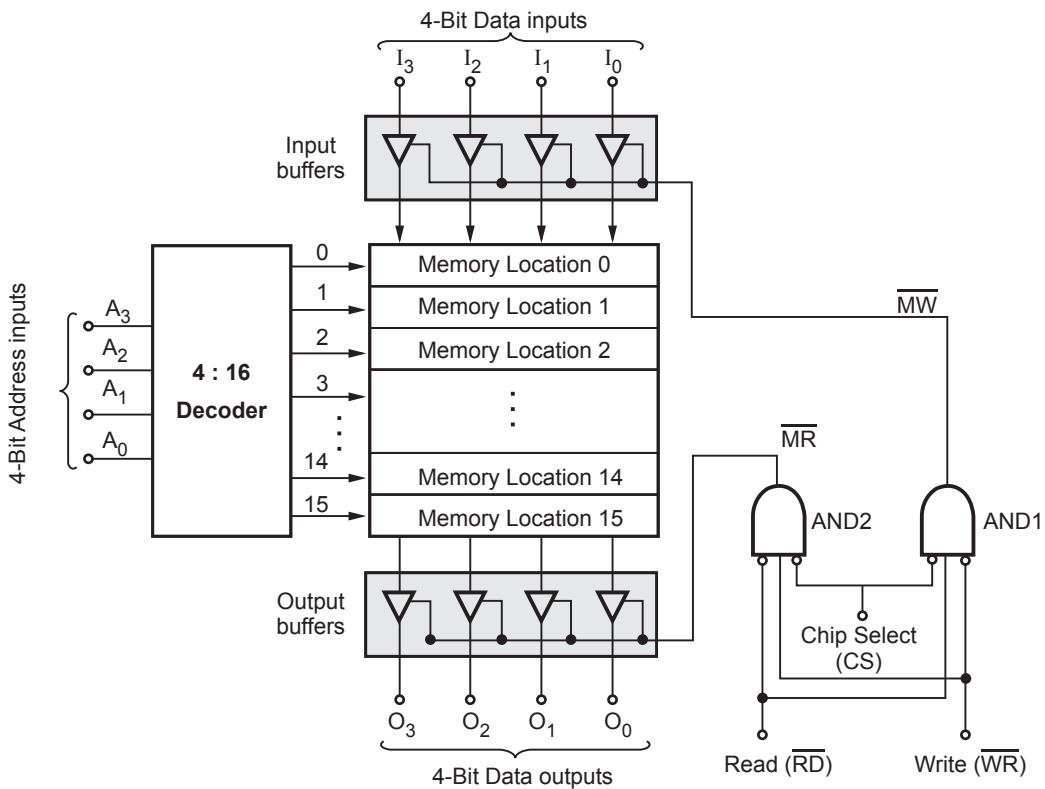


Fig. 11.8.3 Internal organization of 16 X 4 memory chip

- The 4 : 16 decoder is used to decode the contents on address lines and select one of the sixteen possible memory locations.
- Two AND gates circuitry controlled by (\overline{RD}) , (\overline{WR}) and (\overline{CS}) control signals is used to active read/write operation. Note that, here (\overline{RD}) and (\overline{WR}) are the two separate signals.
- Input and output buffers are enabled according to the operation to be performed. For memory write input buffers are enabled and for memory read output buffers are enabled.

Review Questions

- Write a note on memory organization.*
- Draw and explain the block diagram of memory unit.*
- Draw and explain the internal organization of 16 X 4 memory chip.*

11.9 Memory Operation

- The memory unit supports two basic operations : **read** and **write**. The read operation reads previously stored data and the write operation stores a new value in memory.
- Both of these operations require a memory address. In addition, the write operation requires specification of the data to be written.
- Two metrics are used to characterize memory :
- Access time** refers to the amount of time required by the memory to retrieve the data at the addressed location.
- The other metric is the **memory cycle time**, which refers to the minimum time between successive memory operations.
- The **read operation is non-destructive** in the sense that one can read a location of the memory as many times as one wishes without destroying the contents of that location. The write operation, however, is destructive, as writing a value into a location destroys the old contents of that memory location.

11.9.1 Read Operation

Steps to perform memory read operation are as follows :

- Place the address of the location to be read on the address bus.
- Activate the memory read control signal on the control bus.
- Read the data from the data bus.
- De-activate the memory read control signal to terminate the read cycle

- Fig. 11.9.1 shows the timing diagram for read cycle.

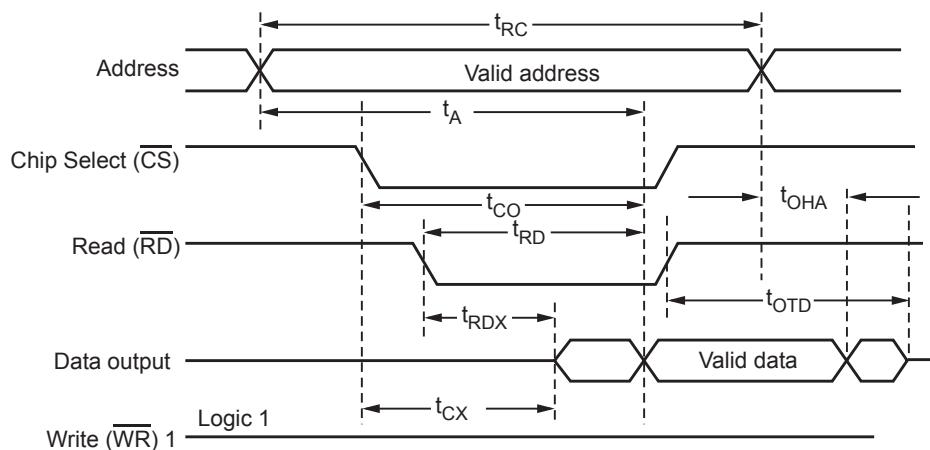


Fig. 11.9.1 Read cycle timing waveform

- The timing diagram is drawn on the basis of different timing parameters. These are as follows :

 - t_{RC} : Read Cycle Time :** It is the minimum time for which an address must be held stable on the address bus, in read cycle.
 - t_A : Address Access Time :** It is the maximum specified time within which a valid new data is available on the data bus after an address is applied.
 - t_{RD} : Read to Output Valid Time :** It is the maximum time delay after (\overline{RD}) goes LOW (beginning of read pulse) and the availability of valid data on the data bus. Using this timing parameter we can specify the maximum rate at which data can be read. It is given as

The maximum rate at which data can be read = $\frac{1}{t_{RD}}$

- For example, if $t_{RD} = 100$ ns,

$$\text{The maximum rate at which data can be read} = \frac{1}{100 \times 10^{-9}} = 10 \times 10^6 \text{ words/s}$$

- t_{RDX} : Read to Output Active Time :** It is the minimum time delay after (\overline{RD}) goes LOW (beginning of read pulse) and the output buffers coming to active state.
- t_{CO} : Chip Select to Output Valid Time :** It is the maximum time delay after (\overline{CS}) goes LOW (beginning of chip select pulse) and the availability of valid data on the data bus.

6. t_{CX} : Chip Select to Output Active Time : It is the minimum delay after (\overline{CS}) goes LOW (beginning of chip select pulse) and the output buffers coming to active state.
7. t_{OTD} : Output tri-state from Read : It is the maximum time delay after (\overline{RD}) goes HIGH (end of read pulse) and the output buffers going to high impedance state.
8. t_{OHA} : Data Hold Time : It is the minimum time for which the valid data is available on the data output after the address ends.

11.9.2 Write Operation

Steps to perform memory write operation are as follows :

1. Place the address of the location to be written on the address bus.
 2. Place the data to be written on the data bus.
 3. Activate the memory write control signal on the control bus.
 4. De-activate the memory write control signal to terminate the write cycle
- Fig. 11.9.2 shows the write cycle for memory.

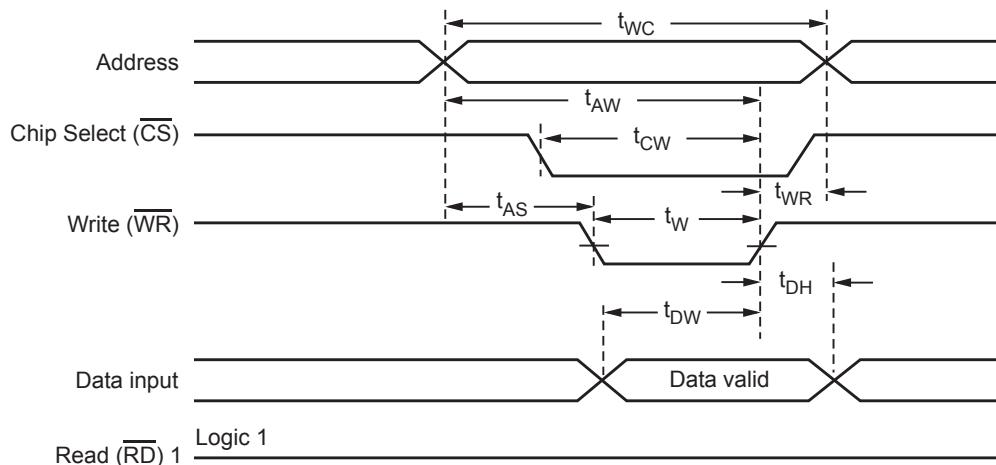


Fig. 11.9.2 Write cycle timing waveform

- The timing diagram is drawn on the basis of different timing parameters. These are as follows :
1. t_{WC} : Write Cycle Time : It is the minimum time for which an address must be held stable on the address bus, in write cycle. Using this timing parameter we can specify the maximum rate at which data can be stored. It is given as

$$\text{The maximum rate at which data can be stored} = \frac{1}{t_{WC}}$$

For example if $t_{WC} = 200 \text{ ns}$,

- The maximum rate at which data can be stored = $\frac{1}{200 \times 10^{-9}} = 5 \times 10^6 \text{ words/s}$
- 2. **t_{AW} : Address Valid to End of Write :** It is the time at which address must be applied on the address bus before WR goes high.
- 3. **t_{WR} : Write Recovery Time :** It is the time for which address will remain on address bus after WR goes high.
- 4. **t_{AS} : Address Setup Time :** When address is applied, it is the time after which WR can be made low.
- 5. **t_{CW} : Chip Selection to the End of Write :** It is the time at which the CS must be made low to select the device before WR goes high.
- 6. **t_W : Write Pulse Width :** It is the time for which WR goes low.
- 7. **t_{DW} : Data Valid to the End of Write :** It is the minimum time for which data must be valid on the data bus before WR goes high.
- 8. **t_{DH} : Data Hold Time :** It is the time for which data must be held valid after WR goes high.

Review Questions

1. Write a note on memory read operation.
2. Write a note on memory write operation.
3. Explain the read cycle waveform for memory.
4. Explain the write cycle waveform for memory.
5. Define timing parameters : t_{WR} , t_{WC} , t_{CO} , t_{RD} , t_{RC} .



Notes

Notes