

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.
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 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



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. Kimbahunne

Dr. S. M. Sangue

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	.	9
In power of 10	5 10^3	6 10^2	7 10^1	8 10^0	.	9 10^{-1}
	MSD				LSD	

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.

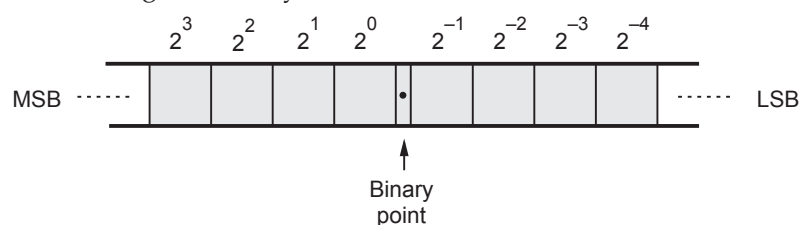


Fig. 1.2.2 Binary position values as a power of 2

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

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).

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

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

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

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

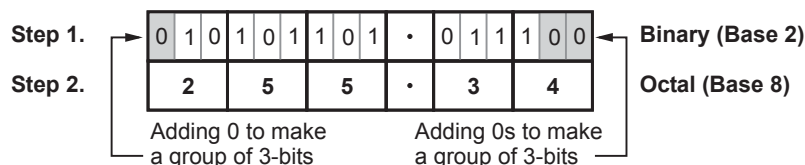
- The base for octal number is 8 and the base for binary number is 2.
- The base for octal number is the third power of the base for binary numbers. Therefore, by grouping 3 digits of binary numbers and then converting each group digit to its octal equivalent we can convert binary number to its octal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



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

1.3.2 Octal to Binary Conversion

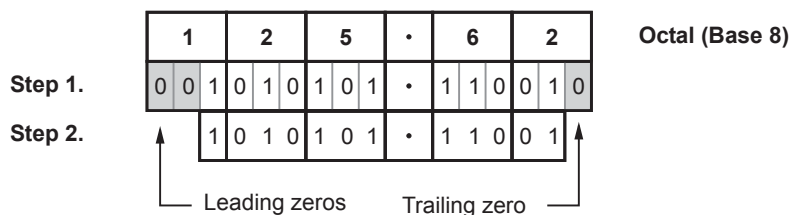
- Conversion from octal to binary is a reversal of the process explained in the previous section. Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

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.



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

1.3.3 Binary to Hexadecimal Conversion

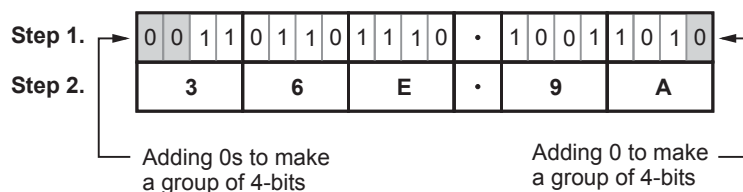
- The base for hexadecimal numbers is 16 and the base for binary numbers is 2.
- The base for hexadecimal number is the fourth power of the base for binary numbers. Therefore, by grouping 4 digits of binary numbers and then converting each group digit to its hexadecimal equivalent we can convert binary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



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

1.3.4 Hexadecimal to Binary Conversion

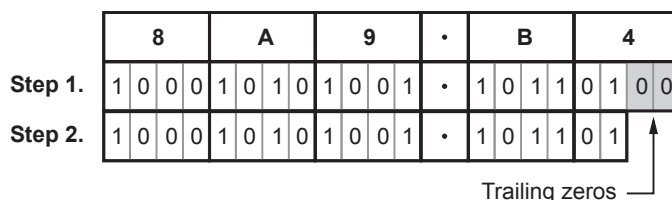
- Conversion from hexadecimal to binary is a reversal of the process explained in the previous section. Each digit of the hexadecimal number is individually converted to its binary equivalent to get hexadecimal to binary conversion of the number.

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.



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

1.3.5 Octal to Hexadecimal Conversion

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

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

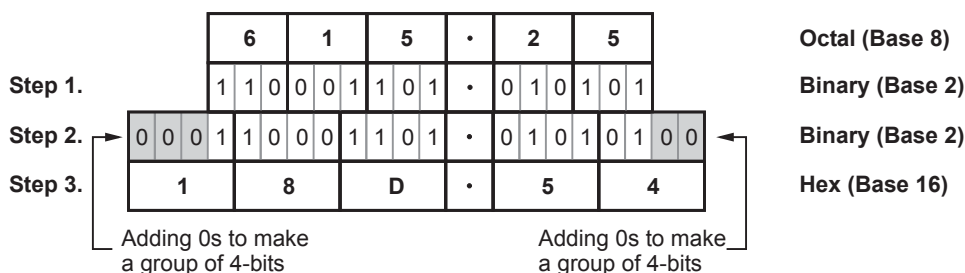
Example 1.3.5 Convert $(615.25)_8$ to its hexadecimal equivalent.

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.



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.

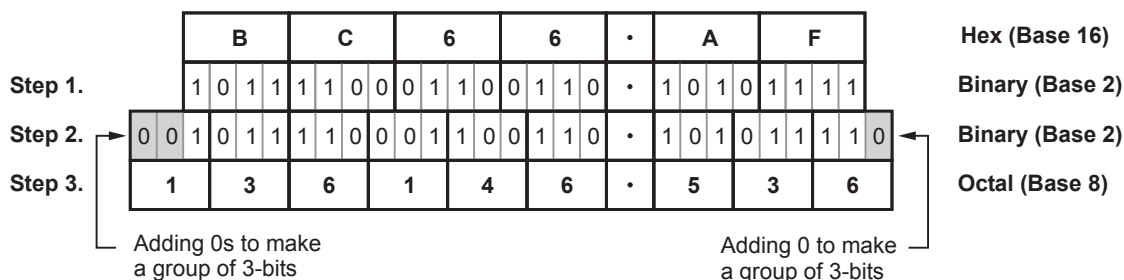
Example 1.3.6 Convert $(BC66.AF)_{16}$ to its octal equivalent.

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.



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

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

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

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

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

$$\text{Negative is not applicable} \quad \therefore x = 16$$

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

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

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

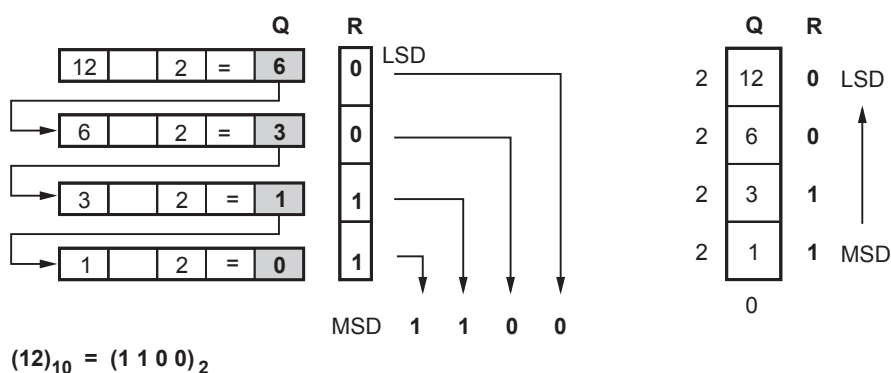
Steps in Successive Multiplication Method

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.

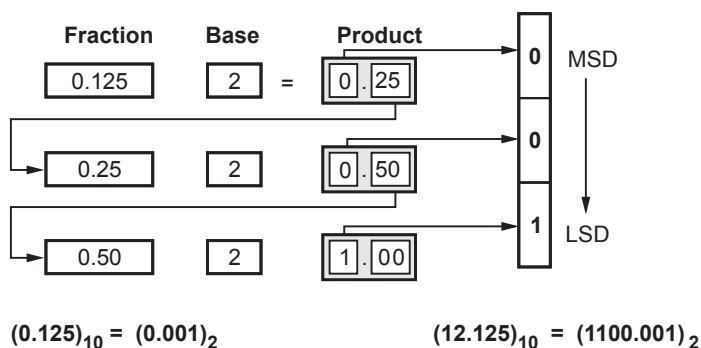
Example 1.3.9 Convert 12.125 decimal into binary.

Solution :

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



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

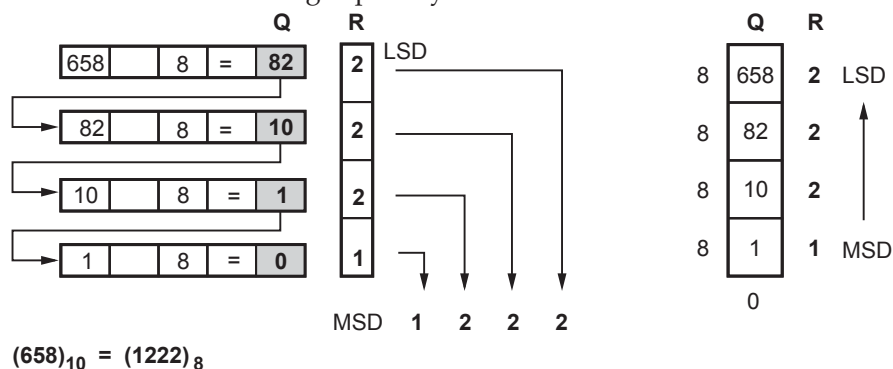


$$(12.125)_{10} = (1100.001)_2$$

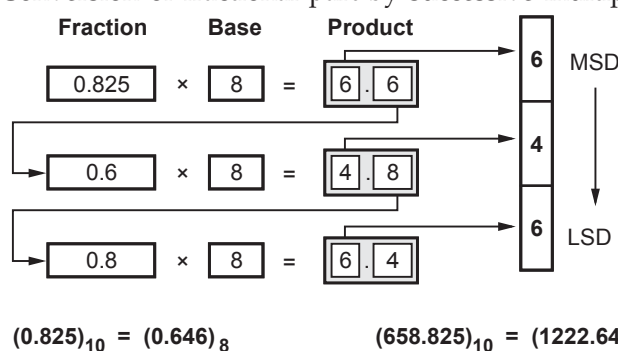
Example 1.3.10 Convert 658.825 decimal into octal.

Solution :

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



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

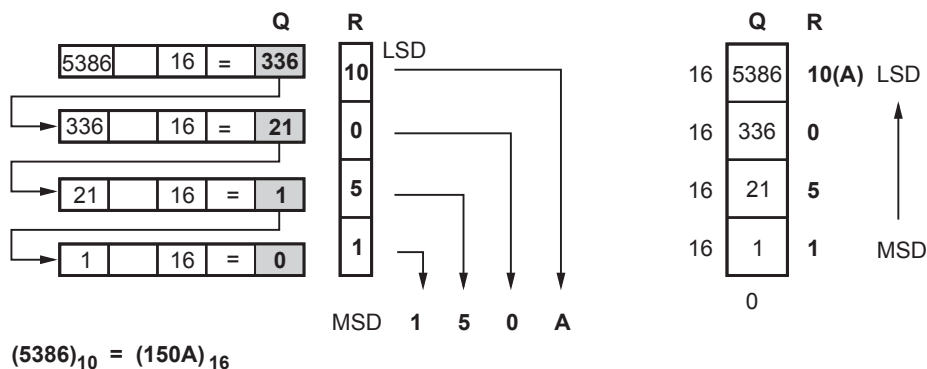


In this example, we have restricted fractional part up to 3 digits. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 8 until we have as many digits as necessary for our application.

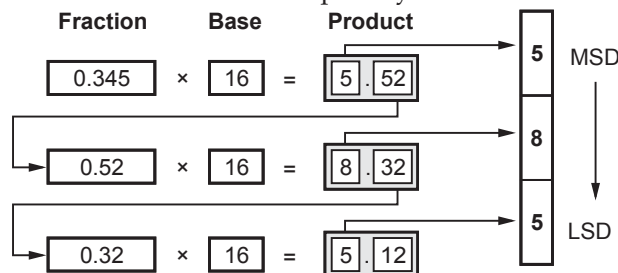
Example 1.3.11 Convert 5386.345 decimal into hexadecimal.

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.

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) $(292)_{10} = (1204)_b$ 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.

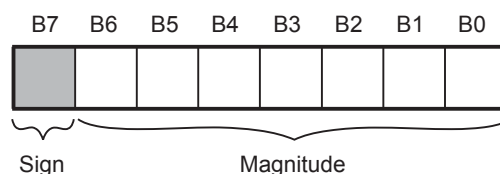


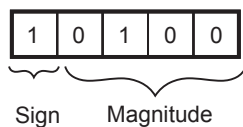
Fig. 1.4.1

1. $+6 = 0000\ 0110$
2. $-14 = 1000\ 1110$
3. $+24 = 0001\ 1000$
4. $-64 = 1100\ 0000$

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,

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

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	
		1	0	0	1	1	1	0	0	0

1's complement

Add 1

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. $\text{Result} \leftarrow A + 1's \text{ complement of } B$.
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 : $(28)_{10} = (011100)_2$
 $(15)_{10} = (001111)_2$

	0 0 1 1 1 1	(15) ₁₀	
			Carry
	1 1 0 0 0 0	1's complement of (15) ₁₀	
Sign Extension →	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">+</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; text-align: center;"> 1 1 0 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 0 1 </div> </div>	Carry Binary equivalent of (28) ₁₀ 1's complement of 15, i.e. (-15) ₁₀ Result Add end around carry Final result : Binary equivalent of (13) ₁₀	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">+</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; text-align: center;"> (28)₁₀ (-15)₁₀ (13)₁₀ </div> </div>

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

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

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

	0 1 1 1 0 0	Binary equivalent of (28) ₁₀	
	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">+</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; text-align: center;"> 1 1 1 0 0 0 1 1 </div> </div>	Carry 1's complement of (28) ₁₀	
+	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">+</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; text-align: center;"> 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0 1 0 </div> </div>	Carry Binary equivalent of (15) ₁₀ 1's complement of (28) ₁₀ Result = Binary equivalent of (-13) ₁₀	<div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">+</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; text-align: center;"> (15)₁₀ (-28)₁₀ (-13)₁₀ </div> </div>

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 $\leftarrow A + 2$'s 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 : $(28)_{10} = (011100)_2$

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

		0	0	1	1	1	1	$(15)_{10}$	
		1	1	0	0	0	0	Carry	
							1	1's complement of $(15)_{10}$	
	+							Add 1	
		1	1	0	0	0	1	2's complement of 15, i.e., $(-15)_{10}$	
		1	1					Carry	
			0	1	1	1	0	Binary equivalent of $(28)_{10}$	$(28)_{10}$
Sign Extension	+	1	1	0	0	0	1	2's complement of 15, i.e., $(-15)_{10}$	$(-15)_{10}$
Ignore Carry		1	0	0	1	1	0	Result : Binary equivalent of $(13)_{10}$	$(13)_{10}$

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

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

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

		0	1	1	1	0	0	Binary equivalent of $(28)_{10}$	
				1	1			Carry	
		1	0	0	0	1	1	1's complement of $(28)_{10}$	
	+						1	Add 1	
		1	0	0	1	0	0	2's complement of $(28)_{10}$, i.e., $(-28)_{10}$	
No carry	→	0		1	1			Carry	$(15)_{10}$
			0	0	1	1	1	Binary equivalent of $(15)_{10}$	$(15)_{10}$
	+		1	0	0	1	0	2's complement of $(28)_{10}$	$(-28)_{10}$
		1	1	0	0	1	1	No carry, thus result is negative and in 2's complement form	$(-13)_{10}$
Verification			0	0	1	1	0	1's complement of result	
	+						1	Add 1	
		0	0	1	1	0	1	- Result = Binary equivalent of $(13)_{10}$	

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	BCD code			
Digit	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			
8-4-2-1 BCD	0	1	0	1	1	0	0	0

Fig. 1.5.1

Advantages

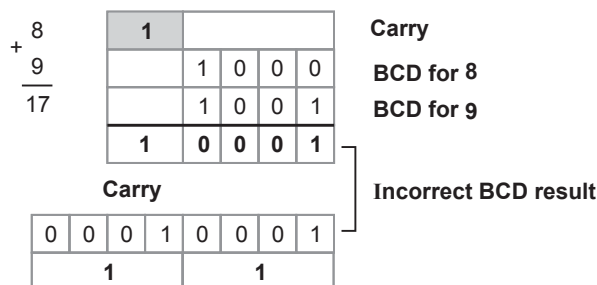
1. Easy to convert between it and decimal.

Disadvantages

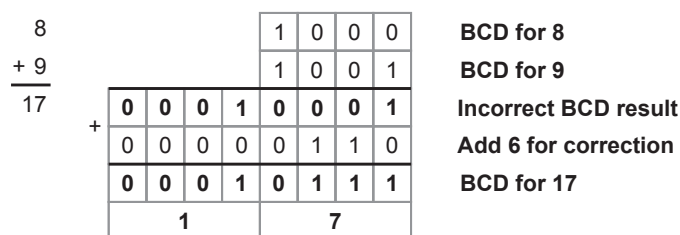
1. Since it requires more binary digits to represent multi-digit number than binary number system. It is less **efficient**. For example, to represent the same number (58) in binary : 111010_2 , we require only 6 digits.
2. Arithmetic operations are more complex.

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.



- 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.



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 (0110₂) to the four-bit sum and ignore carry of this addition.
 - Add 1 (0001₂) 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.

$$\begin{array}{rcl}
 592_{10} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline \end{array} \\
 \text{9's complement of } 592_{10} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}
 \end{array}
 \quad \begin{array}{l} \leftarrow \text{Complement of} \\ \leftarrow \text{each other} \end{array}$$

b) By referring Table 1.5.2.

$$\begin{array}{rcl}
 403_{10} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 \text{9's complement of } 403_{10} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline \end{array}
 \end{array}
 \quad \begin{array}{l} \leftarrow \text{Complement of} \\ \leftarrow \text{each other} \end{array}$$

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.

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

Example 1.5.2 Convert gray code 101011 into its binary equivalent.

Solution :

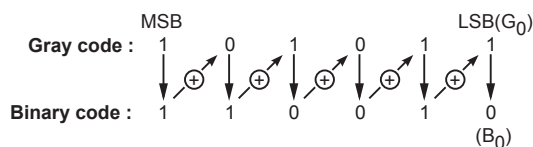


Fig. 1.5.2

$$\therefore (1\ 0\ 1\ 0\ 1\ 1)_{\text{gray}} = (1\ 1\ 0\ 0\ 1\ 0)_2$$

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

$$\begin{aligned} B_n (\text{MSB}) &= G_n (\text{MSB}) & B_2 &= B_3 \oplus G_2 \\ B_{n-1} &= B_n \oplus G_{n-1} & \dots & \\ B_{n-2} &= B_{n-1} \oplus G_{n-2} & B_1 &= B_2 \oplus G_1 \\ & & B_0 &= B_1 \oplus G_0 \end{aligned}$$

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 1011011 in binary into its equivalent gray code.

Solution :

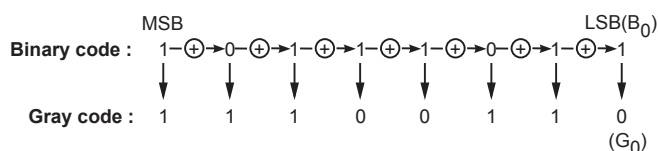


Fig. 1.5.3

$$\therefore (1\ 0\ 1\ 1\ 0\ 1\ 1)_2 = (1\ 1\ 1\ 0\ 0\ 1\ 0)_{\text{gray}}$$

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

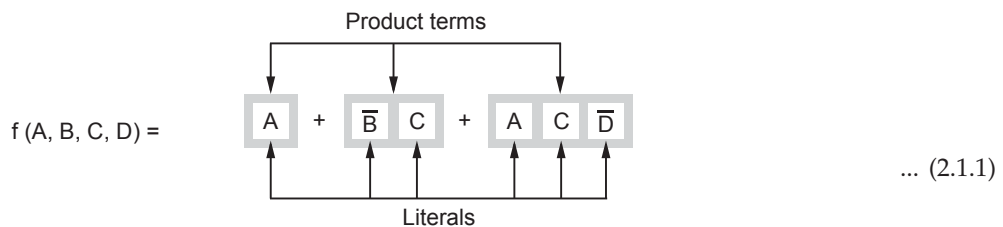
- 2.1 Boolean Expressions
- 2.2 Karnaugh (K-Map) Minimization
- 2.3 Simplification of SOP Expression **Dec.-09, May-13,19 , Marks 12**
- 2.4 Simplification of POS Expression. **Dec.-15, Marks 6**
- 2.5 Summary of Rules for K-Map Simplification
- 2.6 Limitations of Karnaugh Map
- 2.7 Quine-McCluskey or 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

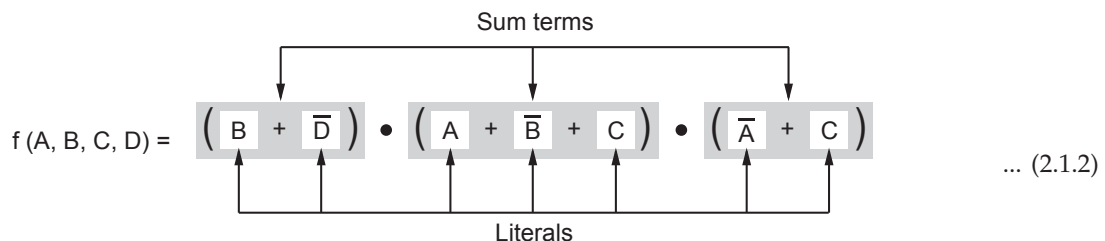
- Boolean expressions are constructed by connecting the Boolean constants and variables with the Boolean operations. These Boolean expressions are also known as **Boolean formulas**. We use Boolean expressions to describe switching function or **Boolean functions**. For example, if the Boolean expression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



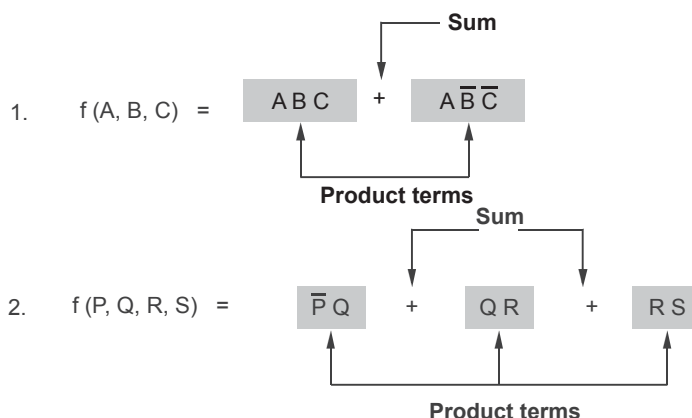
- In this Boolean function the variables are appeared either in a complemented or an uncomplemented form. Each occurrence of a variable in either a complemented or an uncomplemented form is called a **literal**. Thus, the above Boolean function 1.1.1 consists of six literals. They appear in the product 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 $A C \bar{D}$. 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

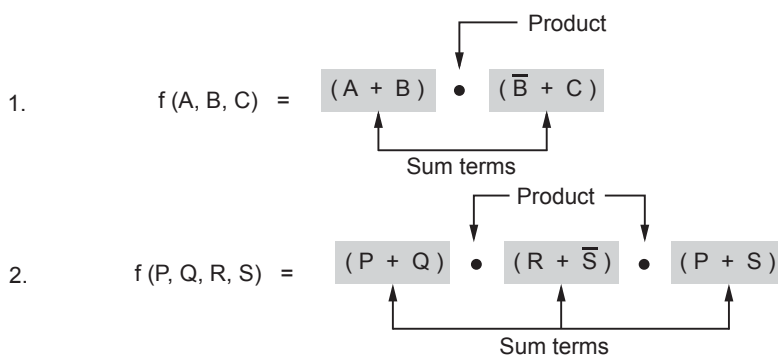
The words sum and product are derived from the symbolic representations of the OR and AND functions by + and \cdot (addition and multiplication), respectively. But we realize that these are not arithmetic operators in the usual sense. A product term is any group of literals that are ANDed together. For 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 $f(A, B, C) =$  $+$  $+$ . Each product term consists of all literals in either complemented form or uncomplemented form. 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) = \text{

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 form 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.**

$$f(A, B, C) = \boxed{AC} + \boxed{AB} + \boxed{BC}$$

Literal A is missing.
 Literal C is missing.
 Literal B is missing.

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

$$f(A, B, C) = \boxed{AC} \cdot (B + \bar{B}) + \boxed{AB} \cdot (C + \bar{C}) + \boxed{BC} \cdot (A + \bar{A})$$

Missing literals and their complements

Step 3 : Expand the terms and reorder literals.

$$\text{Expand : } f(A, B, C) = ACB + AC\bar{B} + ABC + AB\bar{C} + BCA + BC\bar{A}$$

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

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.

$$f(A, B, C) = ABC + A\bar{B}C + \boxed{ABC} + AB\bar{C} + \boxed{ABC} + \bar{A}BC$$

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

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.**

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

Literals B and C are missing

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

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

Original terms

Missing literals and their complements

Step 3 : Expand the terms and reorder literals.

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

Step 4 : Omit repeated product term

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

Repeated product term

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

Examples for Practice

Example 2.1.3 Express the following function in canonical SOP form

$$F_1 = AB + \overline{C}D + ABC$$

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

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

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

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.

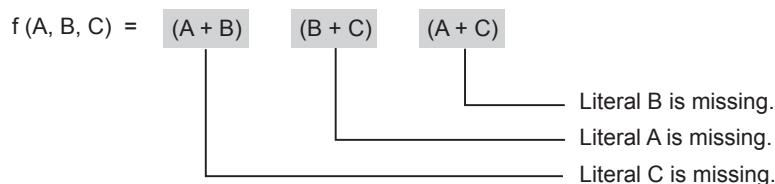
Step 4 : Reduce the expression by omitting repeated sum terms if any. Because $A \cdot A = A$.

Illustrative Examples

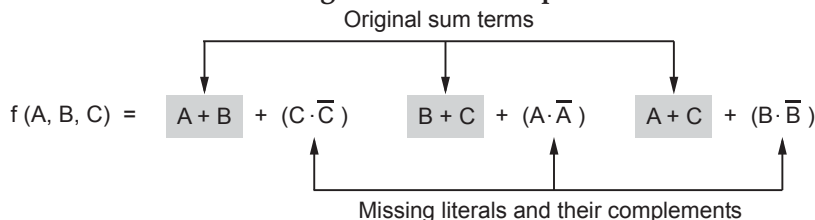
Example 2.1.5 Convert the given expression in canonical POS form.

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

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,

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

Reorder :

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

Step 4 : Omit repeated sum terms.

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.

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

Literals B and C are missing

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.

$$\begin{aligned} f(A, B, C) &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C}) \overbrace{(A + B + C)}^{\text{Repeated sum term}} \\ f(A, B, C) &= (A + B + C)(A + \bar{B} + C)(A + B + \bar{C})(A + \bar{B} + \bar{C}) \end{aligned}$$

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 :

$$\begin{aligned}
 1. \quad f(A, B, C) &= \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} \\
 &= m_0 + m_1 + m_3 + m_6 \\
 &= \sum m(0, 1, 3, 6) \\
 2. \quad f(A, B, C) &= (A + B + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C) \\
 &= M_1 \cdot M_3 \cdot M_6 \\
 &= \Pi M(1, 3, 6)
 \end{aligned}$$

where \sum denotes **sum of product** while Π 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

$\leftarrow \overline{A}\overline{B}\overline{C}$

$\leftarrow \overline{A}\overline{B}C$

$\leftarrow A\overline{B}\overline{C}$

Table 2.1.2

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

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

- 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

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

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

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

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

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

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

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

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

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

Example 2.1.10 Express the Boolean function as

(1) POS form (2) SOP form

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

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

2. SOP form : $D = \bar{A}\bar{B} + \bar{A}C + B\bar{B} + BC$

$$= \bar{A}\bar{B} + \bar{A}C + BC \quad \because B\bar{B} = 0$$

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.

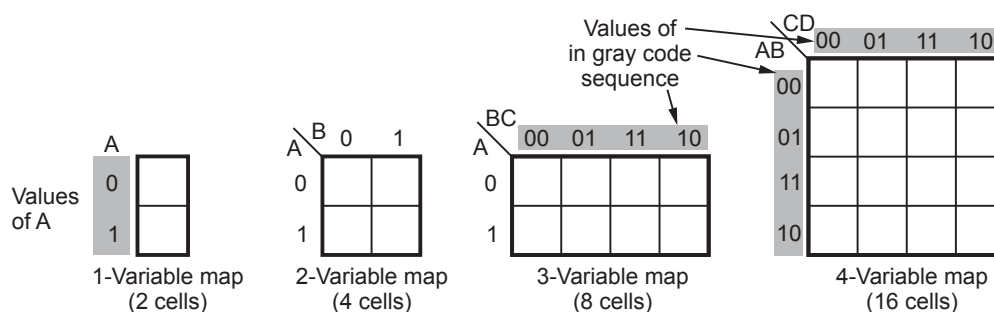


Fig. 2.2.1 Outlines of 1, 2, 3 and 4-variable Karnaugh maps

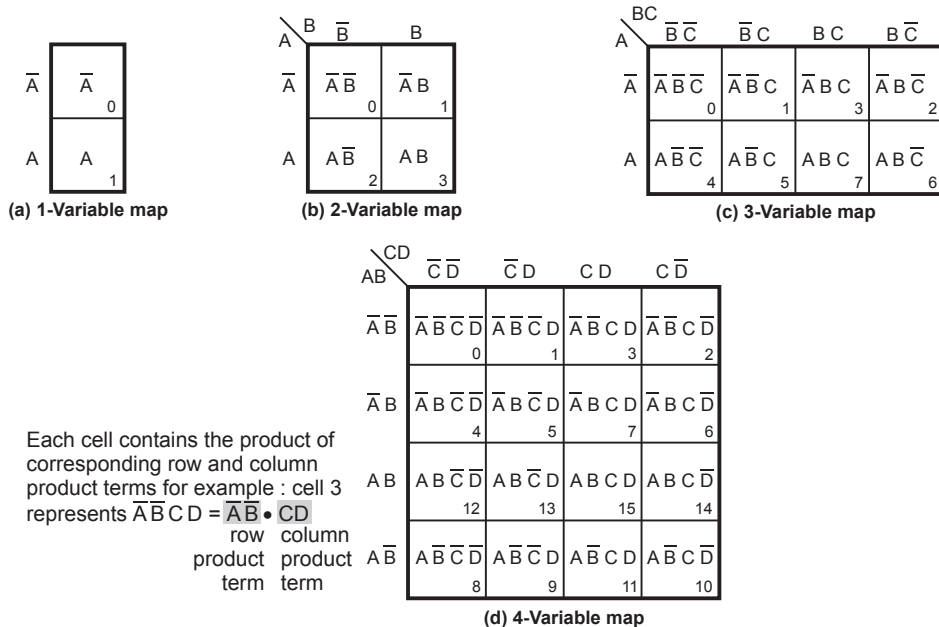


Fig. 2.2.2 1, 2, 3 and 4-variable maps with product terms

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

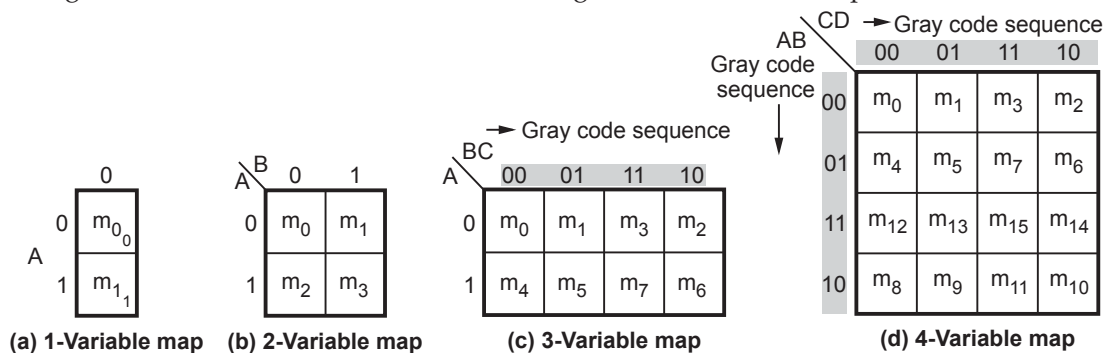


Fig. 2.2.3 Another way to represent 1, 2, 3 and 4-variable maps for SOP expressions

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.

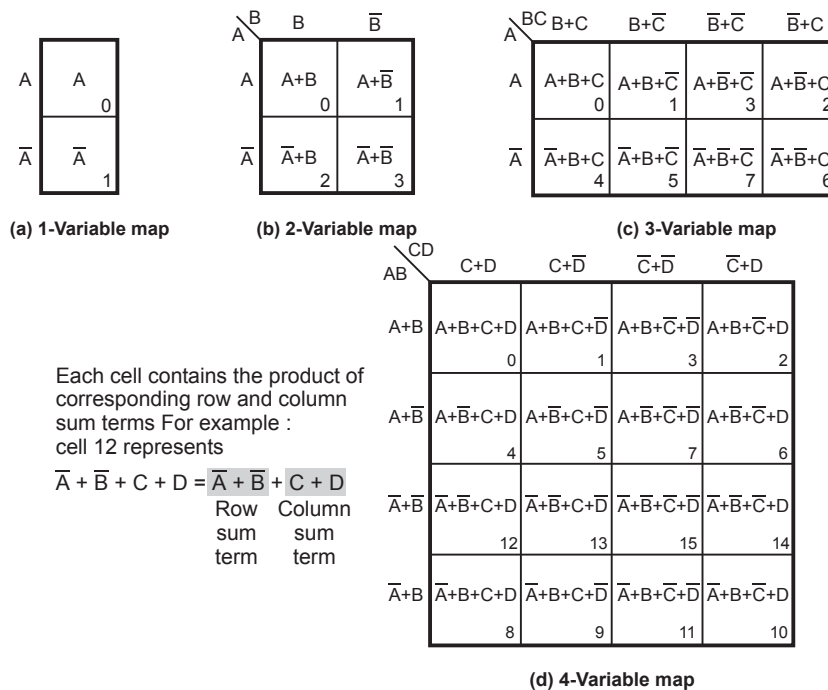


Fig. 2.2.4 1, 2, 3 and 4-variable maps with sum terms

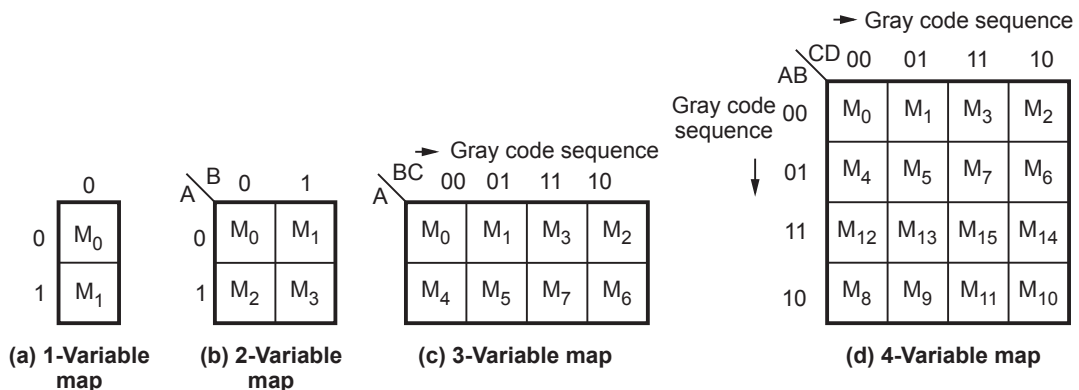


Fig. 2.2.5 Another way to represent 1, 2, 3 and 4-variable map for POS expressions

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 any 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.

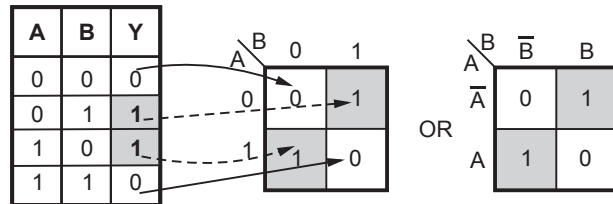


Fig. 2.2.6 (a) Representation of 2-variable truth table on K-map

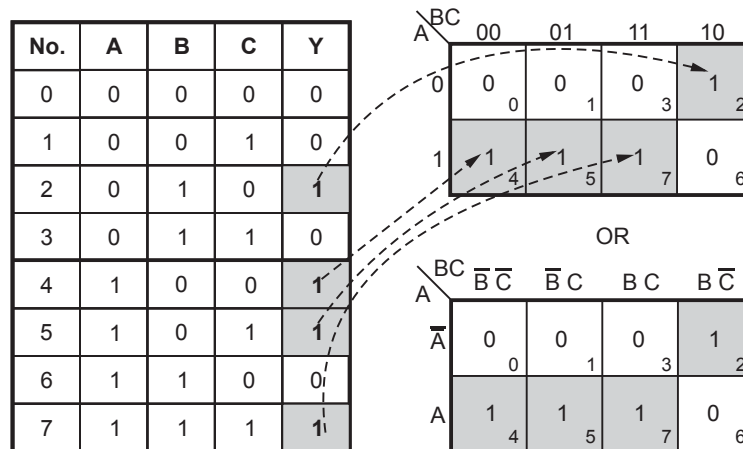
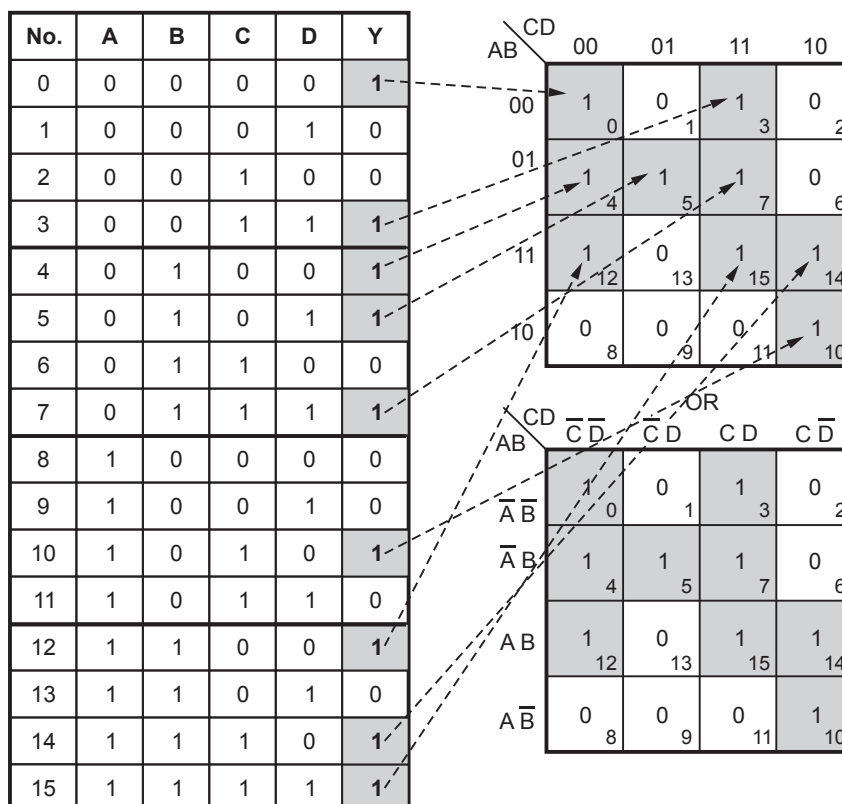


Fig. 2.2.6 (b) Representation of 3-variable truth table on K-map

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

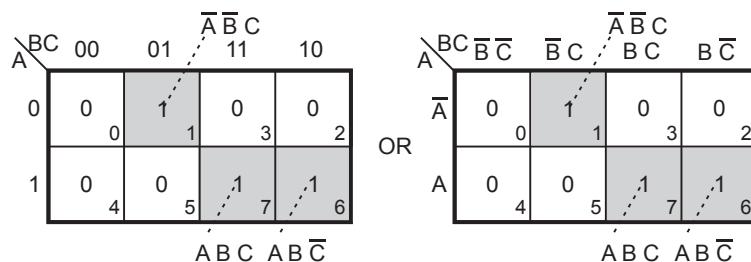


Fig. 2.2.7

Example 2.2.2 Plot Boolean expression

$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D$ on the Karnaugh map.

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

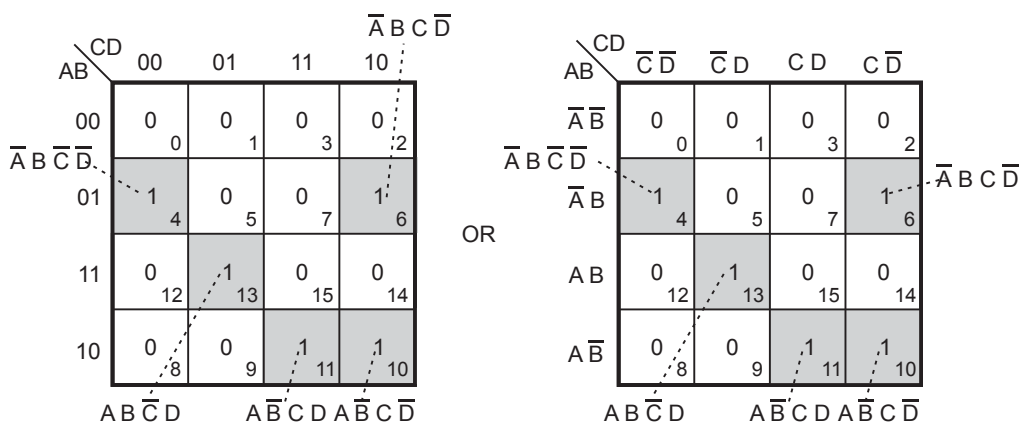


Fig. 1.2.8

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.

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

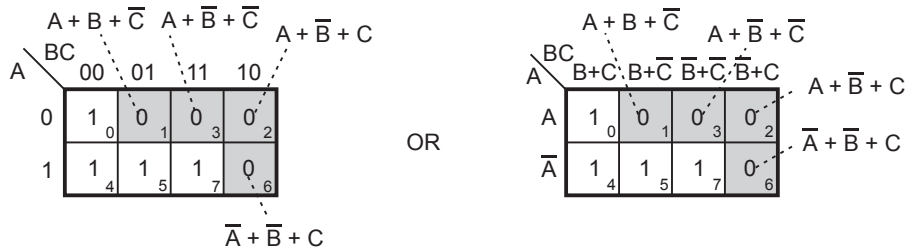


Fig. 2.2.9

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.

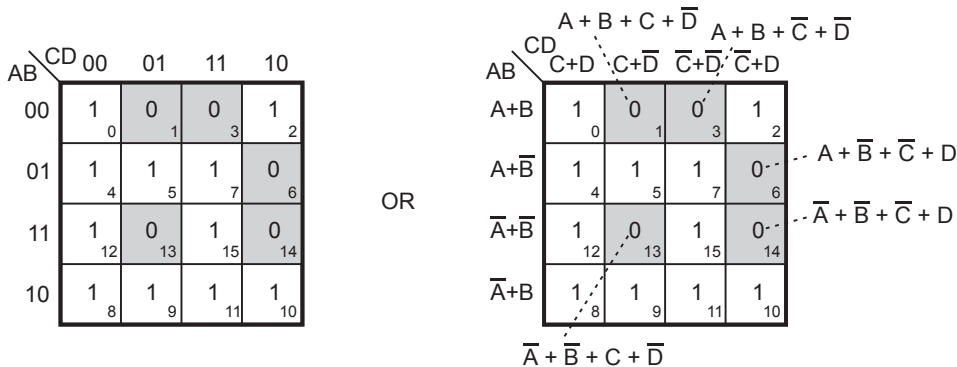


Fig. 2.2.10

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

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

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.

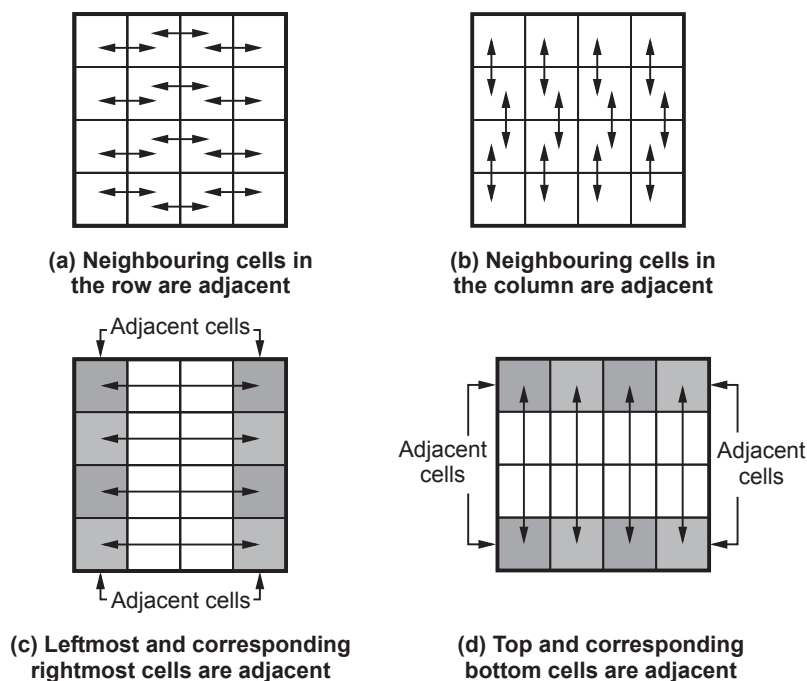


Fig. 2.2.11 Adjacent cells

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 $\bar{A}BC$. 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 :

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

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

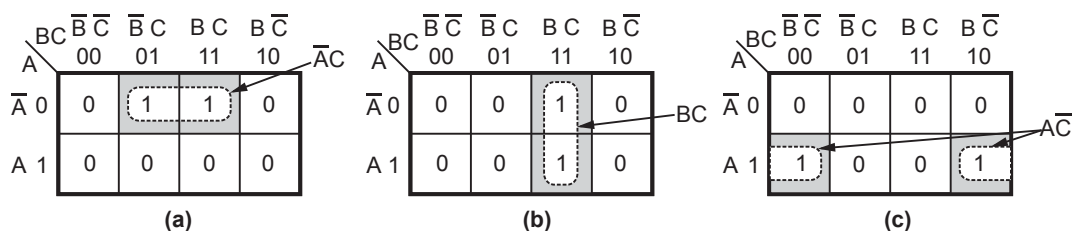


Fig. 2.2.12

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

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

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 :

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

Let us see another example shown in Fig. 2.2.12 (d). Here two 1s from top row and bottom row of some column are combined to eliminate variable A, since in a K-map the top row and bottom row are considered to be adjacent.

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

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.

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

Fig. 2.2.12 (f) shows a K-map where three group of pairs can be formed. But only two pairs are enough to include all 1s present in the K-map. In such cases third pair is not required.

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

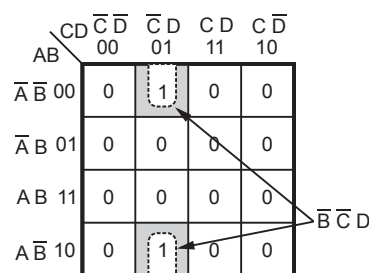


Fig. 2.2.12 (d)

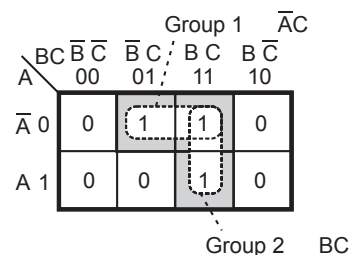


Fig. 2.2.12 (e)

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

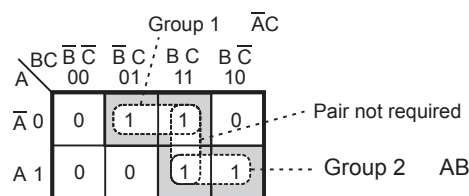


Fig. 2.2.12 (f) Examples of combining pairs of adjacent ones

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

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.

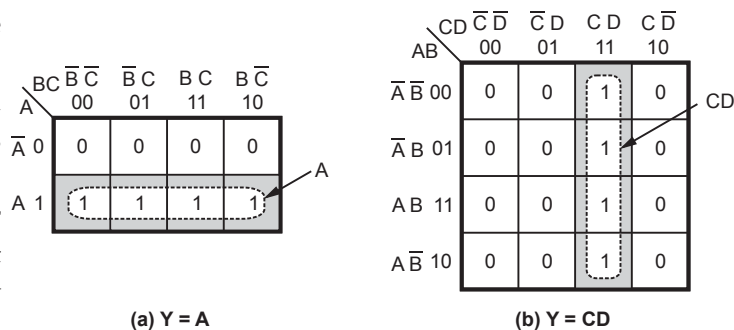


Fig. 2.2.13

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.

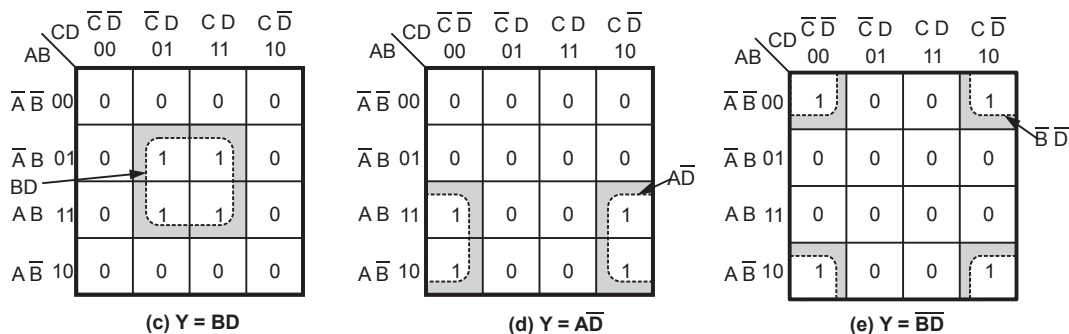


Fig. 2.2.13

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 :

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

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

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.

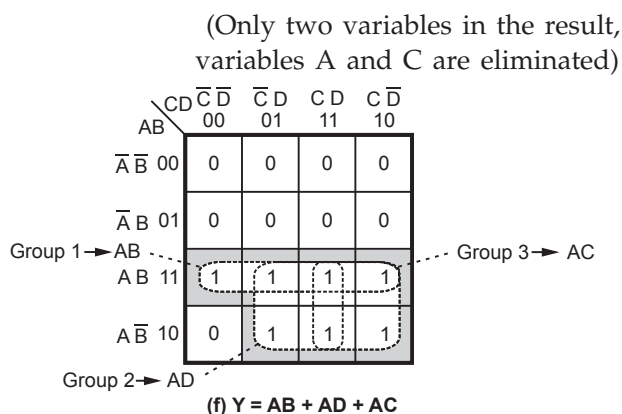


Fig. 2.2.13 Examples of combining quads of adjacent ones

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.

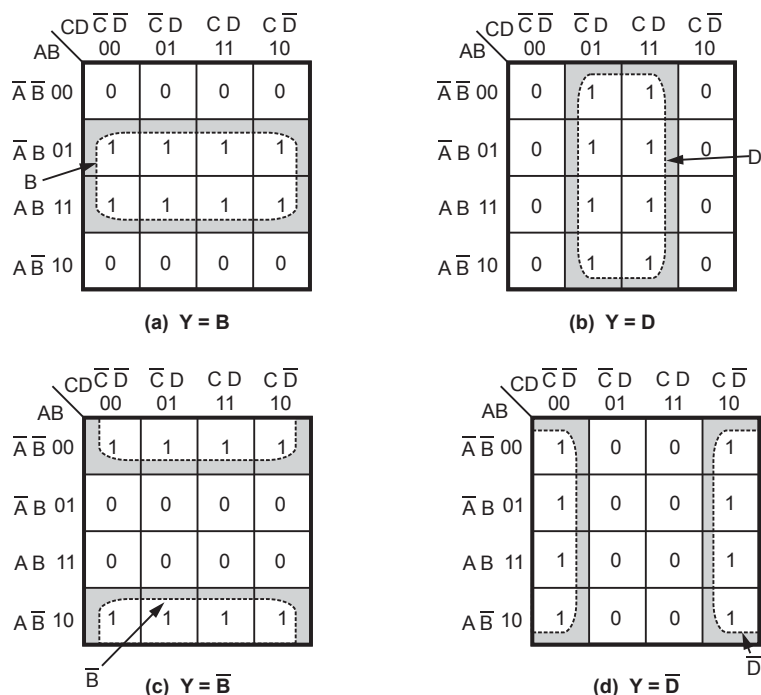


Fig. 2.2.14 Examples of combining octets of adjacent ones

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 :

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

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.

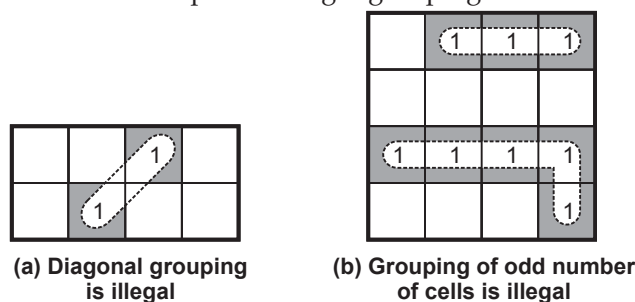


Fig. 2.2.15

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

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,

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

		BC	$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
A		00	01	11	10	
\overline{A}	0	1	1	1	0	2
A	1	1	1	0	0	6

Fig. 2.3.1 (a)

		BC	$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
A		00	01	11	10	
\overline{A}	0	1	1	1	0	
A	1	1	1	0	0	

Fig. 2.3.1 (b)

		BC	$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
A		00	01	11	10	
\overline{A}	0	1	1	1	0	$\overline{A}C$
A	1	1	1	0	0	

Fig. 2.3.1 (c)

Example 2.3.2 Minimize the expression

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

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

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

Fig. 2.3.2 (a)

Step 2 : Cell 2 is the only cell containing a 1 that is not adjacent to any other 1. It is referred to separately as group 1.

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

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.

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

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,

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

CD \ AB	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	$C\bar{D}$ 11	CD 10
$\bar{A}\bar{B}$ 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 (d)

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

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

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,

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

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

Fig. 2.3.3 (a)

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

Fig. 2.3.3 (b)

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

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

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

Fig. 2.3.4 (a)

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

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,

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

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,

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



Fig. 2.3.5 (a)

		BC			
		$\bar{B} \bar{C}$	$\bar{B} C$	$B \bar{C}$	BC
A	0	1	0	1	0
	1	1	0	1	0
		$\bar{B} \bar{C}$	BC		

Fig. 2.3.5 (b)

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

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

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

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

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

Fig. 2.3.6 (a)

Step 2 : There are no isolated 1s.

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

$$Y = AB\overline{D} + \overline{A}D$$

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

Fig. 2.3.6 (c)

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.

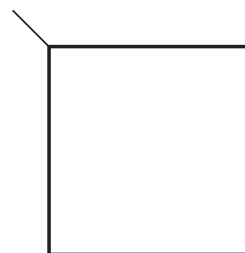


Fig. 2.3.7 (a)

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

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

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

Fig. 2.3.7 (b)

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

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

$$F = ABD + A\bar{B}\bar{D} + BCD + A\bar{B}D + \bar{B}C\bar{D}$$

Fig. 2.3.8

ii)

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

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

Fig. 2.3.9

Examples for Practice

Example 2.3.9 : Simplify following logical expression using Karnaugh maps

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

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

$$\text{Ans. : } \bar{A}\bar{B} + AB\bar{D} + \bar{B}CD$$

Example 2.3.12 : Simplify the following using K-map.

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

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

Example 2.3.13 : Solve the following using minimization technique

$$Z = f(A, B, C, D) = \sum (0, 2, 4, 7, 11, 13, 15)$$

SPPU : Dec.-09, Marks 5

$$\text{Ans. : } Z = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{C}\bar{D} + BCD + ABD + ACD$$

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 000 to 101. For remaining two conditions of input, output is not defined, hence these are called don't care conditions for this truth table.

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

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,

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

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 ABC 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 ABC 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

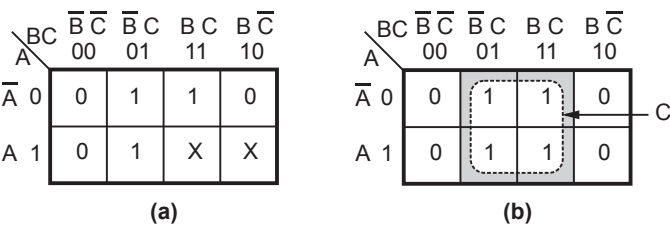
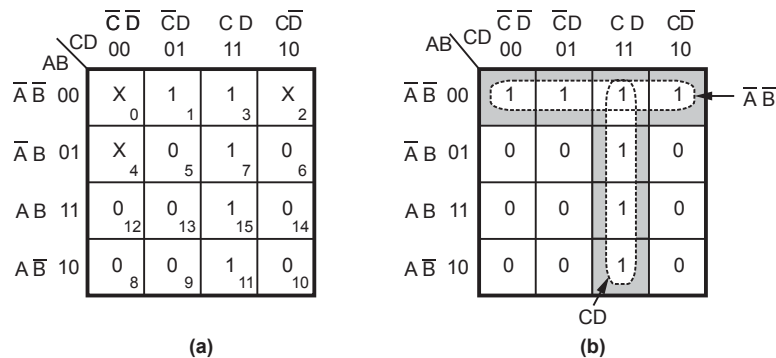


Fig. 2.3.10 Use of don't care conditions

Illustrative Examples**Example 2.3.14** Find the reduced SOP form of the following function.

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

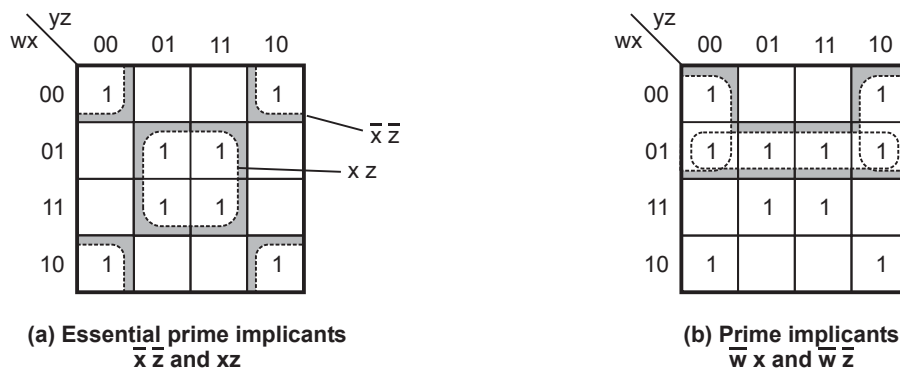
Solution :**Fig. 2.3.11**

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

$$f(A, B, C, D) = \underbrace{\bar{A}\bar{B}}_{\text{Group 1}} + \underbrace{CD}_{\text{Group 2}}$$

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

**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. $f(a,b,c,d) = \Sigma (0,2,4,5,6,8,10,15) + \Sigma \phi(7,13,14)$

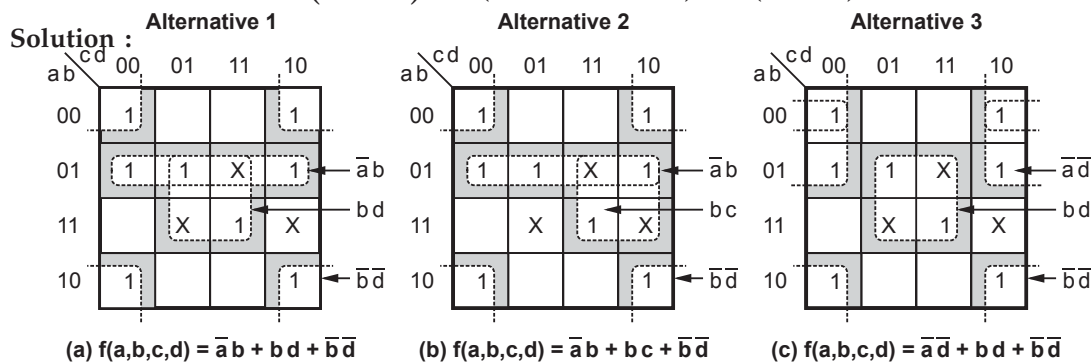


Fig. 2.3.13

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

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

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

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+C	$\bar{B}+\bar{C}$	$\bar{B}+\bar{C}$
A	0	0 ₀	0 ₁	0 ₃	
	1	0 ₄		0 ₇	

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+C̄	B̄+C̄	B̄+C
A	00	01	11	10
A 0	0	0	0	
Ā 1	0		0	

Fig. 2.4.1 (b)

BC				
	B+C	B+C̄	B̄+C̄	B̄+C
A	00	01	11	10
A 0	0	0	0	
Ā 1	0		0	

Group 3 → A + C̄
Group 2 → B̄ + C̄
Group 1 → B + C

Fig. 2.4.1 (c)

Example 2.4.2

Minimize the following expression in the POS form

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

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

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

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+D̄	C̄+D̄	C̄+D
AB	00	01	11	10
A+B 00	0			
A+B̄ 01				
Ā+B̄ 11				
Ā+B 10				

Fig. 2.4.2 (a)

Step 3 : 0 in the cell 0 is adjacent only to 0 in the cell 8. This pair is combined and referred to as group 1.

CD				
	C+D	C+D̄	C̄+D̄	C̄+D
AB	00	01	11	10
A+B 00	0			
A+B̄ 01				
Ā+B̄ 11				
Ā+B 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.

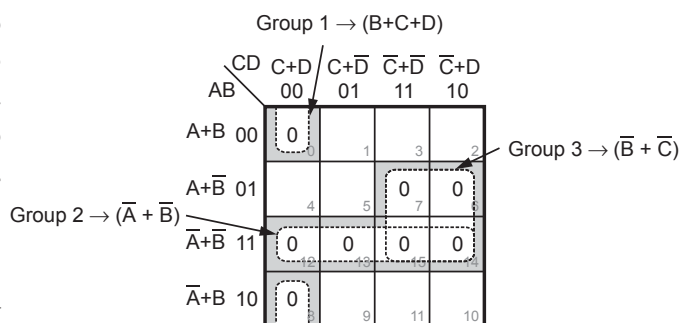


Fig. 2.4.2 (c)

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.

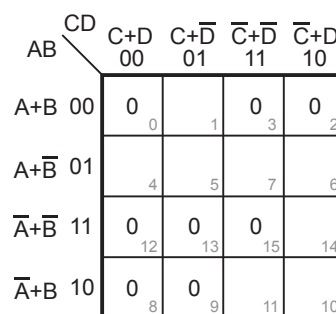


Fig. 2.4.3 (a)

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.

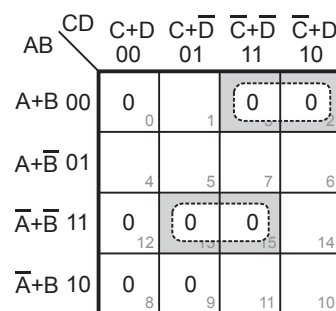


Fig. 2.4.3 (b)

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.

Step 6 : In group 1 and in group 4 variable C is eliminated. In group 2 variable D is eliminated and in group 3 variables B and D are eliminated. Therefore, we get simplified expression in POS form as,

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

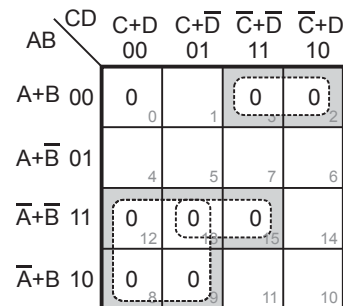


Fig. 2.4.3 (c)

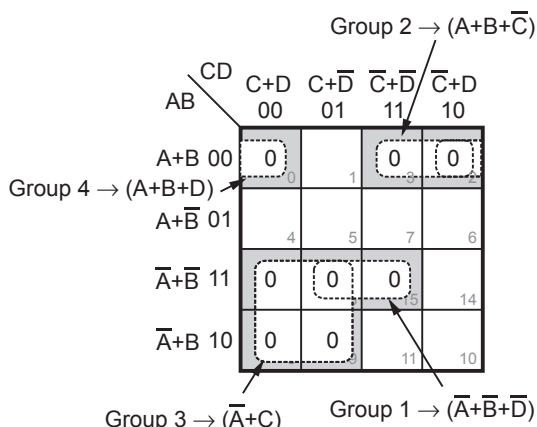


Fig. 2.4.3 (d)

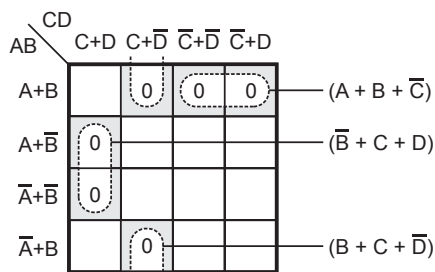
Example 2.4.4 Using K-map convert the following standard POS expression into a minimum POS expression, a standard SOP expression and minimum SOP expression

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

SPPU : Dec.-15, Marks 6

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

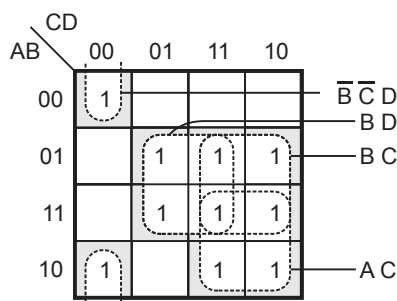
Minimum DOS Expression using K-map



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

Standard SOP Expression

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

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

1. 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 :

1. Group should not include any cell containing a zero.
2. The number of cells in a group must be a power of 2, such as 1, 2, 4, 8 or 16.
3. Group may be horizontal, vertical but not diagonal.
4. 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, $AB\bar{C}$ and $A\bar{B}\bar{C}$ 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 00-0. 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

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

Example 2.7.2

Minimize the expression using Quine McCluskey method.

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

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.

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 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.

Step 5 : List the prime implicants.

Prime implicants	Binary representation
$\bar{A} \bar{B} C \bar{D}$	2
$A \bar{C} D$	9, 13
$B \bar{C}$	4, 5, 12, 13

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	0 - 0 1
$\bar{A} \bar{B} C$ 2, 3	0 0 1 -
$\bar{A} B \bar{C}$ 4, 5	0 1 0 -
$B \bar{C} \bar{D}$ 4, 12	- 1 0 0
$A \bar{B} \bar{C}$ 8, 9	1 0 0 -
$A \bar{C} \bar{D}$ 8, 12	1 - 0 0
$A B \bar{D}$ 12, 14	1 1 - 0
$A C D$ 11, 15	1 - 1 1
$A B C$ 14, 15	1 1 1 -
$\bar{B} D$ 1, 3, 9, 11	- 0 - 1

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						•	•				
$A\bar{C}\bar{D}$	8, 12						•			•		
$AB\bar{D}$	12, 14 ✓									⊙	⊙	
ACD	11, 15								•			•
ABC	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

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

Example 2.7.4

Simply the following function using Quine-McCuskey minimization technique :

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

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 ✓	0	0	0	0
m ₁ ✓	0	0	0	1
m ₂ ✓	0	0	1	0
m ₈ ✓	1	0	0	0
m ₃ ✓	0	0	1	1
m ₅ ✓	0	1	0	1
m ₉ ✓	1	0	0	1
m ₇ ✓	0	1	1	1
m ₁₁ ✓	1	0	1	1
m ₁₄	1	1	1	0

Min terms	Binary Representation			
0, 1 ✓	0	0	0	-
0, 2 ✓	0	0	-	0
0, 8 ✓	-	0	0	0
1, 3 ✓	0	0	1	-
1, 5 ✓	0	-	0	1
1, 9 ✓	-	0	0	1
2, 3 ✓	0	0	1	-
8, 9 ✓	-	0	0	1
3, 7 ✓	0	-	1	1
3, 11 ✓	-	0	1	1
5, 7 ✓	0	1	-	1
9, 11 ✓	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 - -	$\overline{A}\overline{B}$	⊙	⊙	⊙	⊙						
✓	0,1,8,9	- 0 0 -	$\overline{B}\overline{C}$	⊙	⊙					⊙	⊙		
✓	1,3,5,7	0 - - 1	$\overline{A}D$		⊙		⊙	⊙	⊙				
✓	1,3,9,11	- 0 - 1	$\overline{B}D$		⊙		⊙				⊙	⊙	
✓	14	1 1 1 0	$AB\overline{C}\overline{D}$										⊙

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

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 – 0
2	0 1 0	1, 5	– 0 1
5	1 0 1		– 1 0
6	1 1 0	5, 7	1 – 1
7	1 1 1		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 implicates of any function.

Examples for Practice

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

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

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

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

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.

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 :

$Z = f(A,B,C,D) = \sum (0, 3, 8, 9, 10, 12, 15)$

SPPU : May-10, Marks 10

**Ans. : $PI = \overline{A} \overline{B} C D + A B C D + \overline{B} C \overline{D} + A \overline{C} \overline{D} + A \overline{B} \overline{C} + A B \overline{D}$
 $EPI = \overline{A} \overline{B} C D + A B C D + \overline{B} C \overline{D} + A \overline{C} \overline{D} + A \overline{B} \overline{C} + A B \overline{D}$**

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

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

SPPU : May-14, Marks 6

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

Reivew 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

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

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.

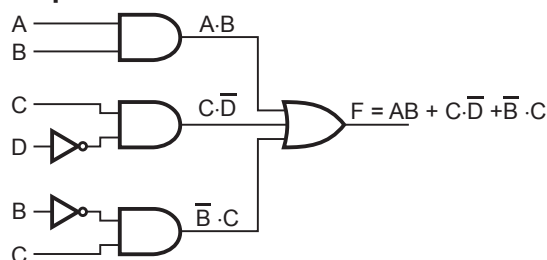


Fig. 2.8.1 Implementation of SOP Boolean expression

2.8.2 Implementation of POS Boolean Expression

Consider the Boolean expression

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

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

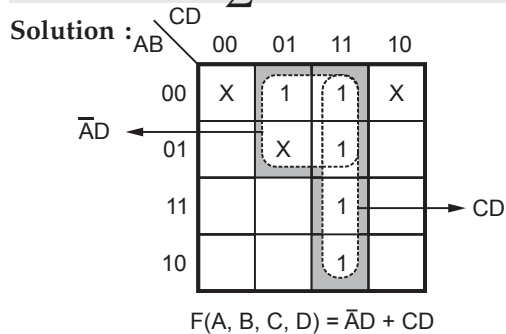


Fig. 2.8.4

Logic diagram

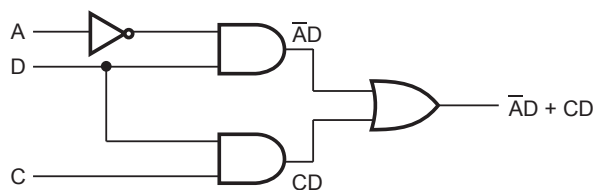


Fig. 2.8.5

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 :

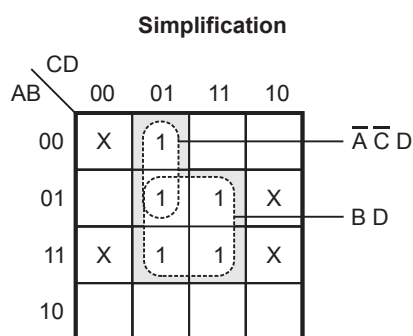


Fig. 2.8.6

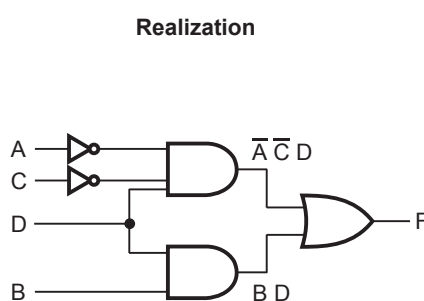


Fig. 2.8.7

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

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

SPPU : May-16, Marks 4

Solution :

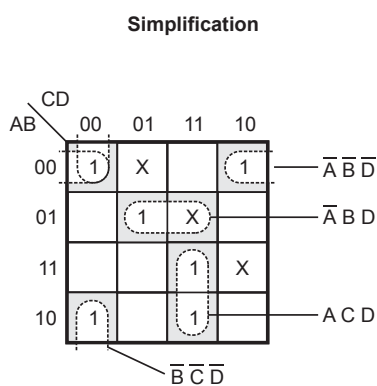


Fig. 2.8.8

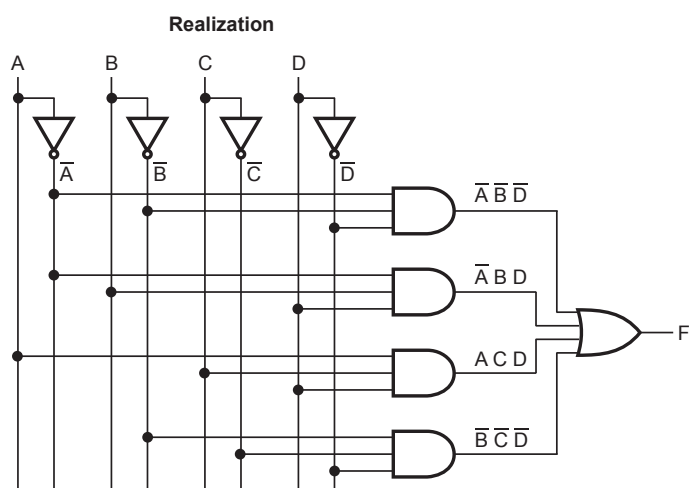


Fig. 2.8.9

2.9 Universal Gates

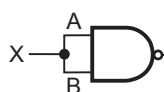
The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates.

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.



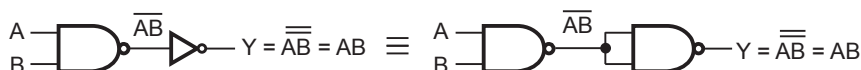
$$Y = \overline{AB} = \overline{XX} = \overline{X} + \overline{X} = \overline{X}$$

	A	B	\overline{AB}	
X = 0	0	0	1	Y = 1
	0	1	1	
	1	0	1	
X = 1	1	1	0	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. $\overline{\overline{AB}} = AB$. Fig. 2.9.2 shows the two input AND gate using NAND gates.

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

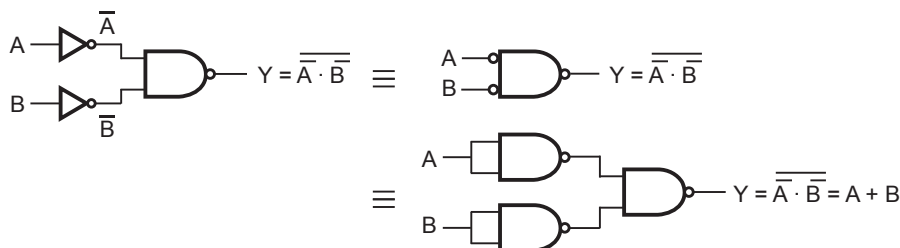
Table 2.9.1 Truth table**Fig. 2.9.2 AND function using NAND gates****OR Function :**

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

$$Y = A + B = \overline{\overline{A}} + \overline{\overline{B}} \quad [\overline{\overline{A}} = A]$$

$$= \overline{\overline{A} \cdot \overline{B}} \quad \text{DeMorgan's Theorem 1}$$

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	$A + B$	\equiv	A	B	$\overline{A \cdot B}$	$\overline{\overline{A \cdot B}}$
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

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

DeMorgan's Theorem 2

$$= \overline{\overline{\overline{A \cdot B}}}$$

$$[\overline{\overline{A}} = A]$$

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

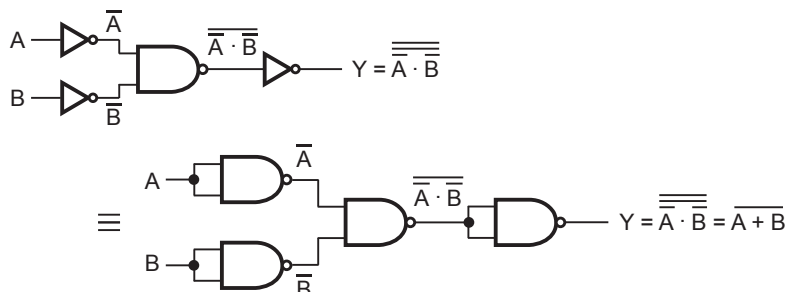


Fig. 2.9.4 NOR function using only NAND gates

A	B	$A + B$	\equiv	A	B	$\overline{A \cdot B}$	$\overline{\overline{A \cdot B}}$	$\overline{\overline{\overline{A \cdot B}}}$
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.

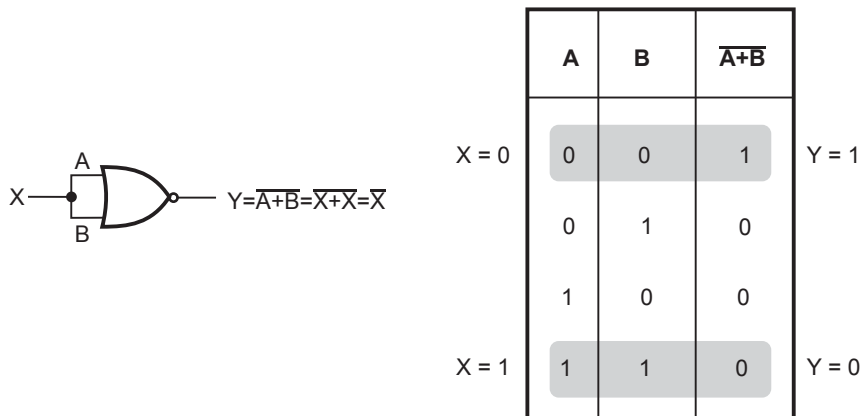


Fig. 2.9.5 NOT function using NOR gate

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.

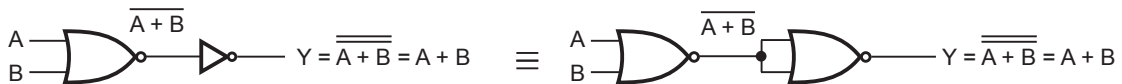


Fig. 2.9.6 OR function using NOR gates

A	B	A + B		A	B	$\overline{A+B}$	$\overline{\overline{A+B}}$
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 2.9.3 Truth table

AND Function :

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

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

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

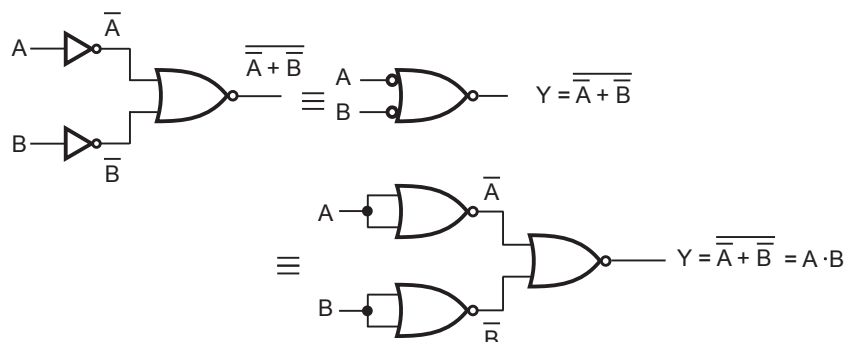


Fig. 2.9.7 AND function using NOR gates

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

A	B	$A \cdot B$	\equiv	A	B	$\overline{A + B}$	$\overline{\overline{A + B}}$
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

$$Y = \overline{A \cdot B} = \overline{A + B} \quad \text{DeMorgan's Theorem 1}$$

$$= \overline{\overline{\overline{A + B}}} \quad [\overline{\overline{A}} = A]$$

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

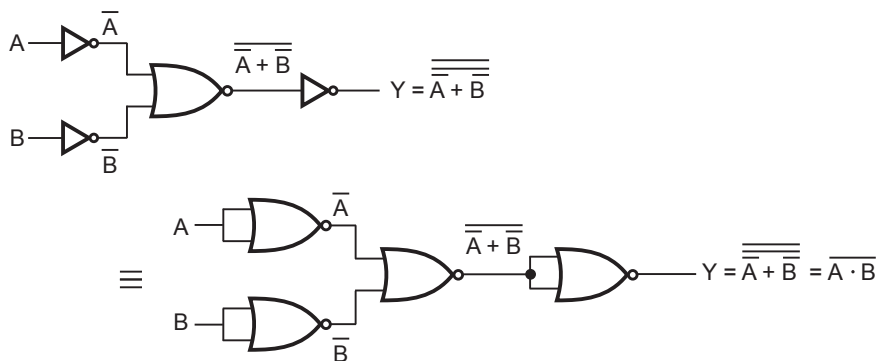


Fig. 2.9.8 NAND function using only NOR gates

A	B	$\overline{A \cdot B}$		A	B	$\overline{A+B}$	$\overline{\overline{A+B}}$	$\overline{\overline{\overline{A+B}}}$
0	0	1	\equiv	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).

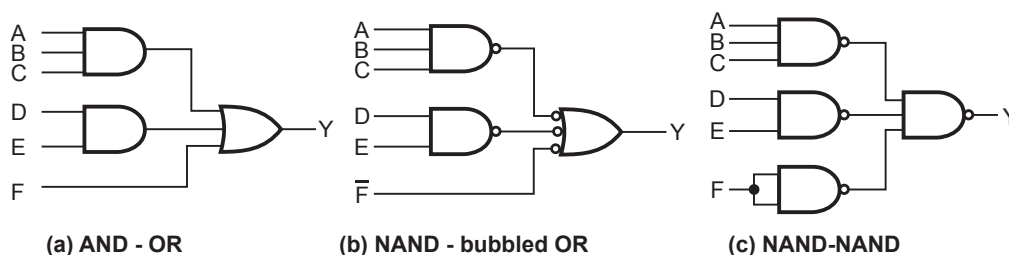


Fig. 2.10.1 NAND-NAND implementation

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.

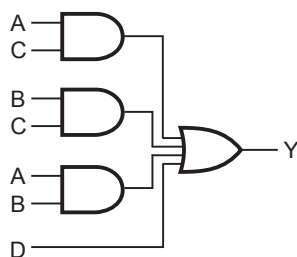


Fig. 2.10.2 (a)

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

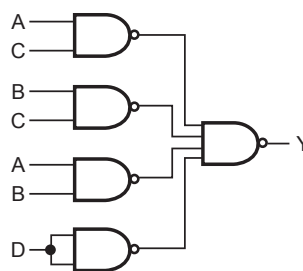


Fig. 2.10.2 (b)

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.

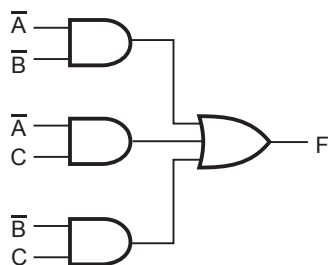


Fig. 2.10.3 (a)

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

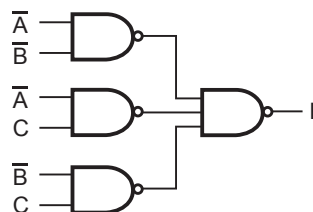


Fig. 2.10.3 (b)

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.

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

Fig. 2.10.4

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

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

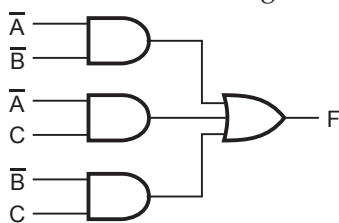


Fig. 2.10.5 (a)

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

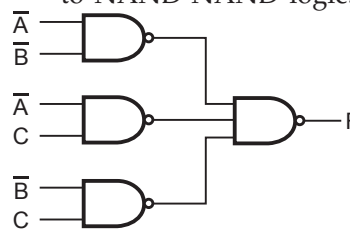


Fig. 2.10.5 (b)

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).

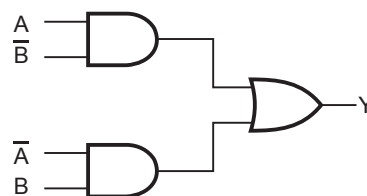


Fig. 2.10.6 (a)

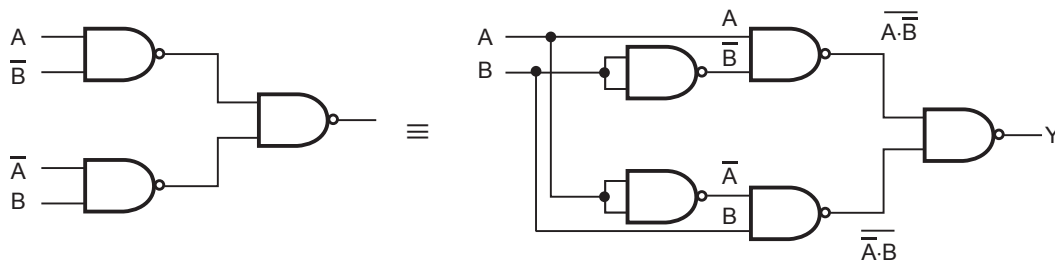


Fig. 2.10.6 (b)

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.

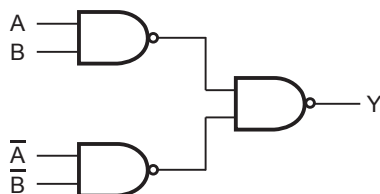


Fig. 2.10.7

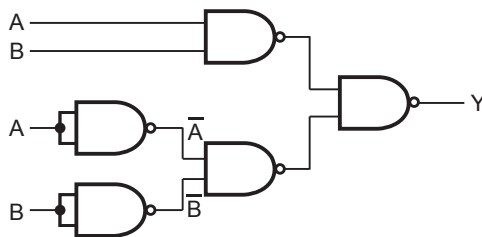


Fig. 2.10.7 (a)

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.

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

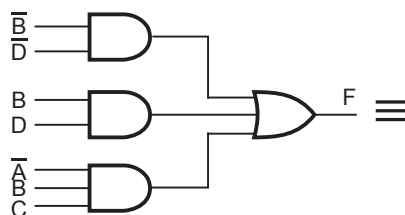
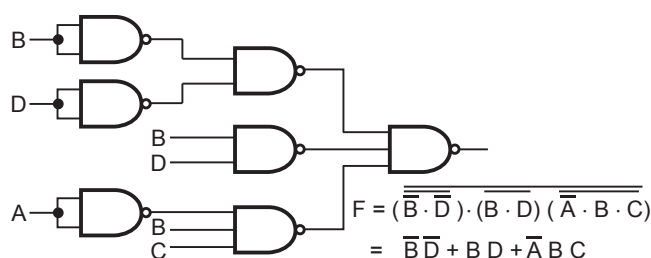
SPPU : May-07, Marks 8

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

K-map simplification :

CD \ AB	00	01	11	10
00	1 ₀	0 ₁	0 ₃	1 ₂
01	0 ₄	1 ₅	1 ₇	1 ₆
11	0 ₁₂	1 ₁₃	X ₁₅	0 ₁₄
10	X ₈	0 ₉	0 ₁₁	X ₁₀

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

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 :

$$F(A, B, C, D) = (\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D} + ABCD)$$

SPPU : May-08, Marks 8**Solution :**

$$F(A, B, C, D) = (\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D} + ABCD)$$

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

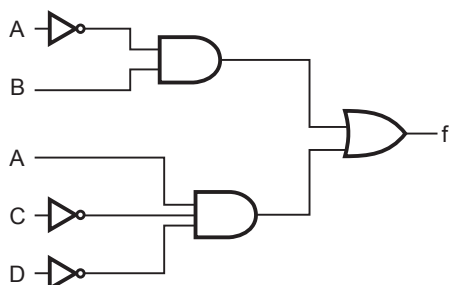
K-map simplification :

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

Fig. 2.10.9

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

Implementation using basic gates



Implementation using only NAND gates

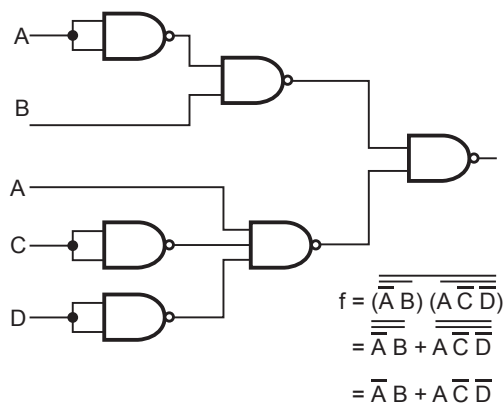


Fig. 2.10.9 (a)

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

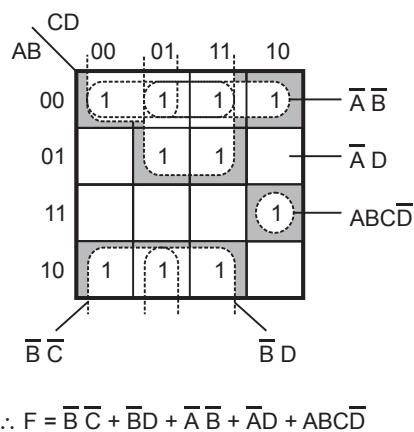


Fig. 2.10.10

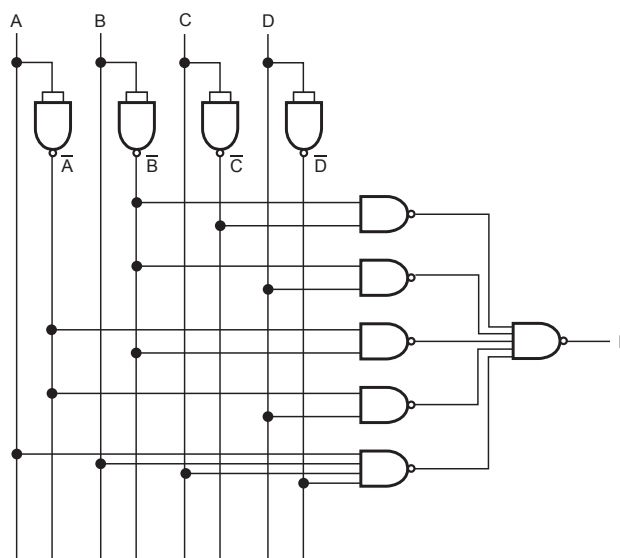


Fig. 2.10.11

Example 2.10.9 Implement each expression with NAND logic 1 :

- i) $ABC + DE$ ii) $ABC + D' + E'$.

SPPU : Dec.-15, Marks 4

Solution : i) $ABC + DE$

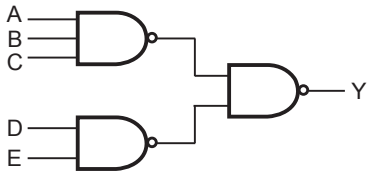


Fig. 2.10.12

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

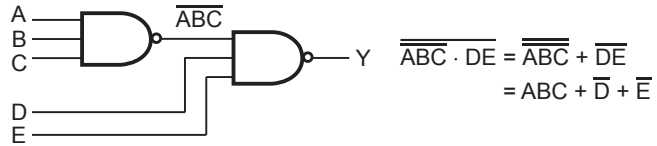


Fig. 2.10.13

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

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

SPPU : Dec.-16, Marks 4

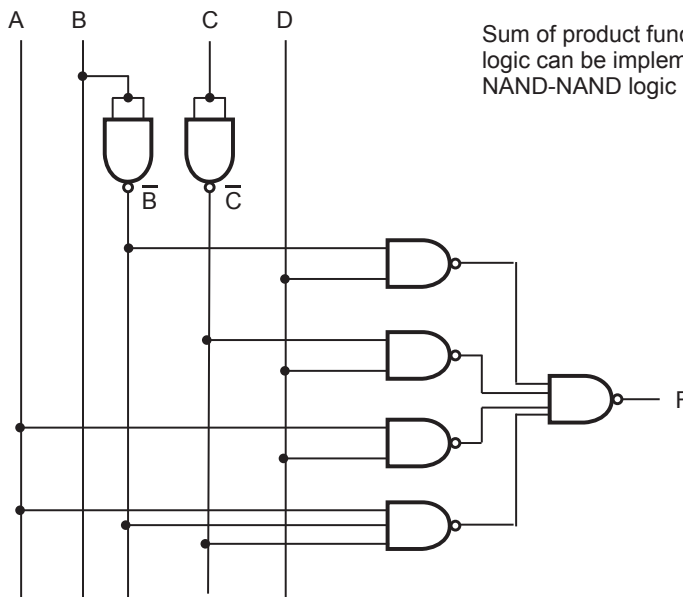
Solution :

K-map simplification

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

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

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 :

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

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

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

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

$$[\text{Ans. : } y = \overline{\overline{B}\overline{C}} + \overline{\overline{A}\overline{B}} + \overline{\overline{A}\overline{D}}] \quad \text{SPPU : May-06, Marks 8}$$

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

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

$$[\text{Ans. : } F = \overline{\overline{A}\overline{B}} \cdot \overline{\overline{A}\overline{C}\overline{D}}] \quad \text{SPPU : Dec.-06, Marks 6}$$

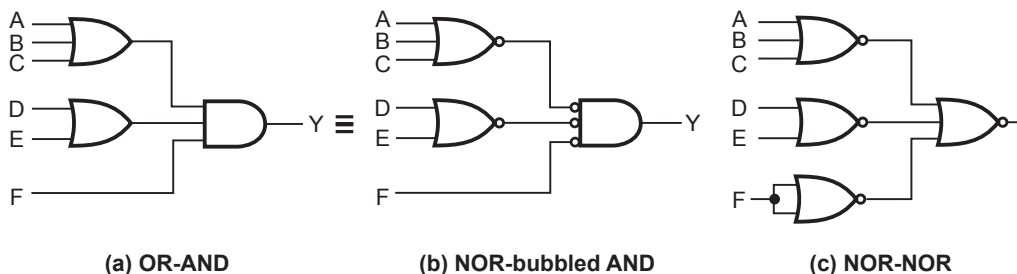
2.11 NOR-NOR Implementation**SPPU : Dec.-05,06,07**

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 bubbled 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.

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

Solution : Step 1 : Simplify the given Boolean function.

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

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

Fig. 2.11.2

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

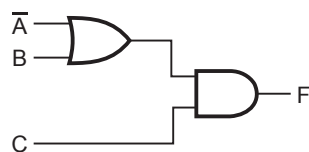


Fig. 2.11.3 (a)

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

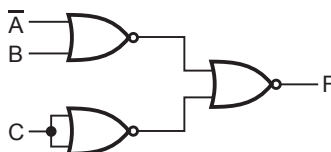


Fig. 2.11.4

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 :

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

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

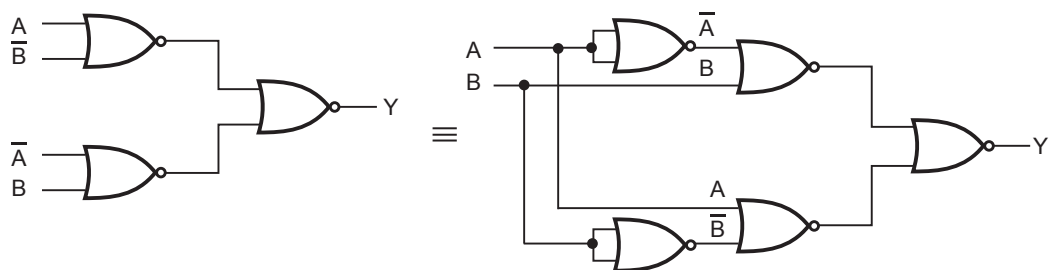


Fig. 2.11.5 (b)

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

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

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

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

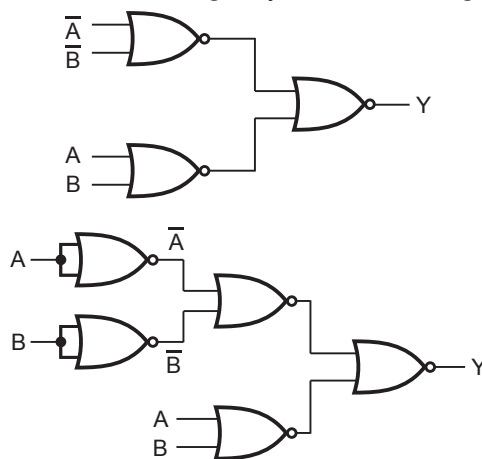


Fig. 2.11.6

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 :

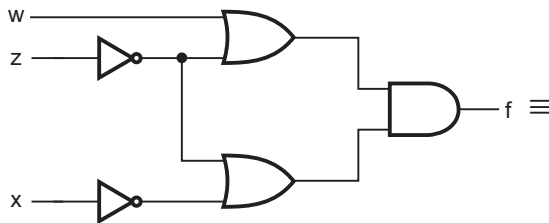
$$f(w, x, y, z) = (w + \bar{z})(\bar{x} + \bar{z})$$

Implementation

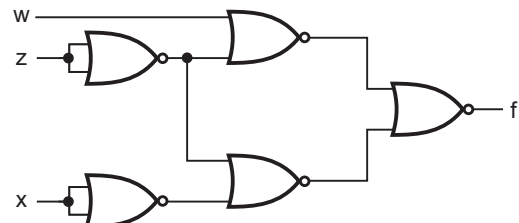
The OR-AND logic can be directly implemented using NOR-NOR logic as below,

	yz	y+z	y+z̄	ȳ+z	ȳ+z̄
wx	00	1	0	0	1
w+x	01	1	0	0	1
w+x̄	11	1	0	0	1
w̄+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 :

	CD	C+D	C+D̄	C̄+D	C̄+D̄
AB					
A+B	0	x	0	x	
A+B̄	1	0	1	0	
Ā+B	x	1	1	1	
Ā+B̄	1	1	x	0	

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 :

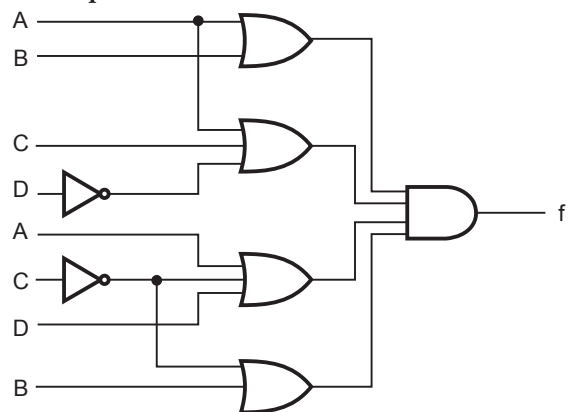


Fig. 2.11.8 (a)

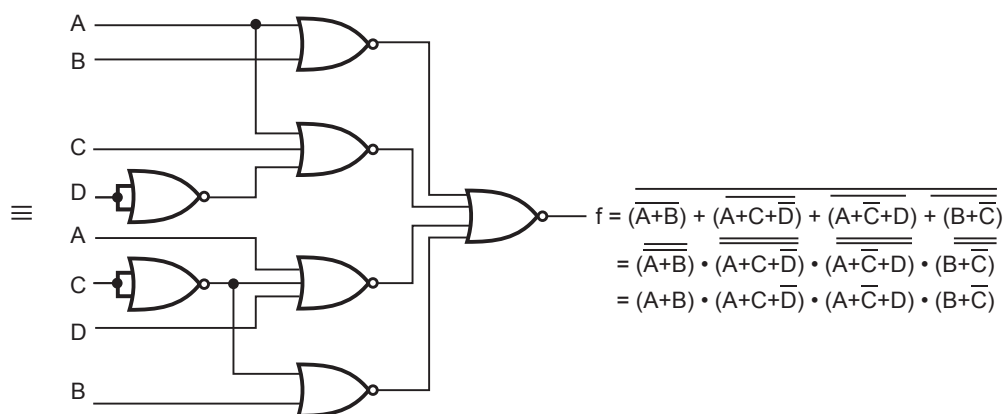


Fig. 2.11.8 (b)

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

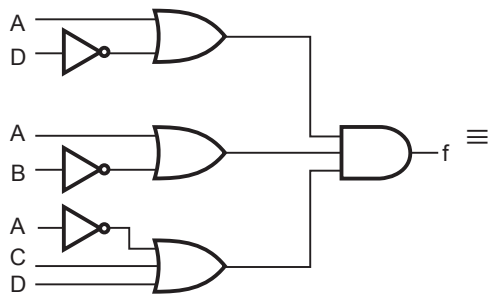
Solution : K-map simplification :

		CD				
AB		C+D	C+D̄	C̄+D̄	C̄+D	
	A + B	1 0	0 1	x 3	1 2	(A+D̄)
	A + B̄	0 4	0 5	0 7	0 6	(A+B̄)
	Ā + B̄	0 12	1 13	1 15	x 14	
	Ā + B	0 8	x 9	x 11	1 10	(Ā+C+D)

Fig. 2.11.9

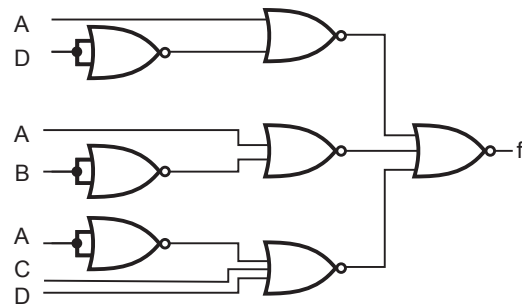
$$\therefore f(ABCD) = (A + \overline{D})(A + \overline{B})(\overline{A} + C + D)$$

Implementation using basic gates



(a)

Implementation using only NOR gates



(b)

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

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

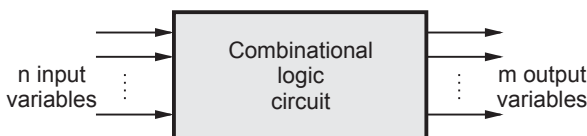


Fig. 3.1.1 Block diagram of a combinational circuit

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.

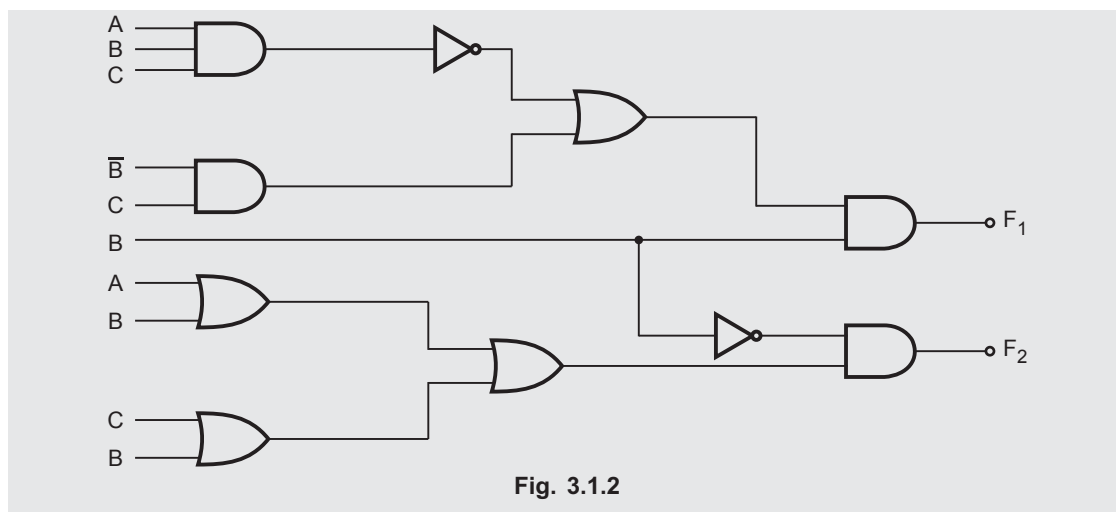


Fig. 3.1.2

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

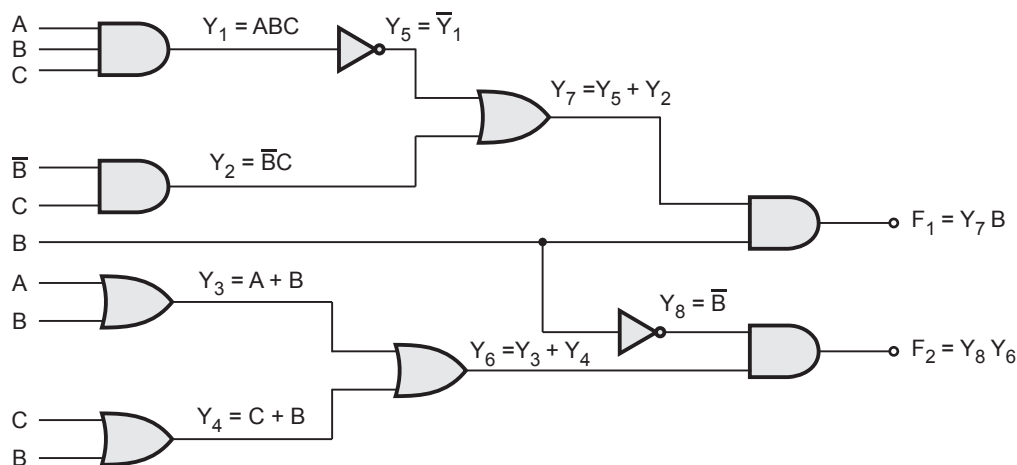


Fig. 3.1.3

$$F_1 = Y_5 B = (\bar{Y}_1 + Y_2) B = (\overline{ABC} + \bar{B}C) B = [(\bar{A} + \bar{B} + \bar{C}) + \bar{B}C] B = (\bar{A} + \bar{C}) B$$

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

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)

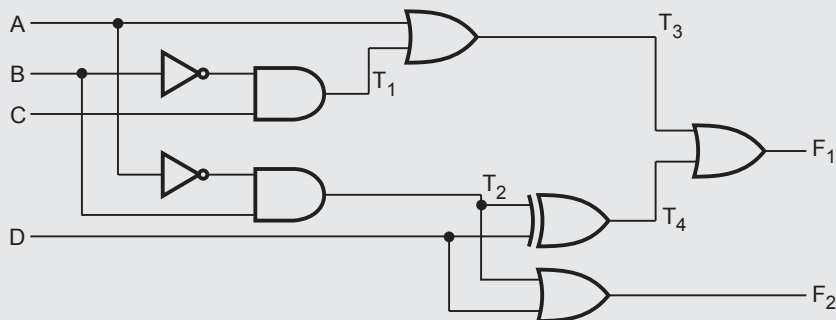


Fig. 3.1.4

Solution : i) $T_1 = \overline{B}C$ $T_2 = \overline{A}B$ $T_3 = A + \overline{B}C$

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

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

$$F_2 = \overline{A}B + D$$

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

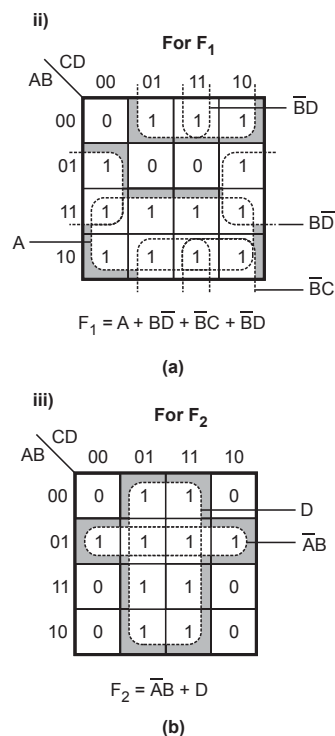


Fig 3.1.5

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

Step 2 : Obtain simplified Boolean expression.

Now we obtain the simplified Boolean expression for output variable Y using K-map simplification.

$$\therefore Y = AC + BC + AB$$

Step 3 : Draw logic diagram.

In this chapter we are going to study various combinational circuits using above illustrated design method.

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

Fig. 3.1.6 K-map simplification

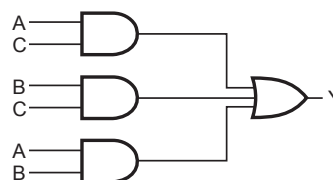


Fig. 3.1.7 Logic diagram

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

1. What is a combinational logic ?
2. Explain the analysis procedure for combinational circuits.
3. 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

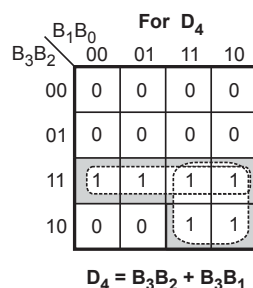
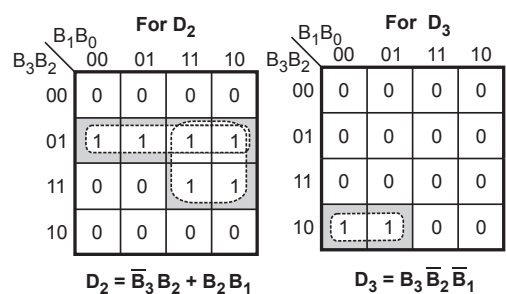
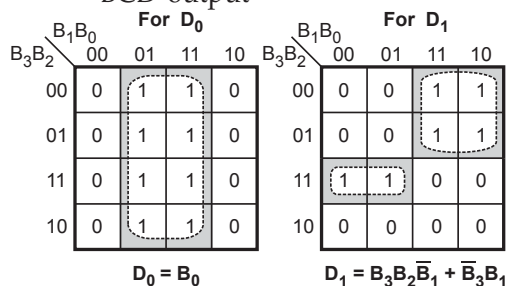
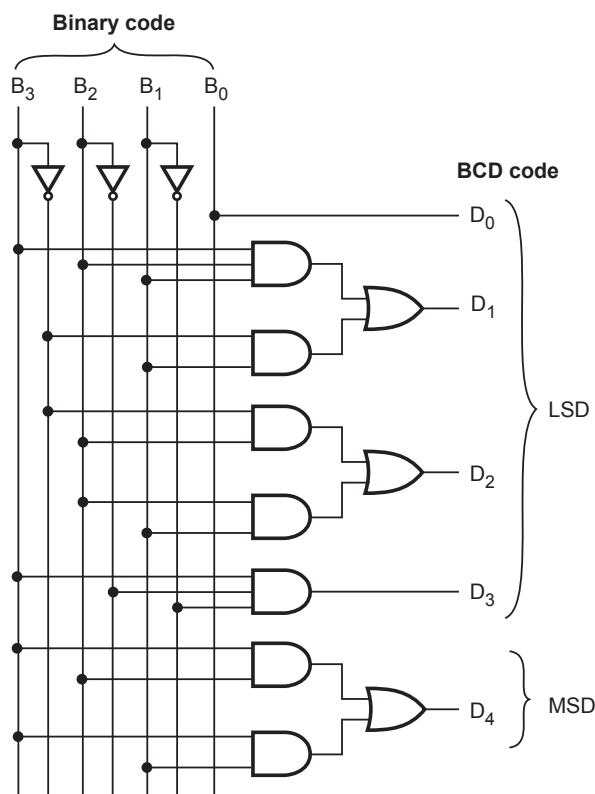


Fig. 3.2.1

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 : $D_3 D_2 D_1 D_0$ (D_0 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.

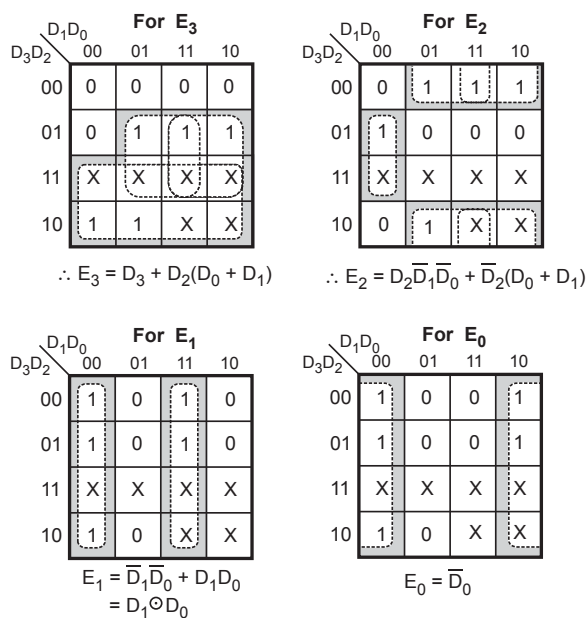


Fig. 3.2.3

Step 3 : Realization of code converter.

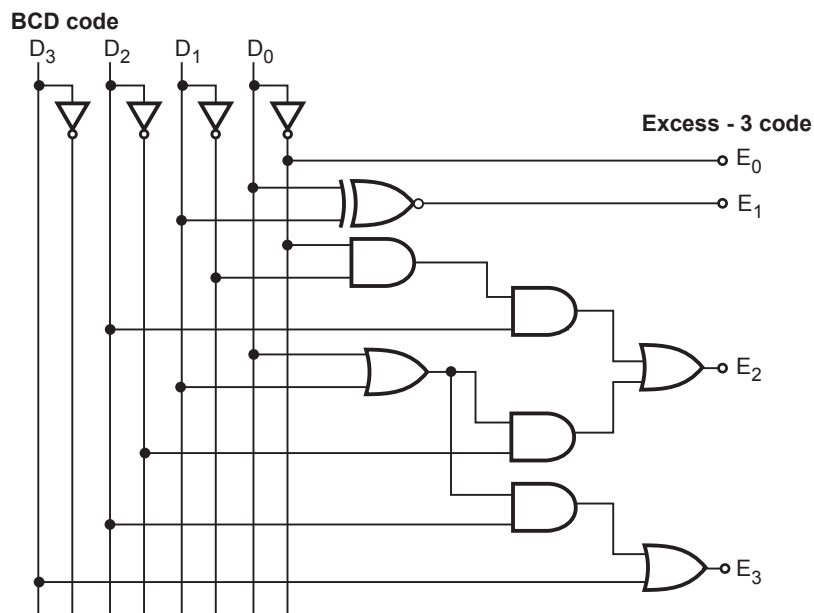


Fig. 3.2.4 BCD to excess-3 code converter

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

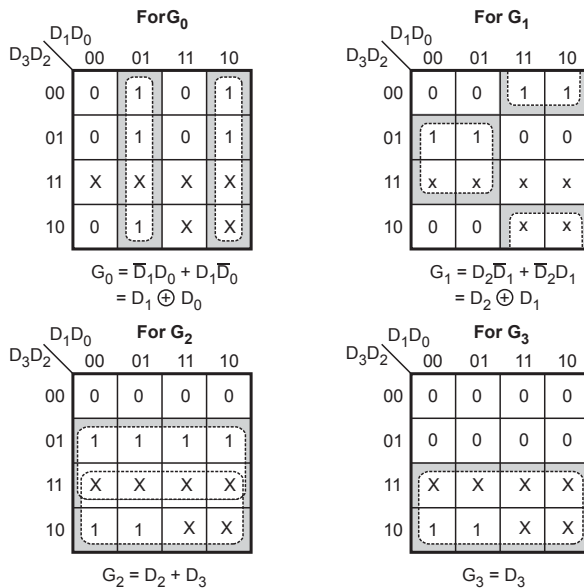


Fig. 3.2.5

Step 3 : Realization of code converter

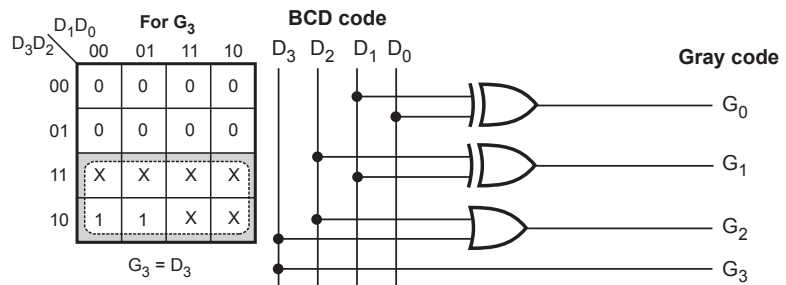


Fig. 3.2.6 BCD to gray code converter

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

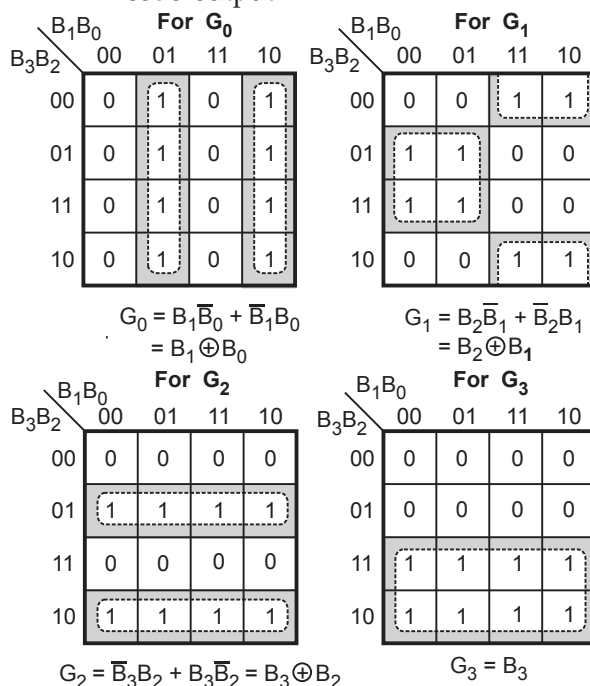


Fig. 3.2.7

Step 3 : Realization of code converter using XOR-gates

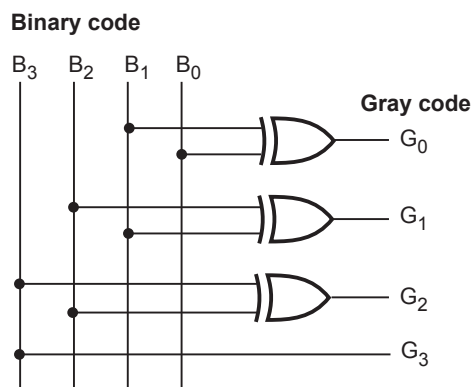


Fig. 3.2.8 Binary to gray code converter

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.

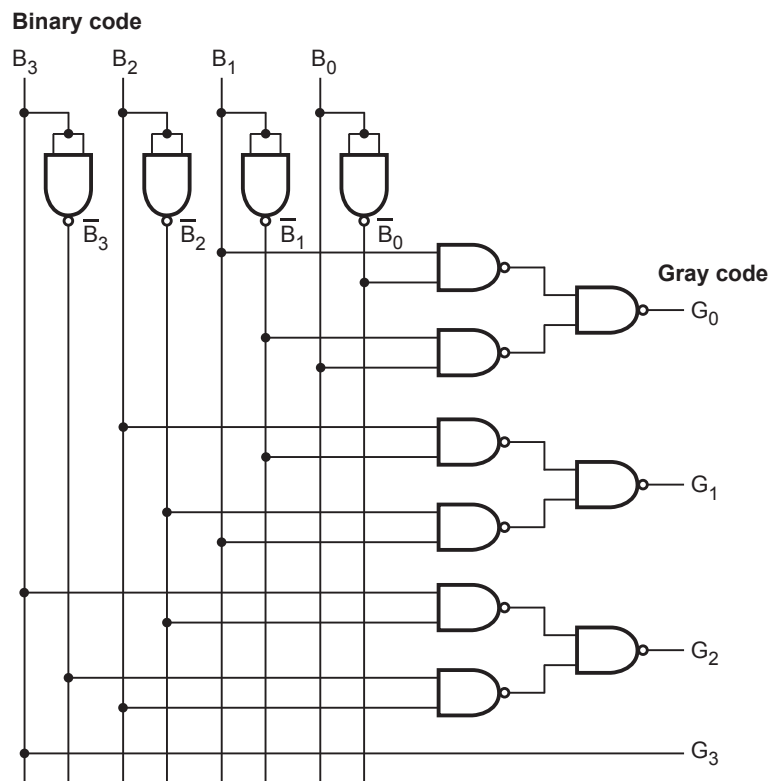


Fig. 3.2.9 Binary to Gray code converter

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

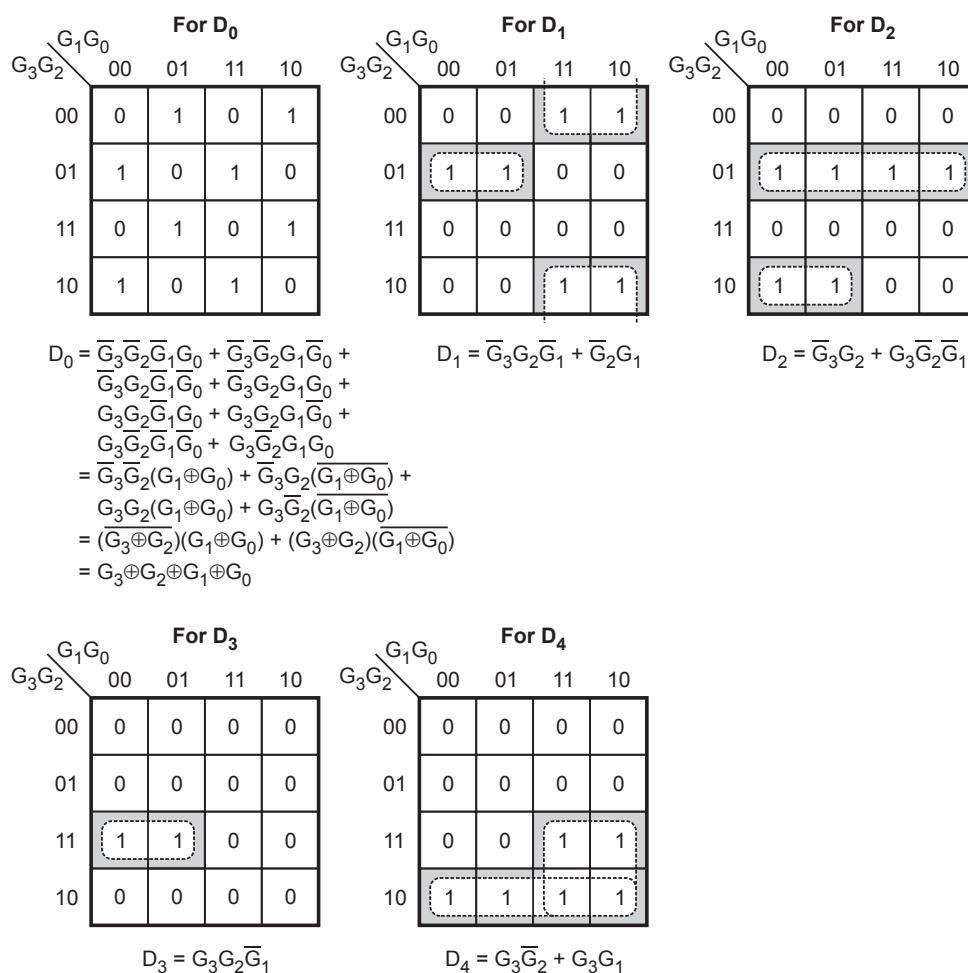


Fig. 3.2.10

Logic Diagram

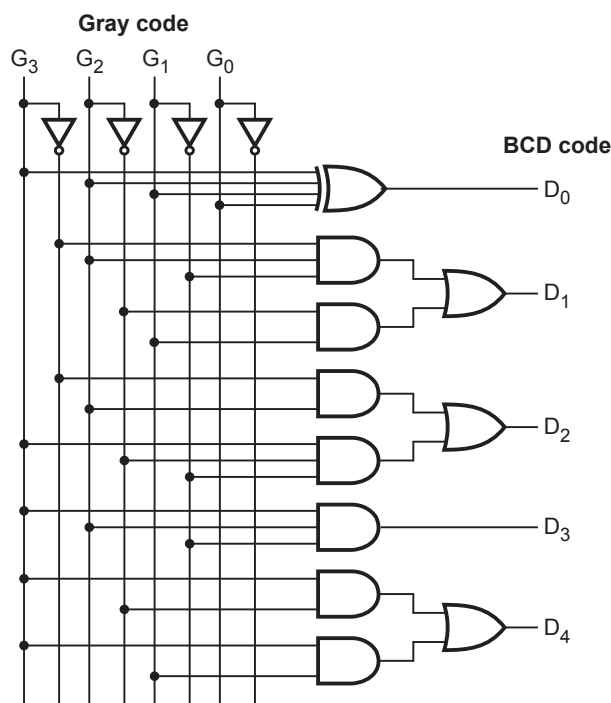


Fig. 3.2.11

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

E_1E_0	00	01	11	10
E_2				
0	X	X	0	X
1	0	0	1	0

$$B_2 = E_2 E_1 E_0$$

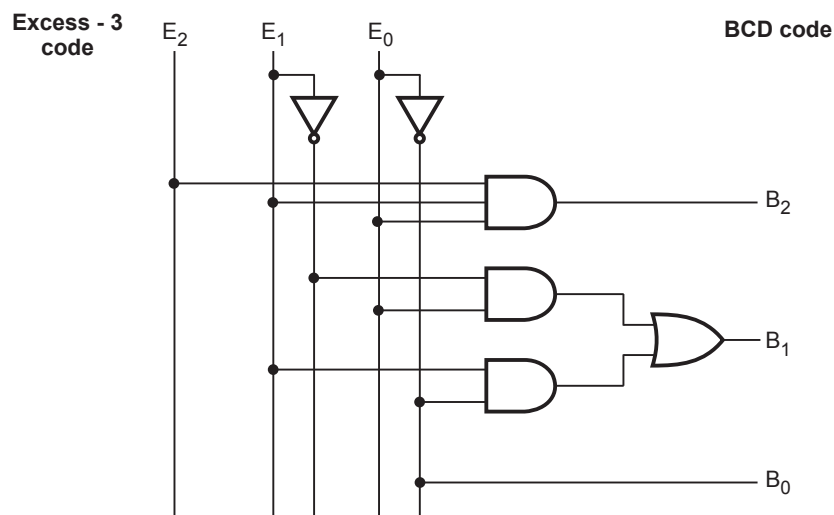
E_1E_0	00	01	11	10
E_2				
0	X	X	0	X
1	0	1	0	1

$$B_1 = \bar{E}_1 E_0 + E_1 \bar{E}_0$$

E_1E_0	00	01	11	10
E_2				
0	X	X	0	X
1	1	0	0	1

$$B_0 = \bar{E}_0$$

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.

$$\begin{aligned} \text{(Ans. : } D_0 &= \bar{E}_0, & D_1 &= E_1 \oplus E_0 \\ D_2 &= \bar{E}_2 \bar{E}_1 + E_2 E_1 E_0 + \bar{E}_2 \bar{E}_0 & D_3 &= E_3 E_2 + E_3 E_1 E_0) \end{aligned}$$

Example 3.2.8 Design a logic circuit to convert gray code to binary code.

$$\begin{aligned} \text{(Ans. : } B_0 &= G_3 \oplus G_2 \oplus G_1 \oplus G_0 & B_1 &= G_3 \oplus G_2 \oplus G_1 \\ B_2 &= G_3 \oplus G_2 & B_3 &= G_3) \end{aligned}$$

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

Table 3.3.1 Truth table for half-adder

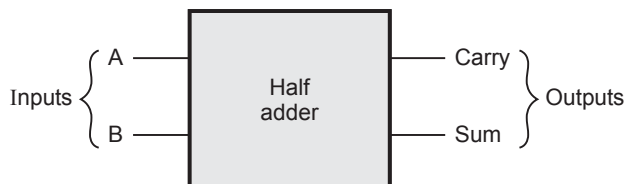


Fig. 3.3.1 Block schematic of half-adder

K-map simplification for carry and sum

For carry

	B	0	1
A	0	0	0
1	0	1	1

$$\text{Carry} = AB$$

For sum

	B	0	1
A	0	0	1
1	1	1	0

$$\begin{aligned} \text{Sum} &= A\bar{B} + \bar{A}B \\ &= A \oplus B \end{aligned}$$

Fig. 3.3.2 Maps for half-adder

Logic diagram

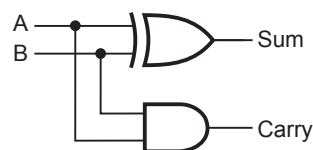


Fig. 3.3.3 Logic diagram for half-adder

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 :

$$\begin{aligned}\text{Sum} &= \overline{A}\overline{B} + \overline{A}B \\ &= \overline{A}\overline{\overline{B}} + \overline{A}\overline{B} \\ &= \overline{A}\overline{B} \cdot \overline{A}B\end{aligned}\quad \begin{aligned}C_{\text{out}} &= AB \\ &= \overline{\overline{A}\overline{B}}\end{aligned}$$

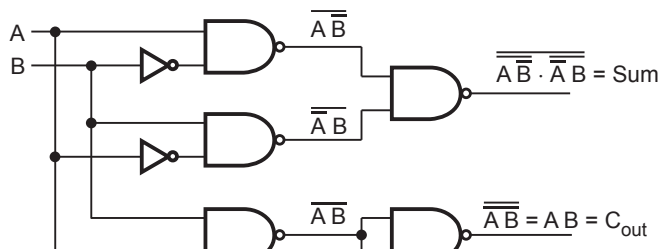


Fig. 3.3.4

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

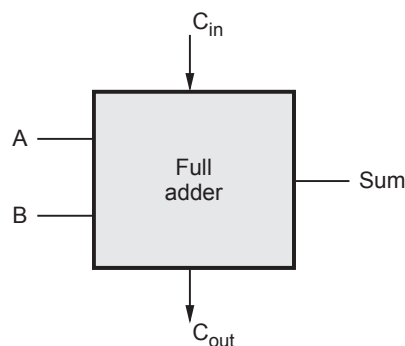


Fig. 3.3.5 Block schematic of full-adder

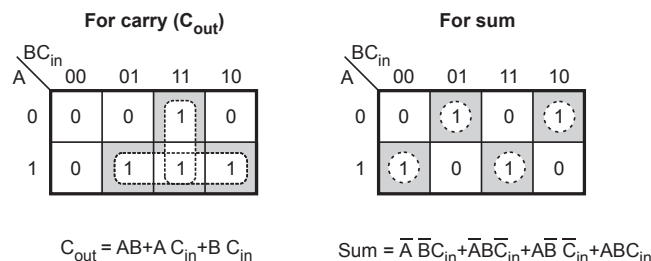
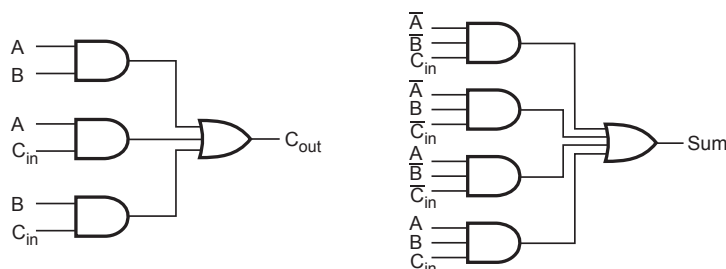
K-map simplification for carry and sum

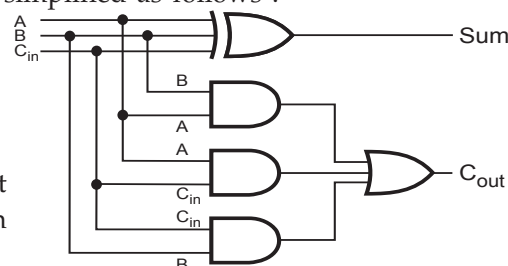
Fig. 3.3.6 Maps for full-adder

Logic diagram**Fig. 3.3.7 Sum of product implementation of full-adder**

The Boolean function for sum can be further simplified as follows :

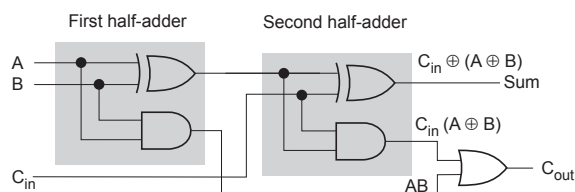
$$\begin{aligned}\text{Sum} &= \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} \bar{C}_{in} + A B C_{in} \\ &= C_{in} (\bar{A} \bar{B} + AB) + \bar{C}_{in} (\bar{A} B + A \bar{B}) \\ &= C_{in} (A \odot B) + \bar{C}_{in} (A \oplus B) \\ &= C_{in} (\overline{A \oplus B}) + \bar{C}_{in} (A \oplus B) = C_{in} \oplus (A \oplus B)\end{aligned}$$

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

$$\begin{aligned}C_{out} &= AB + A C_{in} + B C_{in} = AB + A C_{in} (B + \bar{B}) + B C_{in} (A + \bar{A}) \\ &= AB + ABC_{in} + A \bar{B} C_{in} + A B C_{in} + \bar{A} B C_{in} = AB (1 + C_{in} + C_{in}) + A \bar{B} C_{in} + \bar{A} B C_{in} \\ &= AB + A \bar{B} C_{in} + \bar{A} B C_{in} = AB + C_{in} (A \bar{B} + \bar{A} B) = AB + C_{in} (A \oplus B)\end{aligned}$$

**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) $\Sigma = 1, C_{out} = 1$	ii) $\Sigma = 0, C_{out} = 1$

SPPU : Dec.-15, Marks 2

3.4 Subtractors

SPPU : Dec.-08

The subtraction consists of four possible elementary operations, namely,

$$\begin{aligned} 0 - 0 &= 0 \\ 0 - 1 &= 1 \text{ with 1 borrow} \\ 1 - 0 &= 1 \\ 1 - 1 &= 0 \end{aligned}$$

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.

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

K-map simplification for half-subtractor

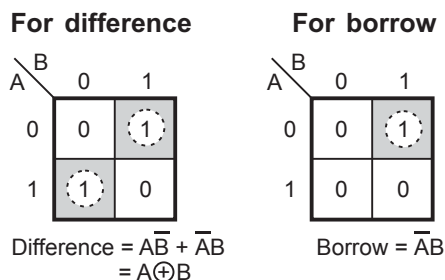


Fig. 3.4.1

Logic diagram

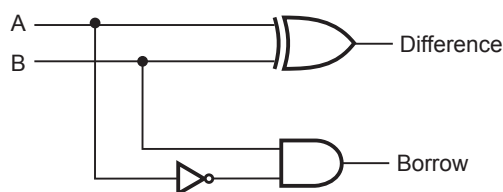


Fig. 3.4.2 Implementation of half-subtractor

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 :

$$\text{Difference} = A\bar{B} + \bar{A}B$$

$$\begin{aligned}
 &= \overline{\overline{A} \overline{B} + \overline{A} B} \\
 &= \overline{\overline{A} \overline{B}} \cdot \overline{\overline{A} B} \\
 \text{Borrow} &= \overline{\overline{A} B} \\
 &= \overline{\overline{A} B}
 \end{aligned}$$

Implementation :

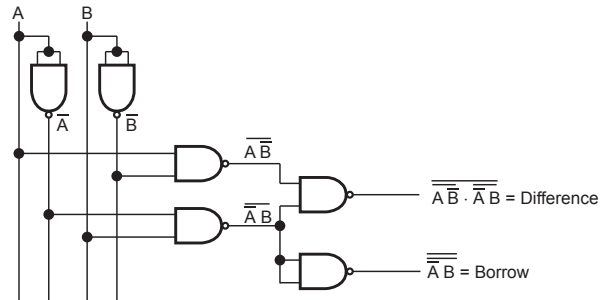


Fig. 3.4.3

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.

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

K-map simplification of D and B_{out}

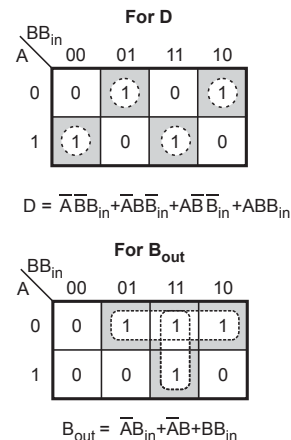


Fig. 3.4.4 Maps for full-subtractor

Logic diagram

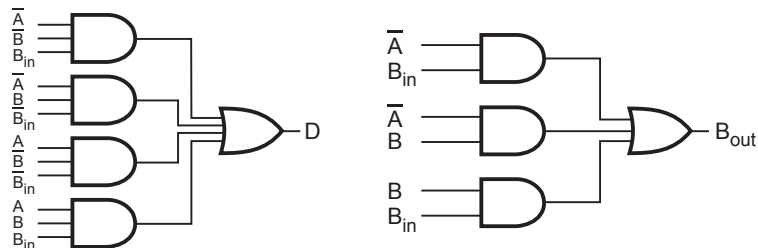


Fig. 3.4.5 Sum of product implementation of full-subtractor

The Boolean function for D (difference) can be further simplified as follows :

$$\begin{aligned} D &= \bar{A} \bar{B} B_{in} + \bar{A} B \bar{B}_{in} + A \bar{B} \bar{B}_{in} + A B B_{in} \\ &= B_{in} (\bar{A} \bar{B} + AB) + \bar{B}_{in} (\bar{A} B + A \bar{B}) = B_{in} (A \odot B) + \bar{B}_{in} (A \oplus B) \\ &= B_{in} (A \oplus B) + \bar{B}_{in} (A \oplus B) = B_{in} \oplus (A \oplus B) \end{aligned}$$

With this simplified Boolean function circuit for full-subtractor can be implemented as shown in the Fig. 3.4.6

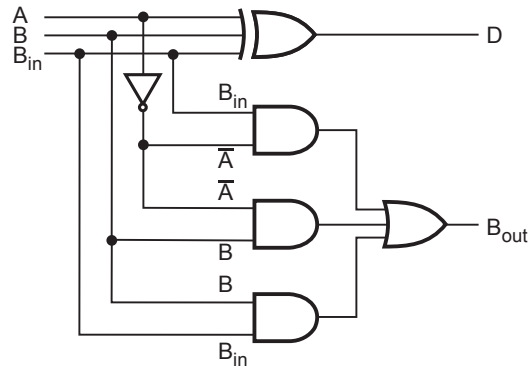


Fig. 3.4.6 Implementation of full-subtractor

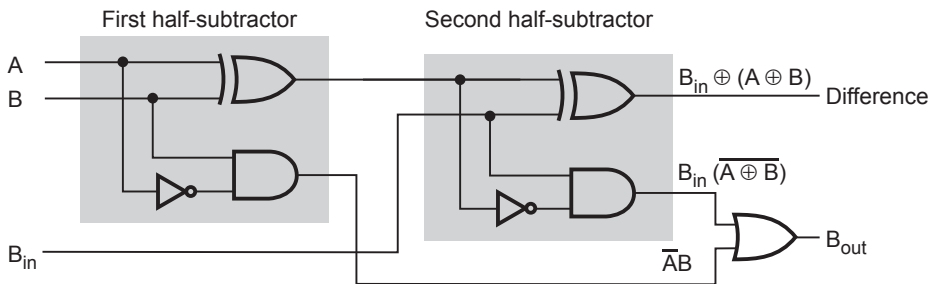


Fig. 3.4.7 Implementation of a full-subtractor with two half-subtractors and an OR gate

The borrow output for circuit shown in Fig. 3.4.7 can be given as,

$$\begin{aligned} B_{out} &= \bar{A} B_{in} + \bar{A} B + B B_{in} \\ &= \bar{A} B_{in} (B + \bar{B}) + \bar{A} B + B B_{in} (A + \bar{A}) \\ &= \bar{A} B B_{in} + \bar{A} \bar{B} B_{in} + \bar{A} B + AB B_{in} + \bar{A} B B_{in} \\ &= \bar{A} B (B_{in} + 1 + B_{in}) + \bar{A} \bar{B} B_{in} + AB B_{in} = \bar{A} B + \bar{A} \bar{B} B_{in} + AB B_{in} \\ &= \bar{A} B + B_{in} (\bar{A} \bar{B} + AB) = \bar{A} B + B_{in} (A \oplus B) \end{aligned}$$

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

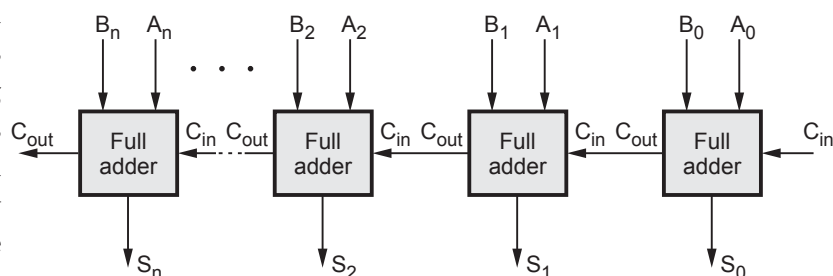


Fig. 3.5.1 Block diagram of n-bit parallel adder

must be employed. 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.

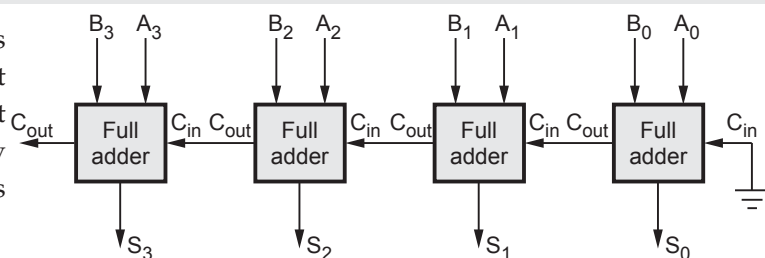


Fig. 3.5.2 Block diagram of 4-bit full-adder

Review Question

1. 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

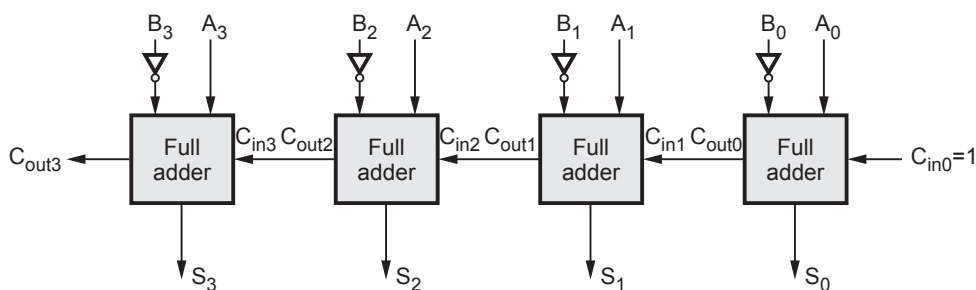


Fig. 3.6.1 4-bit parallel subtractor

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,

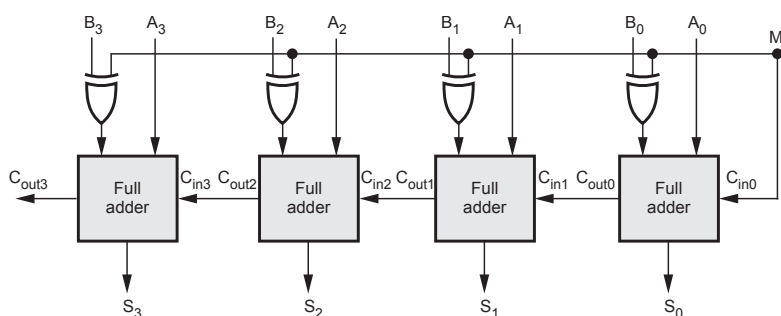


Fig. 3.7.1 4-bit adder-subtractor

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 $C_{in\ 0} = 1$. 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**.

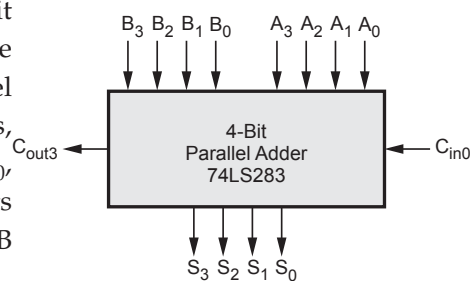
One method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition**. This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated.

Review Question

1. Draw and explain the block diagram of 4-bit parallel adder/subtractor.

3.8 Four-bit Parallel Adder (7483/74283)

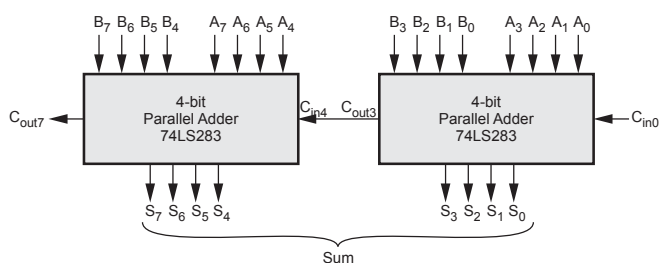
Many high-speed adders available in integrated-circuit form utilize the look-ahead carry or a similar technique for reducing overall propagation delays. The most common is a 4-bit parallel adder IC (74LS83/74S283) that contains four inter connected full-adders and the look-ahead carry circuitry needed 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.

**Fig. 3.8.1 Functional symbol for the 74LS283**

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

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 :

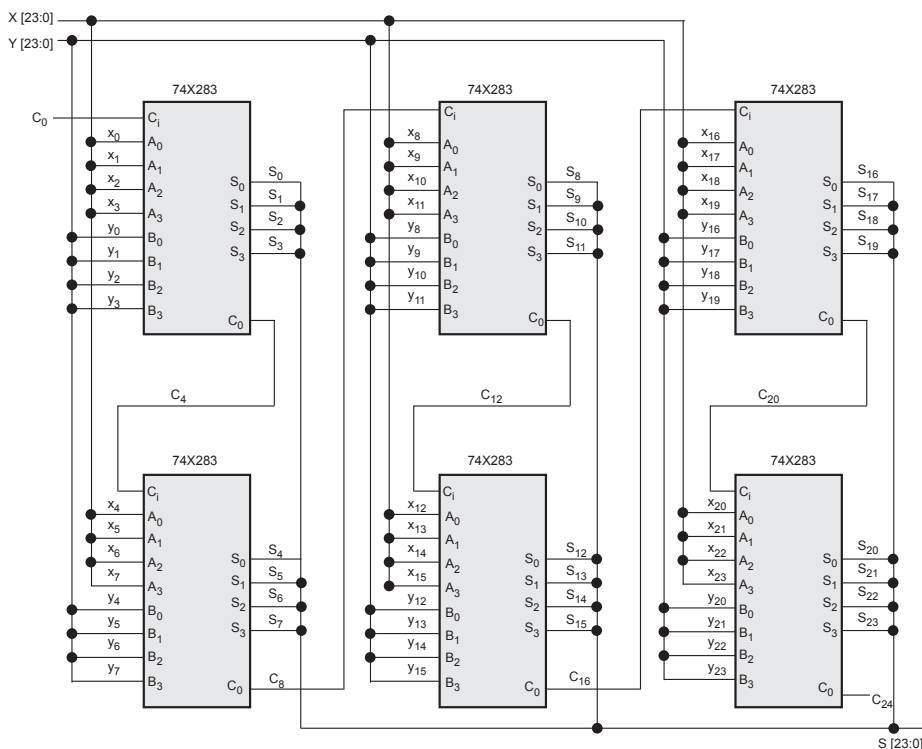


Fig. 3.8.3 A 24-bit group ripple adder using 74X283 ICs

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 $\text{sum} \leq 9$ and $C_{\text{out}} = 0$) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when $\text{sum} > 9$ or $C_{\text{out}} = 1$), binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

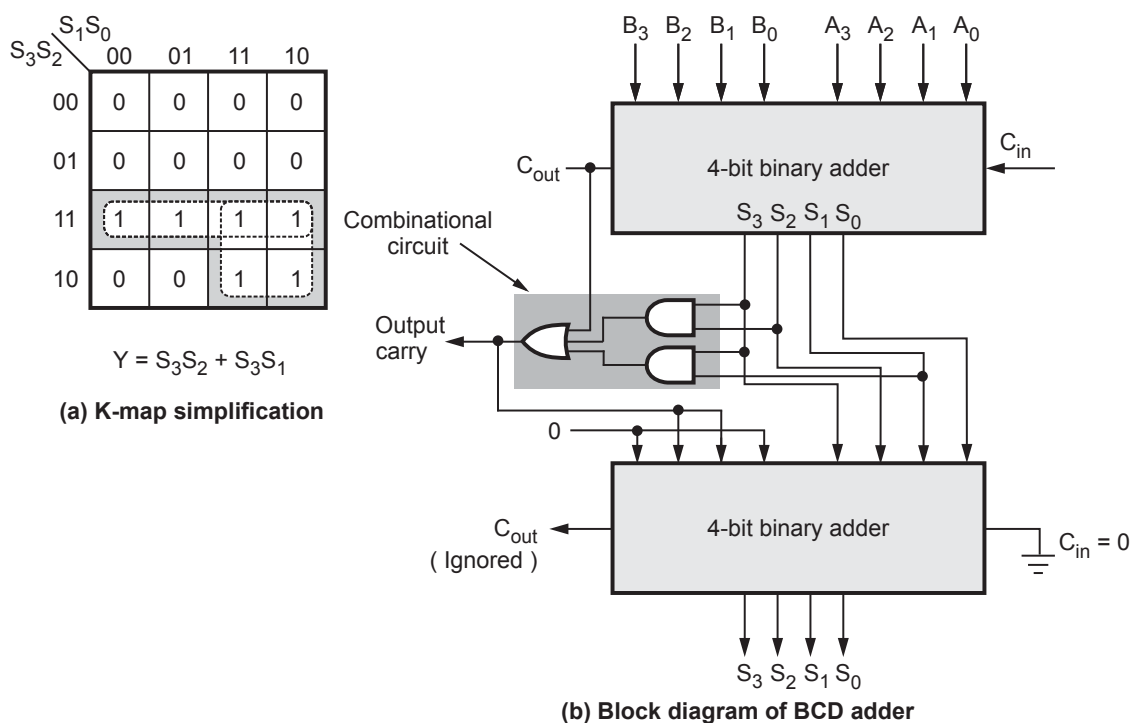


Fig. 3.9.1

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.

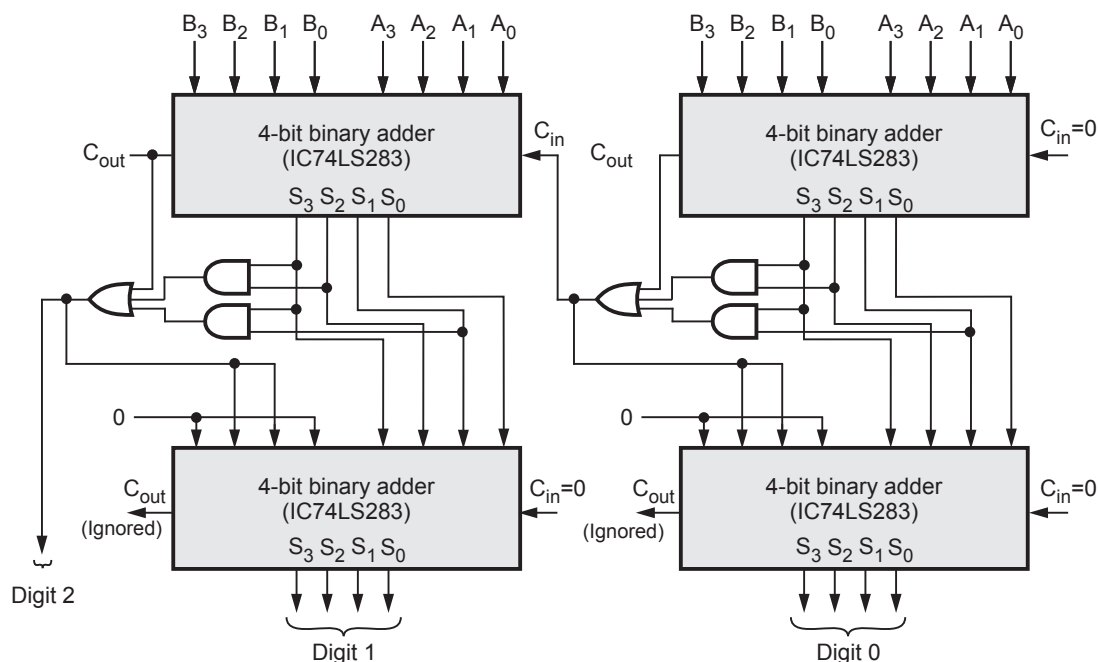


Fig. 3.9.2 8-bit BCD adder using IC 74283

Example 3.9.2 Design a BCD to Excess-3 code converter using binary parallel adder.

Solution :

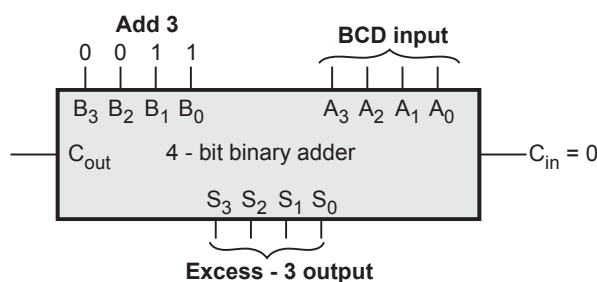


Fig. 3.9.3

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.

$$\begin{array}{r} 0101 \\ + 0011 \\ \hline 1000 \end{array}$$

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 an 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.

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

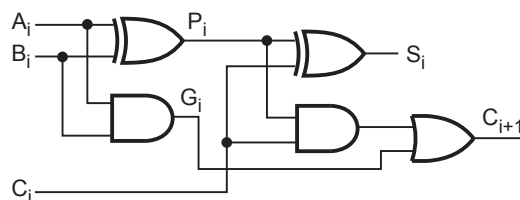


Fig. 3.10.1 Full-adder circuit

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.

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 C_1)$$

$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

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 ,

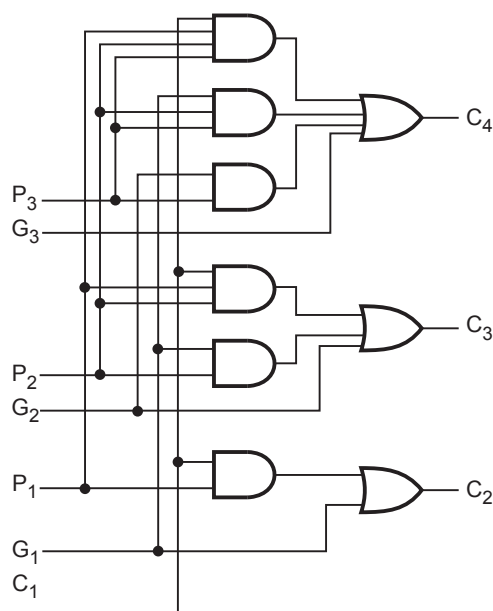


Fig. 3.10.2 Logic diagram of a look ahead carry generator

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.

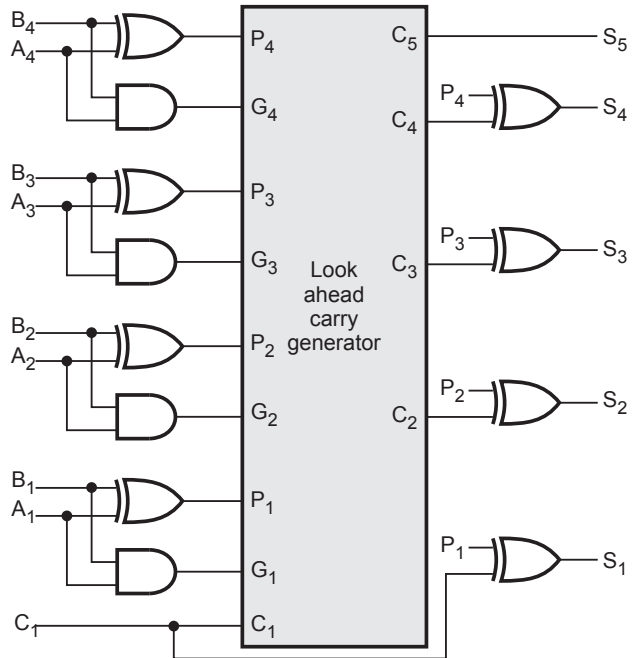


Fig. 3.10.3 4-bit parallel adder with look ahead carry generator

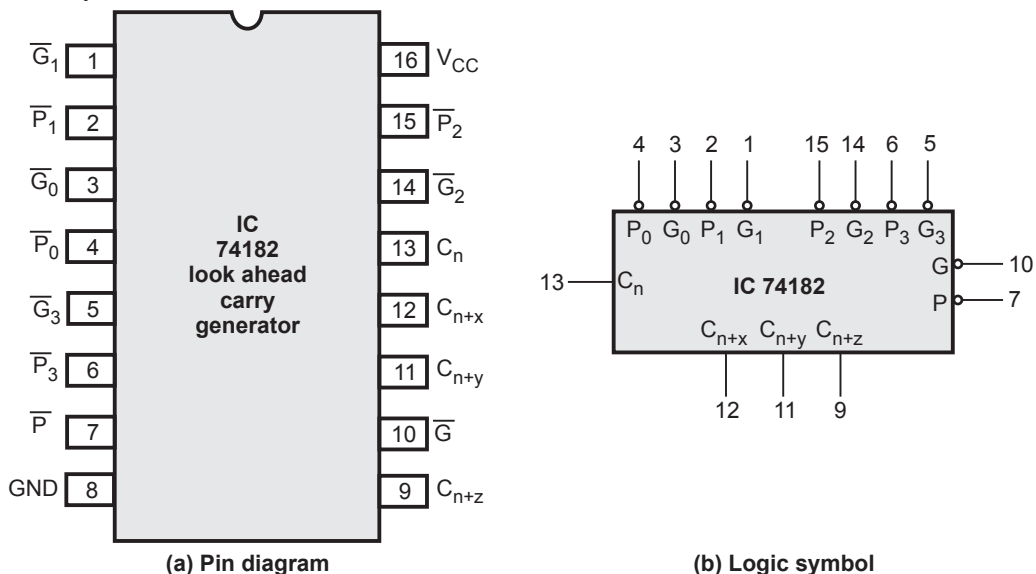


Fig. 3.10.4

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

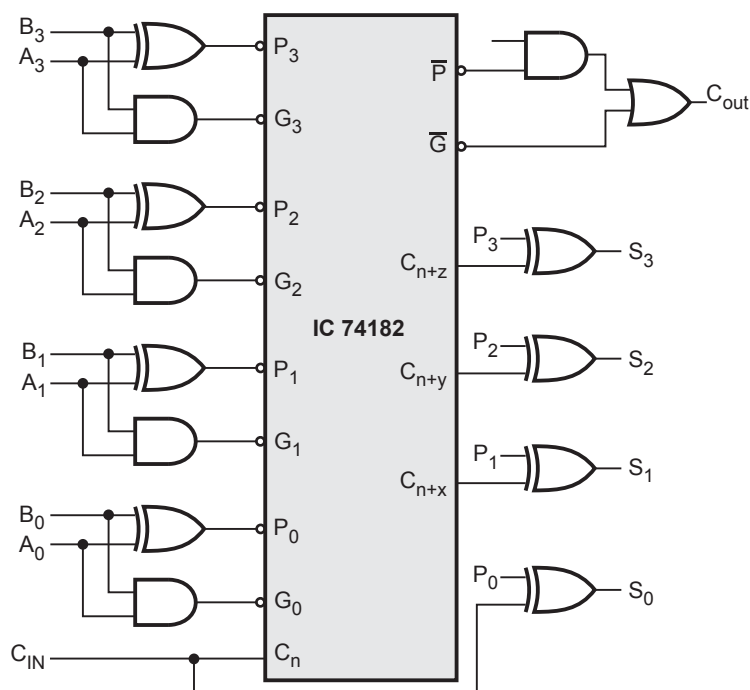


Fig. 3.10.5 4-bit parallel adder using IC 74182

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)

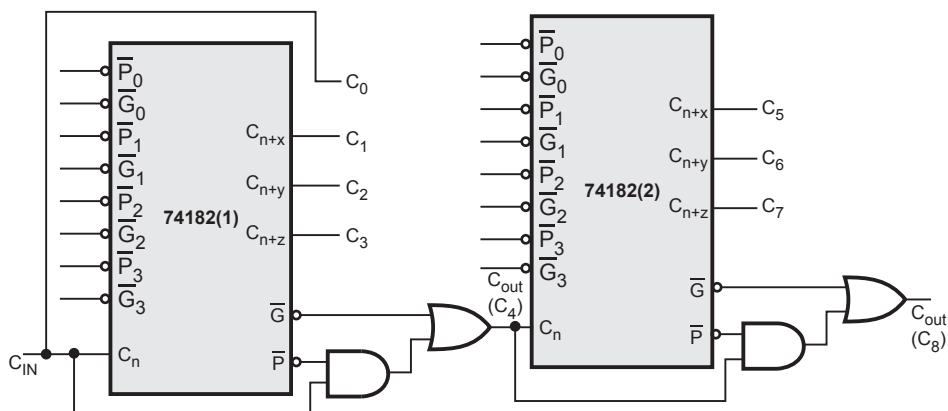


Fig. 3.10.6 Cascading two look ahead carry generators

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.

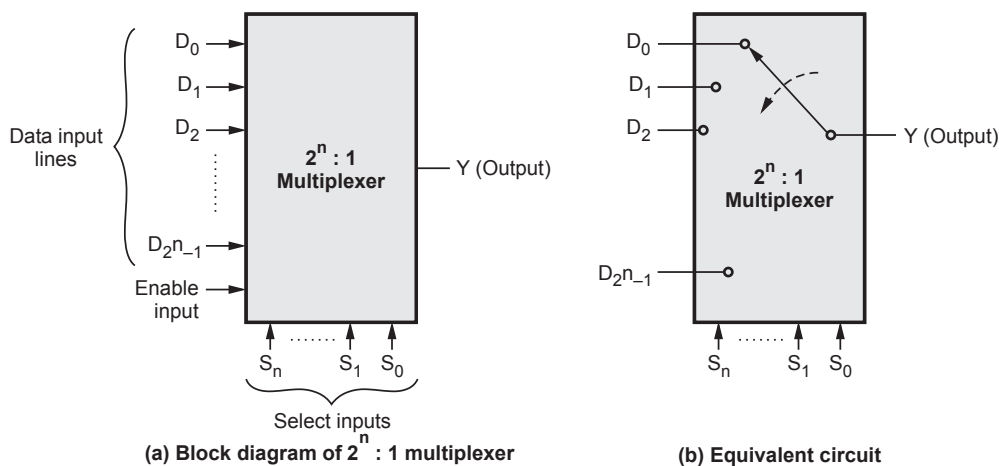


Fig. 3.11.1

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.

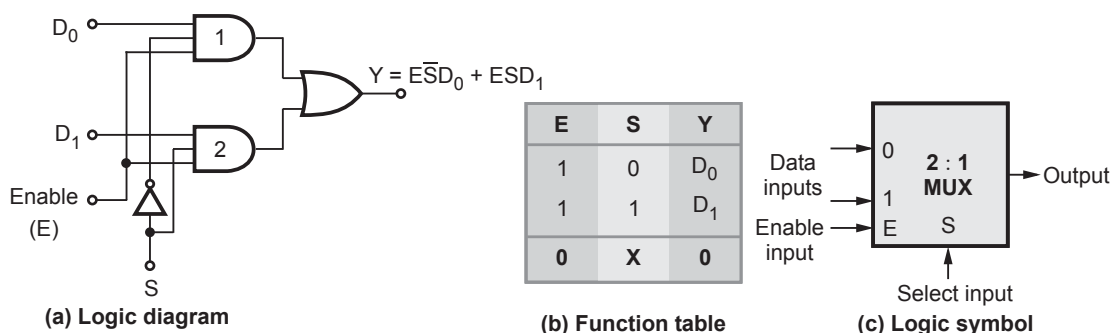


Fig. 3.11.2

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.

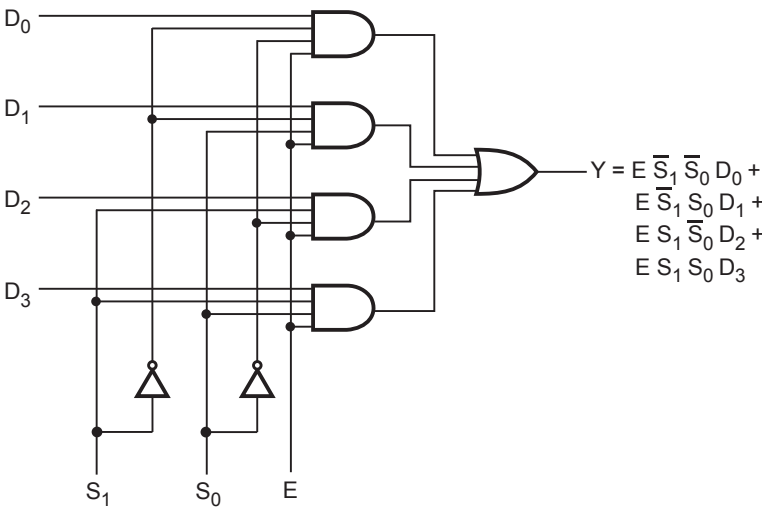
$$\therefore Y = E \bar{S} D_0 + E S D_1$$

Enable (E)	Select (S)	D_1	D_0	Output Y	
1	0	X	0	0	
1	0	X	1	1	$E \bar{S} D_0$
1	1	0	X	0	
1	1	1	X	1	$E S D_1$
0	X	X	X	0	

Table 3.11.1 Truth table for 2 : 1 multiplexer

3.11.2 4 : 1 Multiplexer

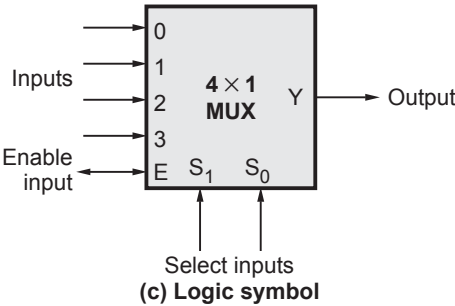
Fig. 3.11.3 (a) shows 4-to-1 line multiplexer. Each of the four lines, D_0 to D_3 , is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.



(a) Logic diagram

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



(c) Logic symbol

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.

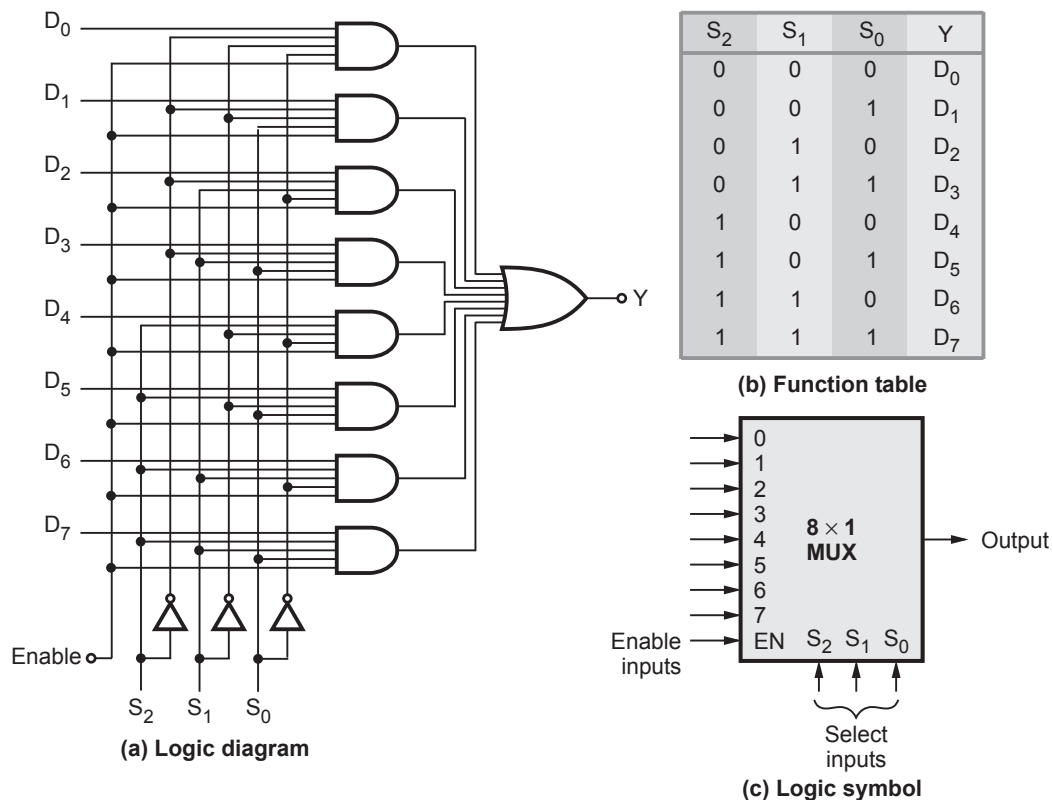


Fig. 3.11.4 8 : 1 Multiplexer

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

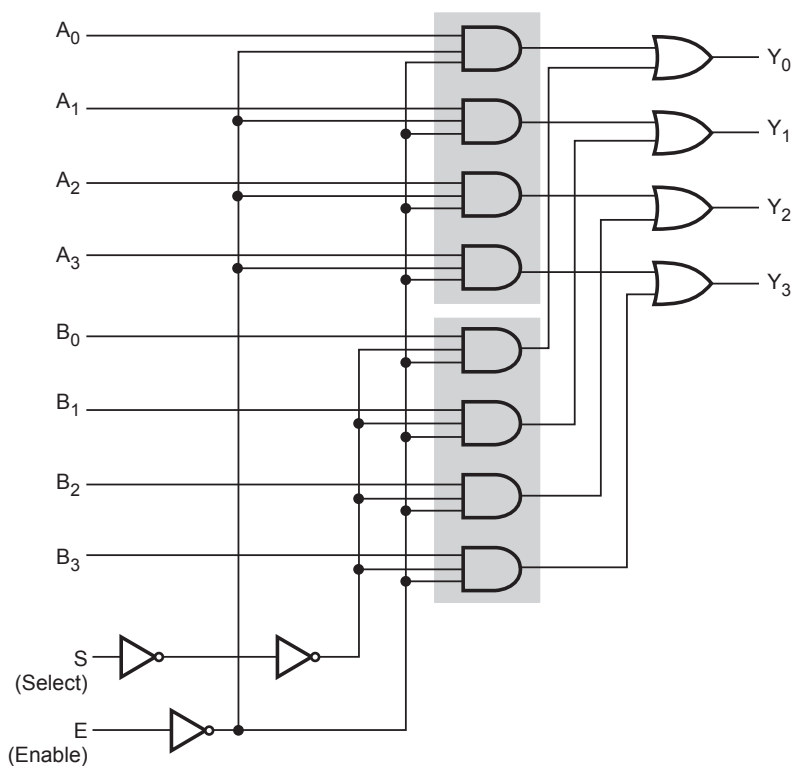


Fig. 3.11.5 Quaduple 2-to-1 line multiplexer

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 \overline{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

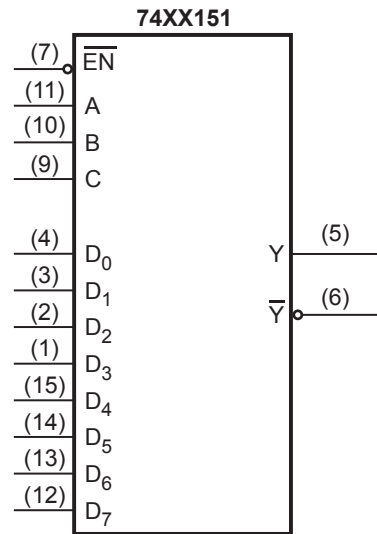


Fig. 3.11.6 Logic symbol for 74XX151, 8 to 1 multiplexer

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

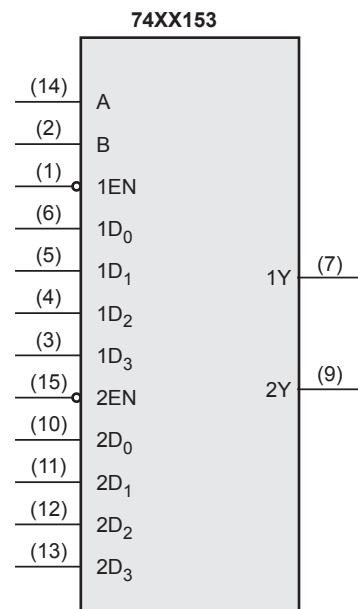


Fig. 3.11.7 Logic symbol for 74XX153

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.

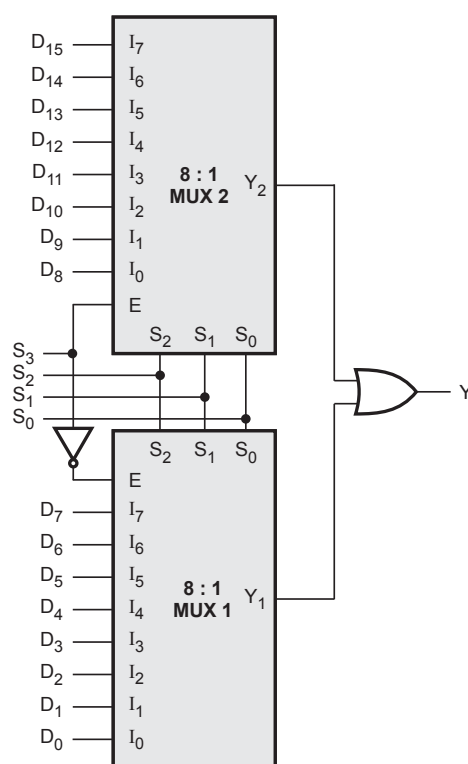


Fig. 3.11.8 16 : 1 multiplexer using two 8 : 1 multiplexers

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.

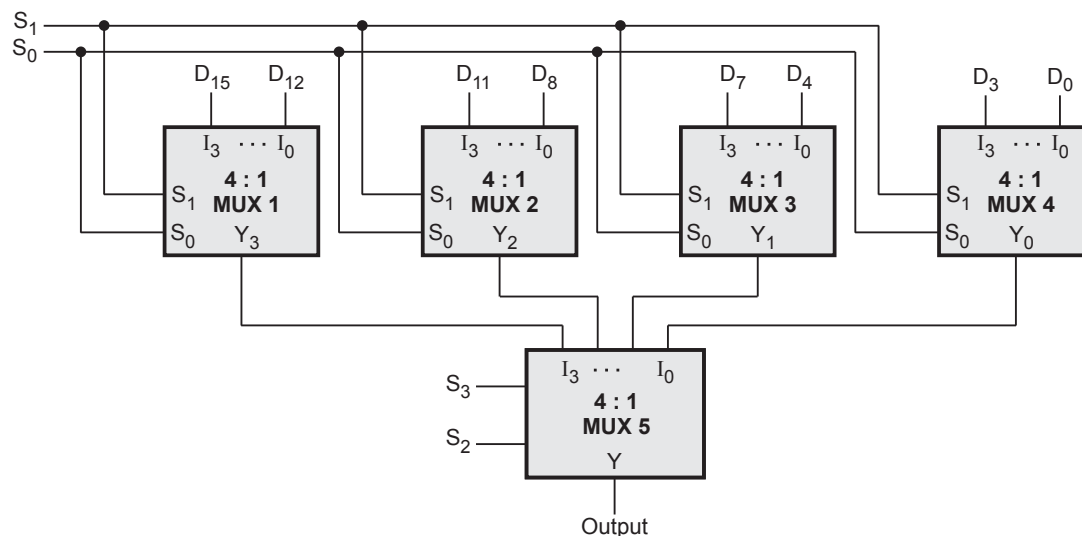


Fig. 3.11.9

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.

$$F(x, y, z) = \sum(0, 2, 6, 7).$$

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.

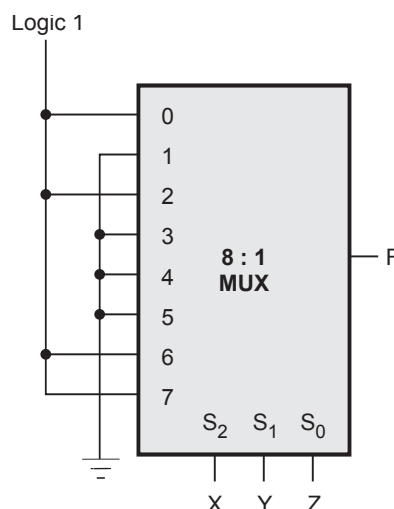


Fig. 3.11.10

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.

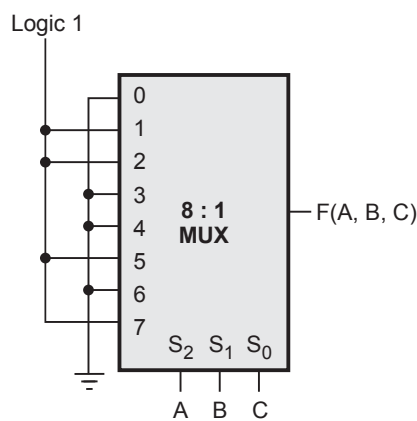


Fig. 3.11.11

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.

$$F(A, B, C) = \sum m(1, 3, 5, 6)$$

Solution : Step 1 : Connect least significant variables as a select inputs of multiplexer. Here, connect C to S_0 and B to S_1 .

Step 2 : Derive inputs for multiplexer using implementation table.

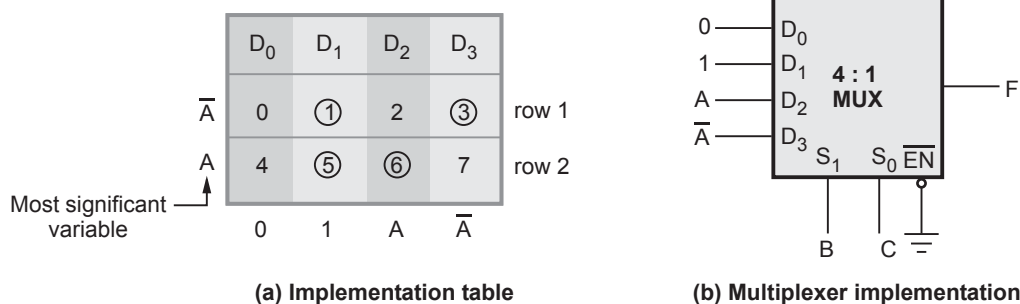


Fig. 3.11.12

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

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

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

$$\begin{aligned}
 F(A, B, C, D) &= \bar{A} B \bar{D} (C + \bar{C}) + A C D (B + \bar{B}) + \bar{B} C D (A + \bar{A}) + \bar{A} \bar{C} D (B + \bar{B}) \\
 &= \bar{A} B C \bar{D} + \bar{A} B \bar{C} \bar{D} + A B C D + A \bar{B} C D \\
 &\quad + A \bar{B} C \bar{D} + \bar{A} \bar{B} C D + \bar{A} B \bar{C} D + \bar{A} \bar{B} \bar{C} D \\
 &= \bar{A} B C \bar{D} + \bar{A} B \bar{C} \bar{D} + A B C D + A \bar{B} C D + \bar{A} \bar{B} C D + \bar{A} B \bar{C} D + \bar{A} \bar{B} \bar{C} D \\
 &= \sum m(6, 4, 15, 11, 3, 5, 1) \\
 &= \sum m(1, 3, 4, 5, 6, 11, 15)
 \end{aligned}$$

Step 2 : Implement it using implementation table.

From the Boolean function in the minterm form can be implemented using 8 : 1 multiplexer as follows :

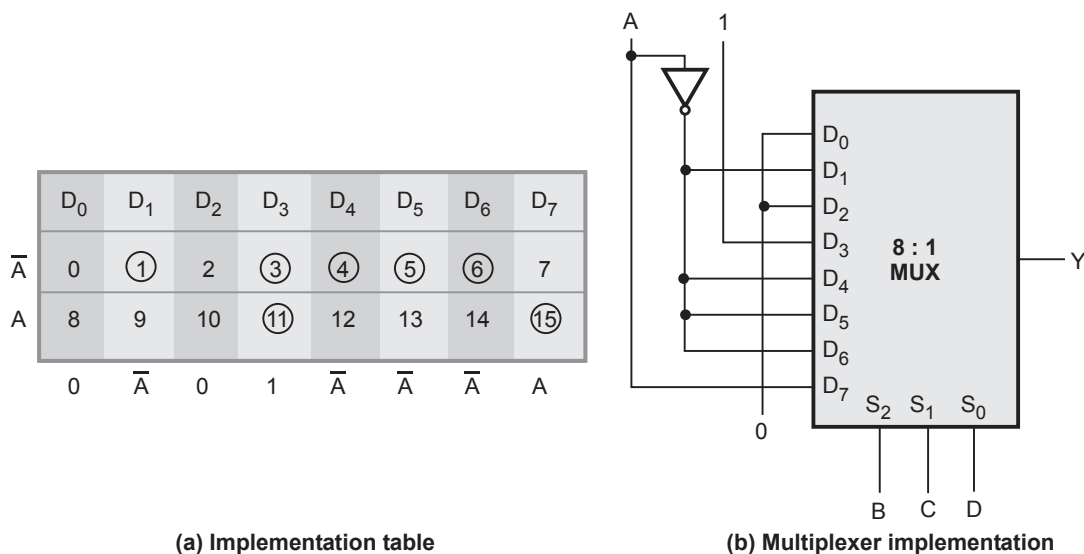


Fig. 3.11.13

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.

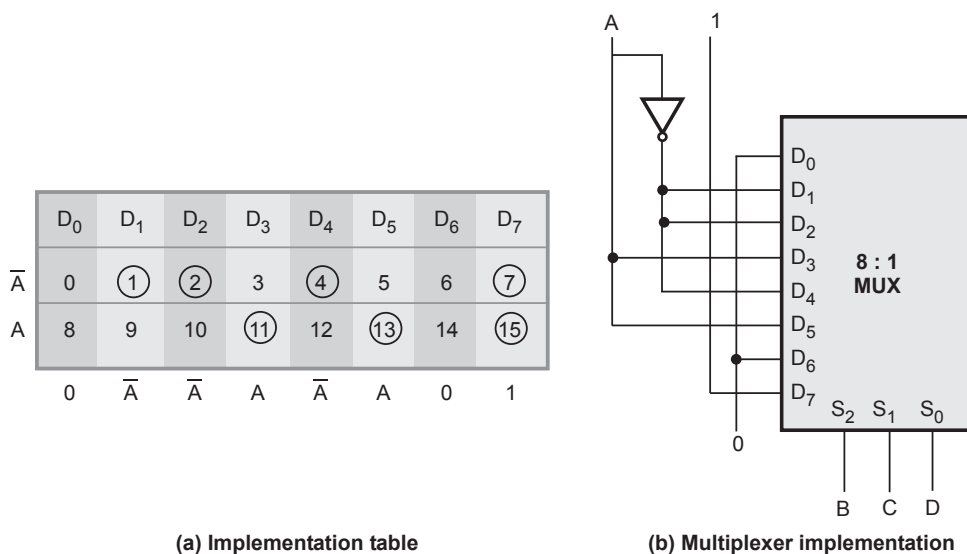


Fig. 3.11.14

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.

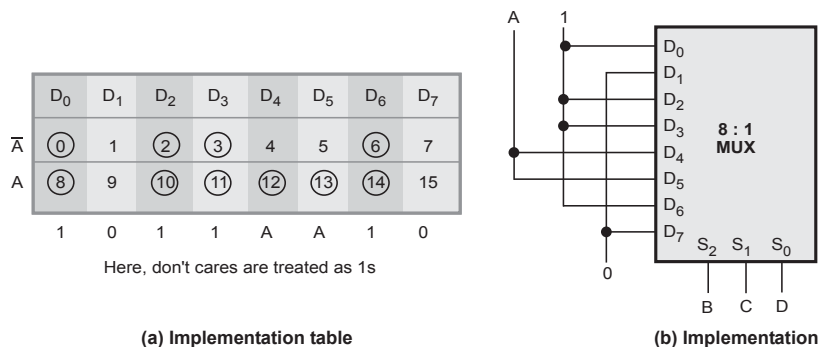


Fig. 3.11.15

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 :

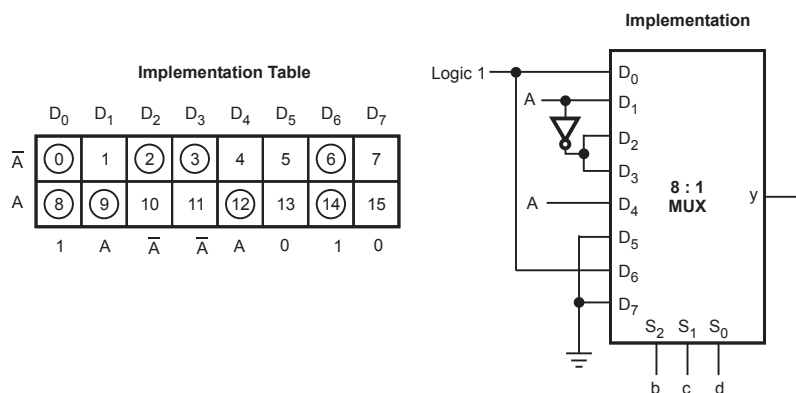


Fig. 3.11.16

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

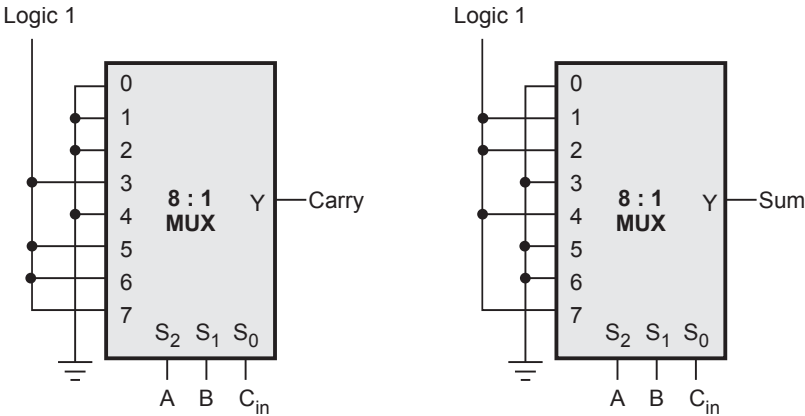


Fig. 3.11.17

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

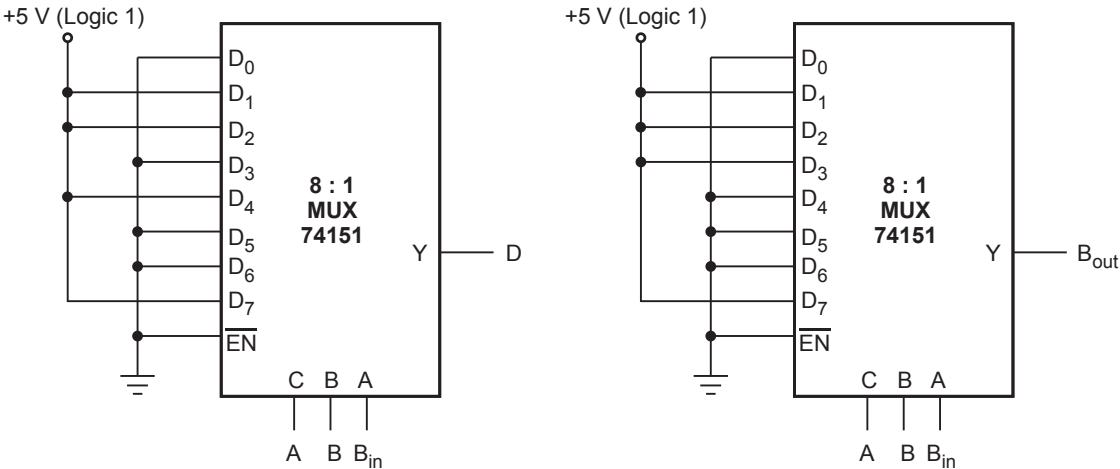


Fig. 3.11.18

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)

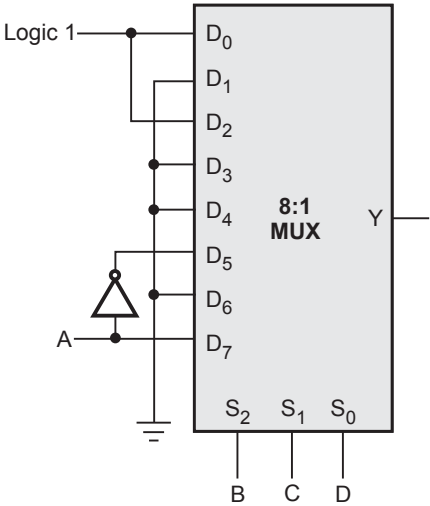


Fig. 3.11.19 (b)

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.

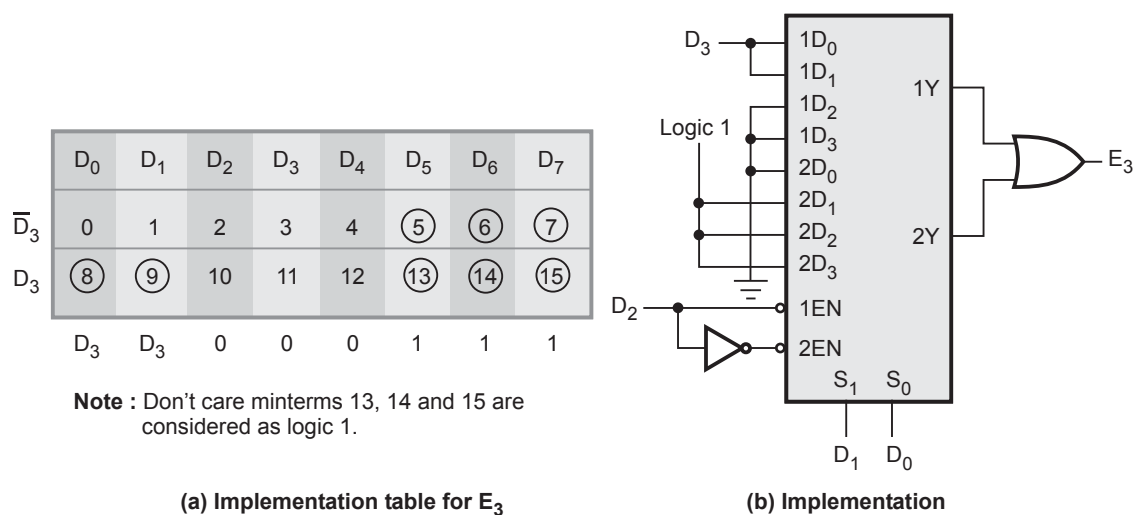


Fig. 3.11.20

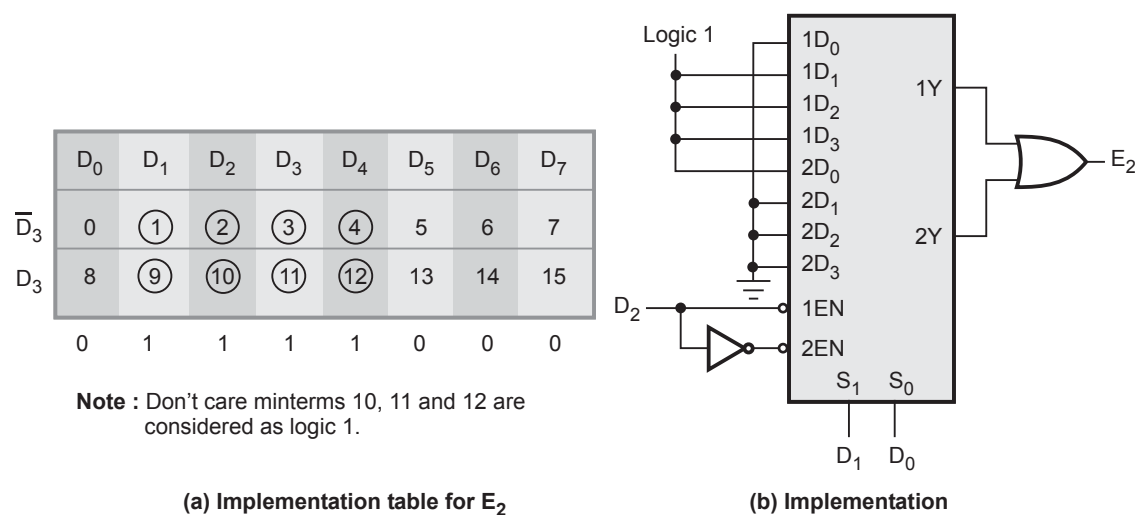


Fig. 3.11.21

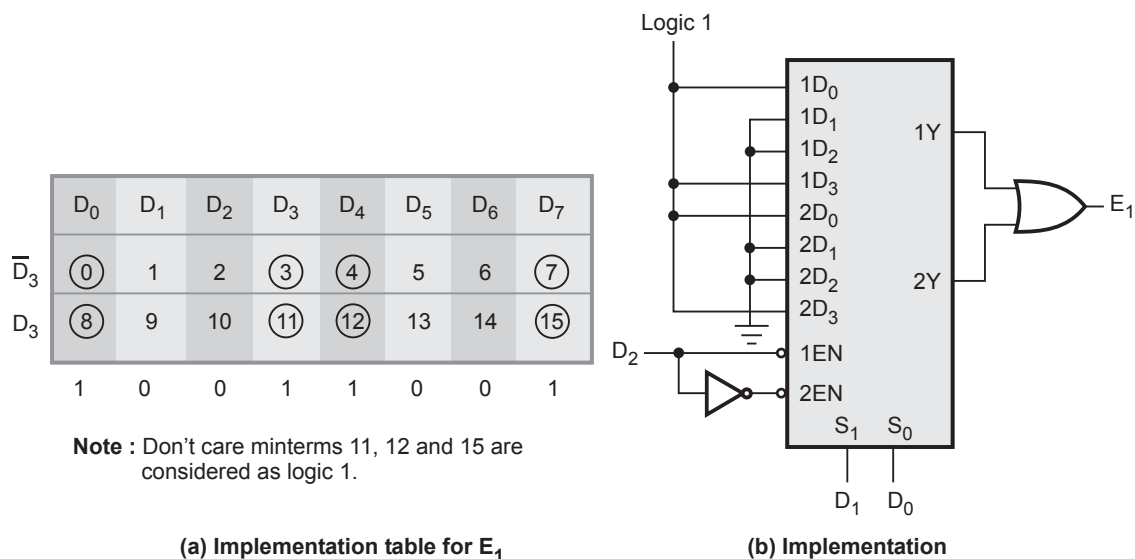


Fig. 3.11.22



Examples for Practice

Example 3.11.18 Implement the following expression using 8 : 1 multiplexer :

$$f(A, B, C, D) = \sum m(2, 4, 6, 7, 9, 10, 11, 12, 15).$$

SPPU : Dec.-12, Marks 8

Example 3.11.19 Implement Boolean function $f = AB + \bar{C}D + A\bar{B}C$ 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

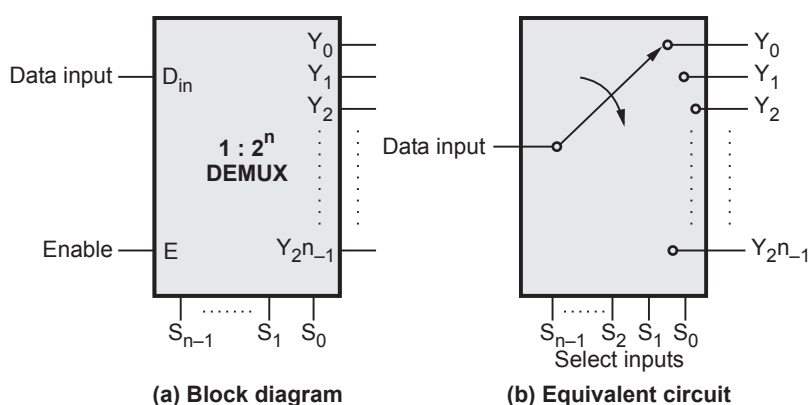
IC number	Function
74150	16 : 1 multiplexer
74151	8 : 1 multiplexer
74153	Dual 4 : 1 multiplexer
74157	Quad 2-input multiplexer

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.

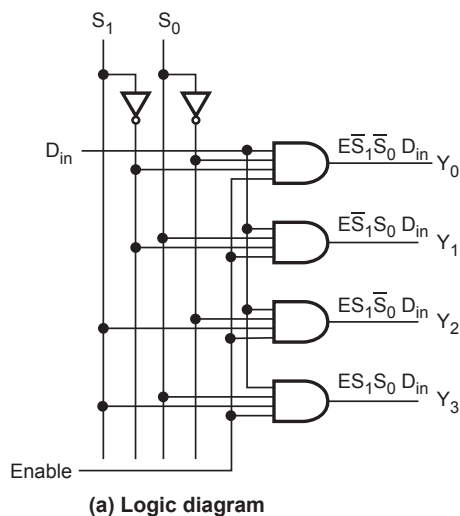
Differentiate between Multiplexer and Demultiplexer

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.



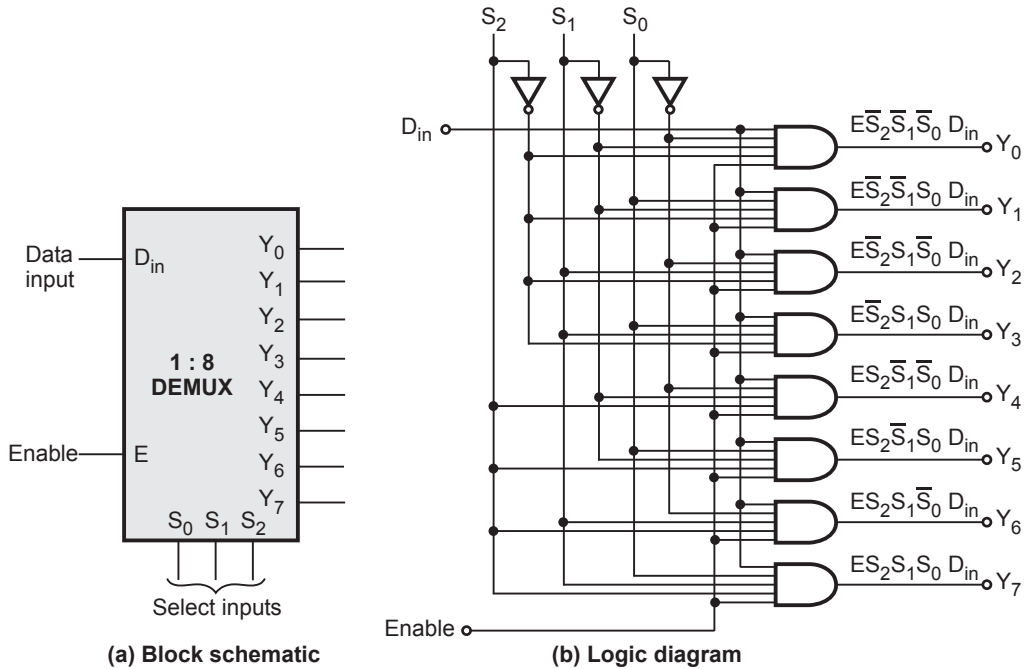
Enable (E)	S_1	S_0	D_{in}	Y_0	Y_1	Y_2	Y_3
0	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

(b) Function table for 1 : 4 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
1	0	0	0	D_{in}	0	0	0	0	0	0	0
1	0	0	1	0	D_{in}	0	0	0	0	0	0
1	0	1	0	0	0	D_{in}	0	0	0	0	0
1	0	1	1	0	0	0	D_{in}	0	0	0	0
1	1	0	0	0	0	0	0	D_{in}	0	0	0
1	1	0	1	0	0	0	0	0	D_{in}	0	0
1	1	1	0	0	0	0	0	0	0	D_{in}	0
1	1	1	1	0	0	0	0	0	0	0	D_{in}

(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.

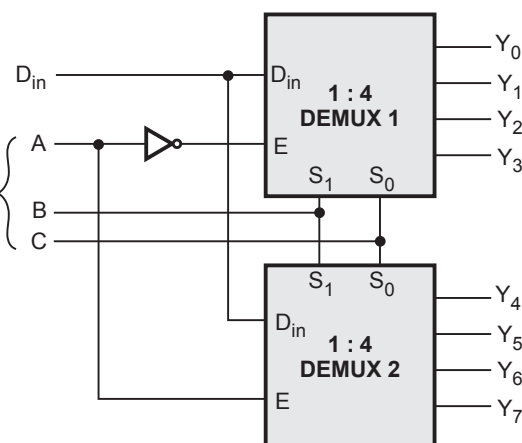


Fig. 3.12.4 Cascading of demultiplexers

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.

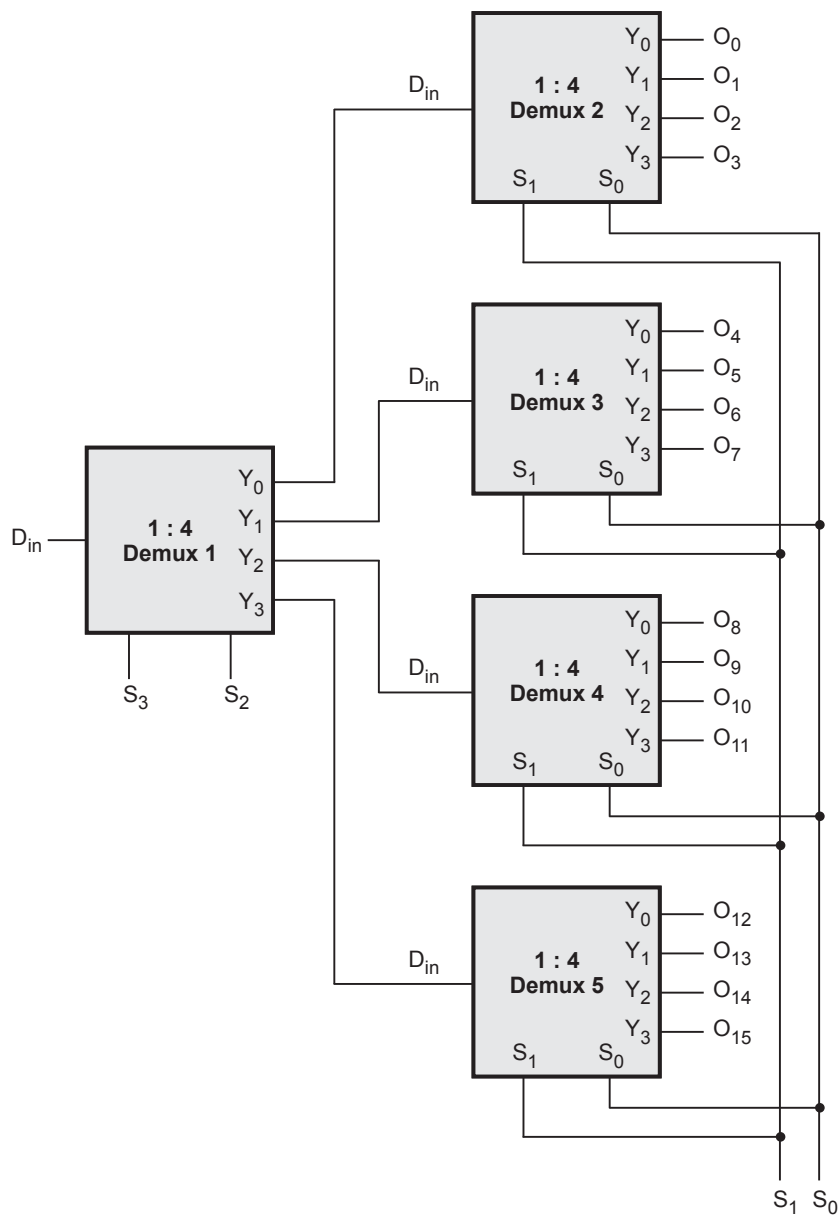


Fig. 3.12.5 1 : 16 Demux using 1 : 4 Demux

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

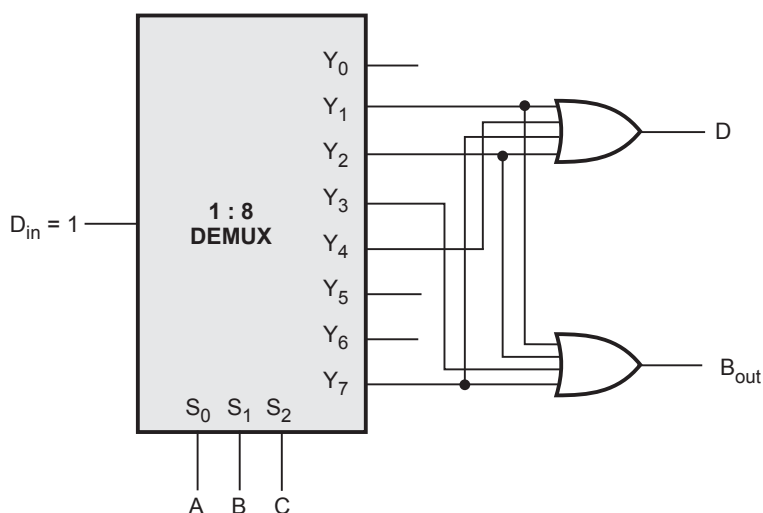


Fig. 3.12.6 Full subtractor using 1 : 8 demultiplexer

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.

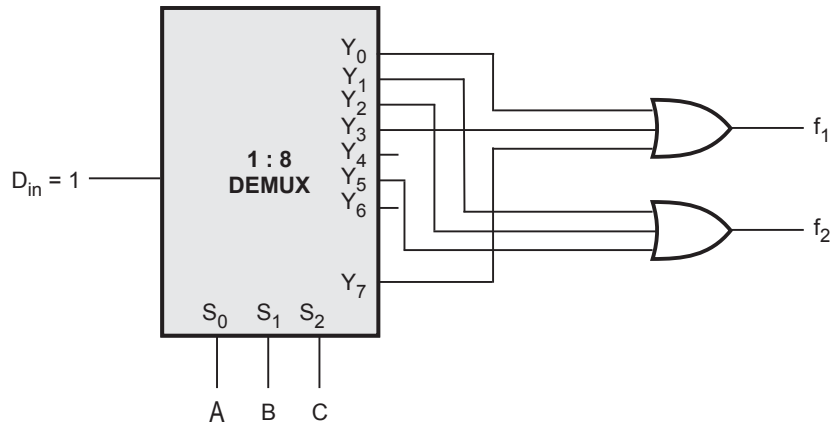


Fig. 3.12.7

Example 3.12.8 Implement the following functions using demultiplexer
 $f_1(A, B, C) = \sum m(1, 5, 7)$, $f_2(A, B, C) = \sum m(3, 6, 7)$

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.

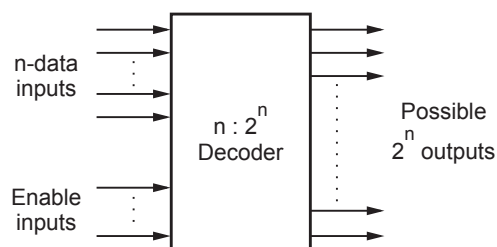


Fig. 3.13.1 General structure of decoder

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

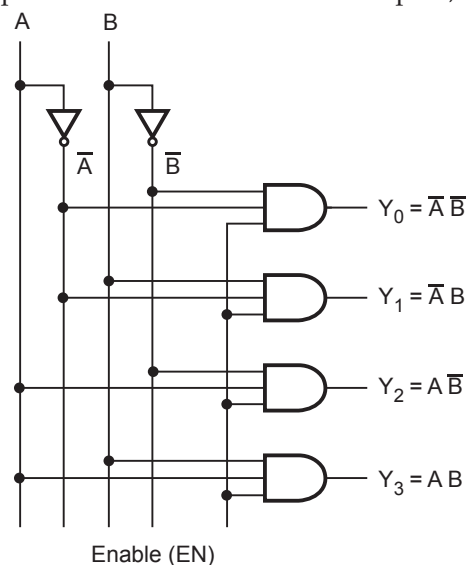


Fig. 3.13.2 2 to 4 line decoder

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.

Inputs				Outputs							
EN	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	1	0	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

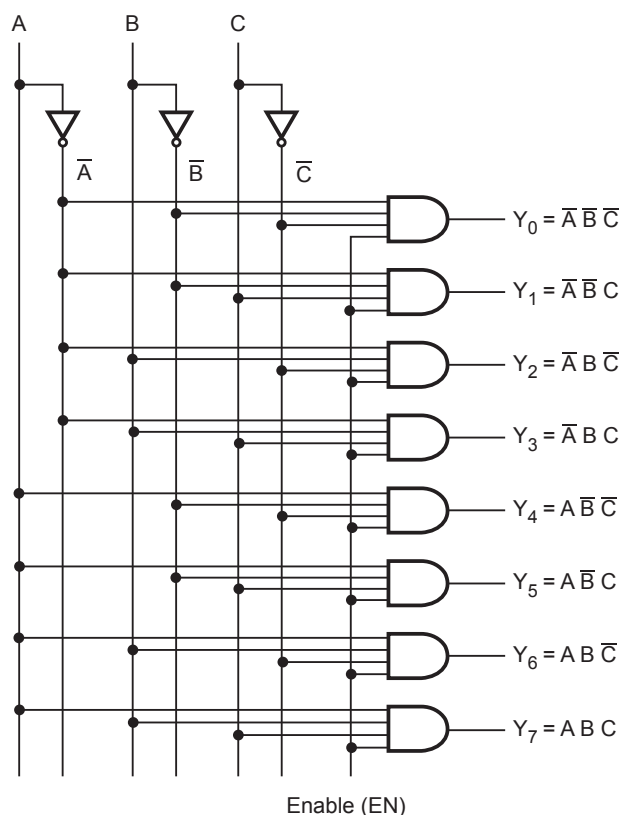


Fig. 3.13.3 3 : 8 line decoder

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 (Y₀ - Y₇). 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	\overline{Y}_7	\overline{Y}_6	\overline{Y}_5	\overline{Y}_4	\overline{Y}_3	\overline{Y}_2	\overline{Y}_1	\overline{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	0	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

Table 3.13.3 Function table

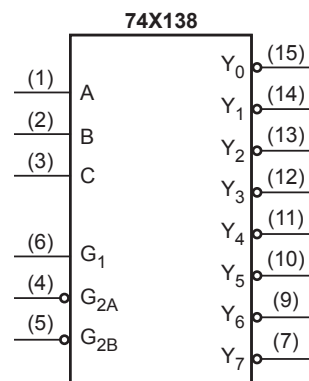


Fig. 3.13.4 Logic symbol

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

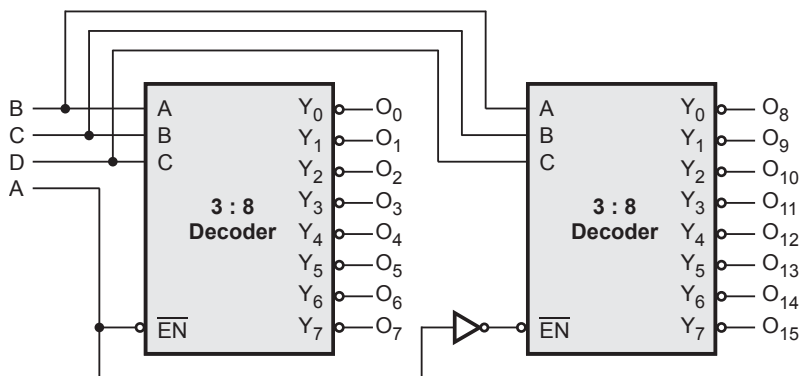


Fig. 3.13.5

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.

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.

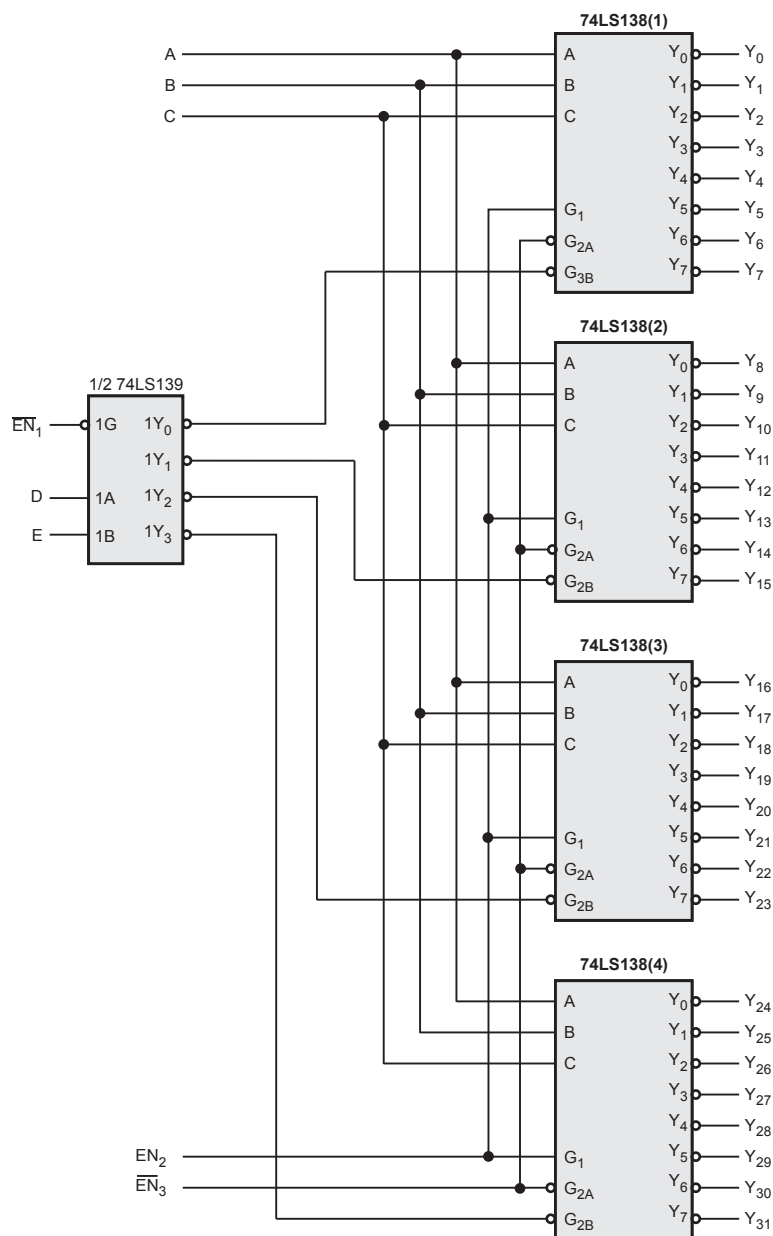


Fig. 3.13.6 5-to-32 decoder using 74LS138 and 74LS139

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.

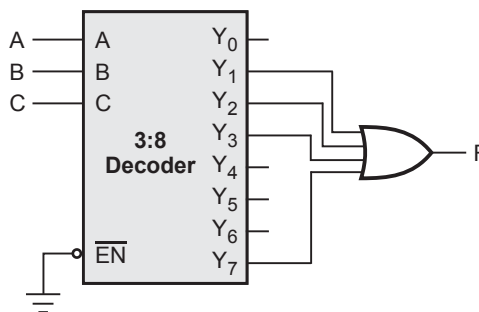


Fig. 3.13.7

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

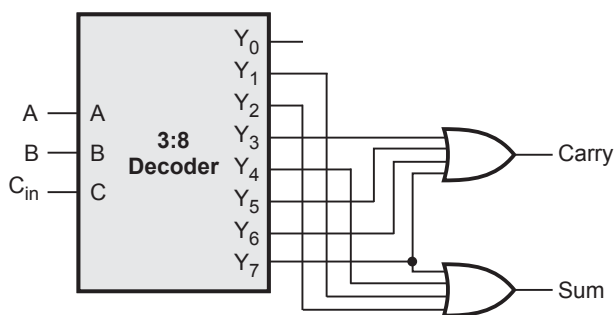


Fig. 3.13.8

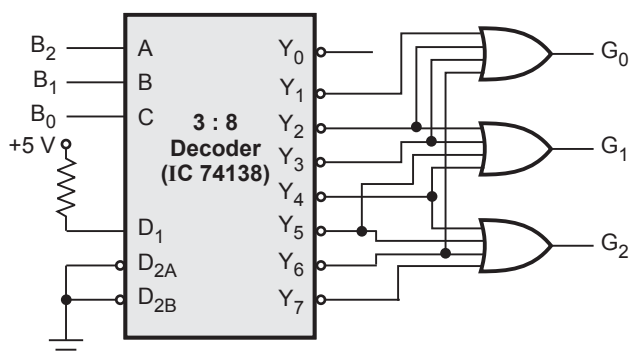
Table 3.13.4 Truth table for full-adder

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

Implementation**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
- Implementation of combinational circuits
- Address decoding
- 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

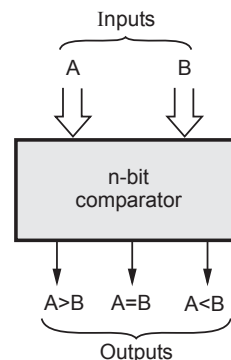
Review Questions

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.

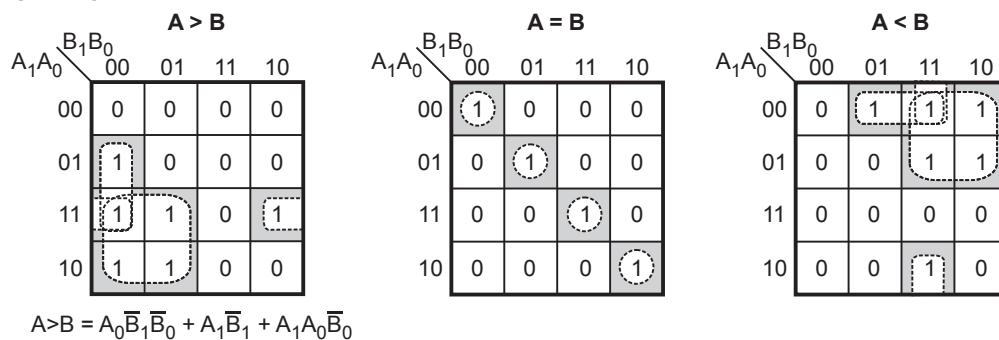
Fig. 3.14.1 shows the block diagram of an n-bit comparator. It receives two n-bit numbers A and B as inputs and the outputs are $A > B$, $A = B$ and $A < B$. Depending upon the relative magnitudes of the two number, one of the outputs will be high.

**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**

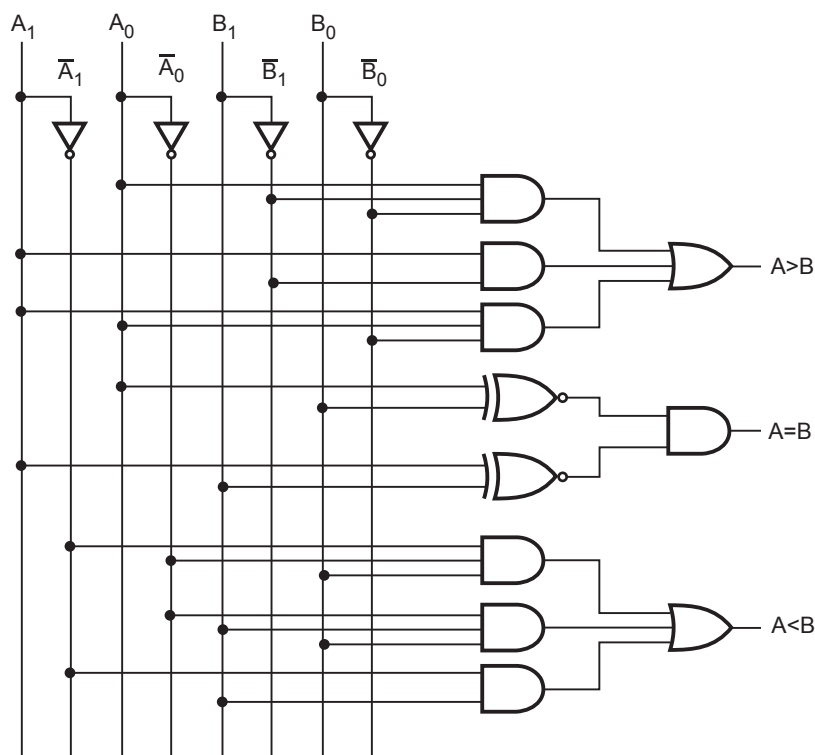
Solution : The truth table for 2-bit is given in Table 3.14.1.

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

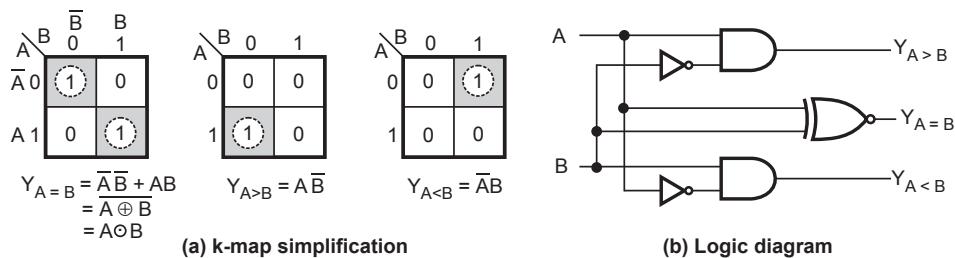
3.14.1 IC 7485 (4-bit Comparator)

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

Table 3.14.2 Function table for IC 7485

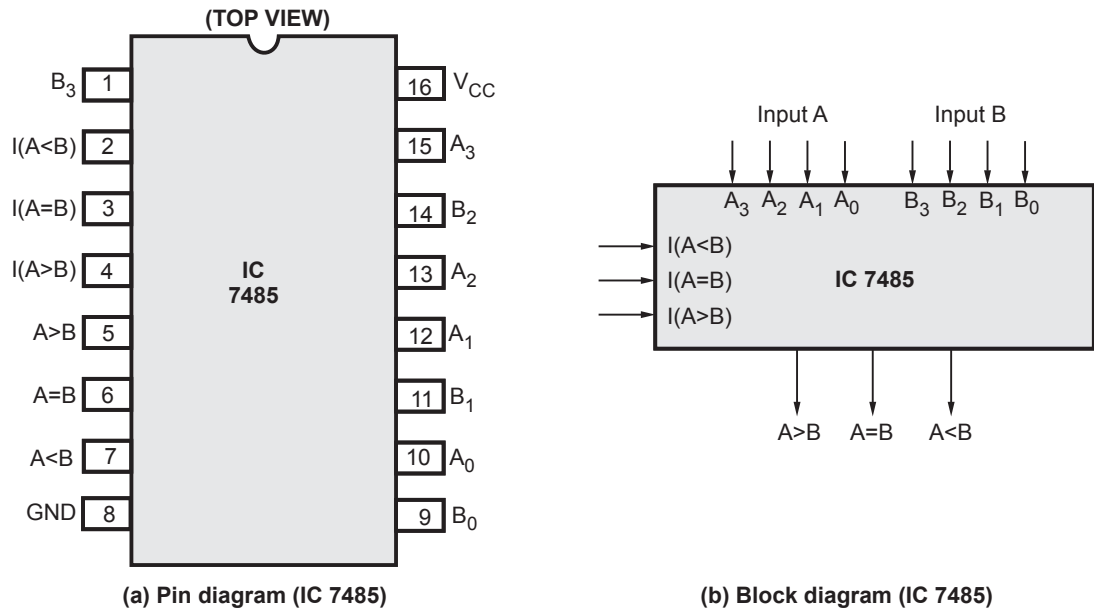


Fig. 3.14.5

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.

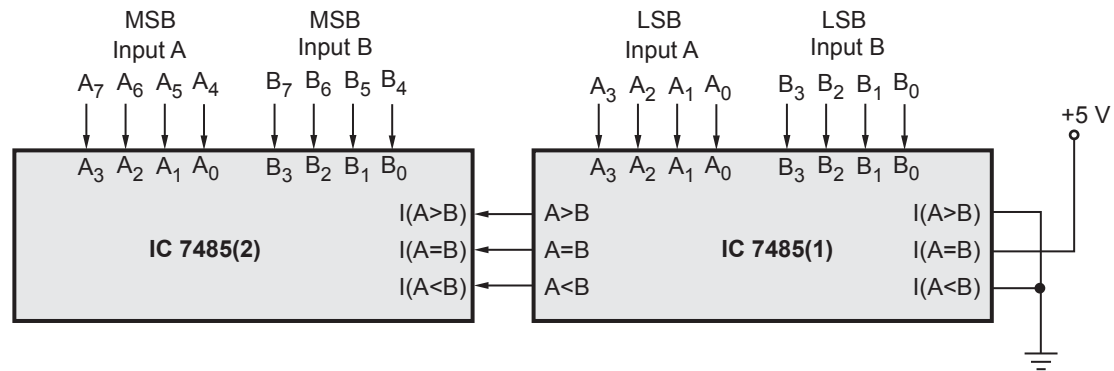


Fig. 3.14.6 8-bit comparator

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 ₀ \ B ₀	0	1
0	0	0
1	1	0

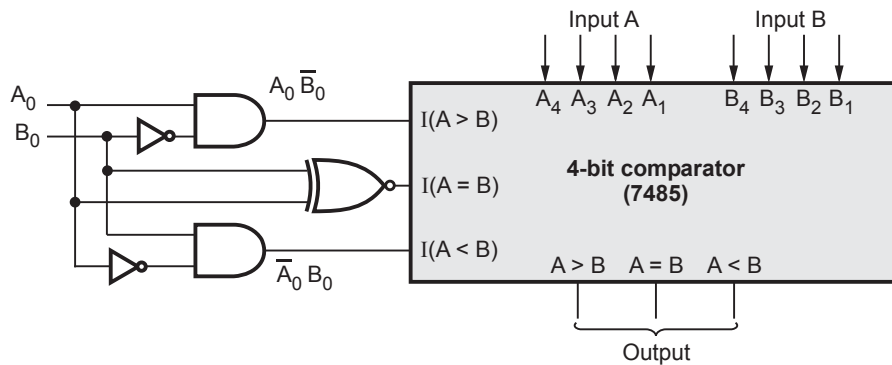
A = B		
A ₀ \ B ₀	0	1
0	1	0
1	0	1

A < B		
A ₀ \ B ₀	0	1
0	0	1
1	0	0

$$(A > B) = A_0 \bar{B}_0$$

$$(A = B) = \bar{A}_0 \bar{B}_0 + A_0 B_0 = A_0 \odot B_0$$

$$(A < B) = \bar{A}_0 B_0$$

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 AeqB, AgtB and AltB 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

$$1_n 1_{n-1} \cdots 1_2 1_1 1_0$$

The above equation says that, if A is greater than B, then the sum of A and \bar{B} should be greater than $1_n 1_{n-1} \cdots 1_2 1_1 1_0$. If n adder is used to add A and \bar{B} , and the final carryout is 1 then we can say that $A + \bar{B} > 1_n 1_{n-1} \cdots 1_2 1_1 1_0$, 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.

$$A + \bar{B} = 1_n 1_{n-1} \cdots 1_2 1_1 1_0$$

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 $n = 4$

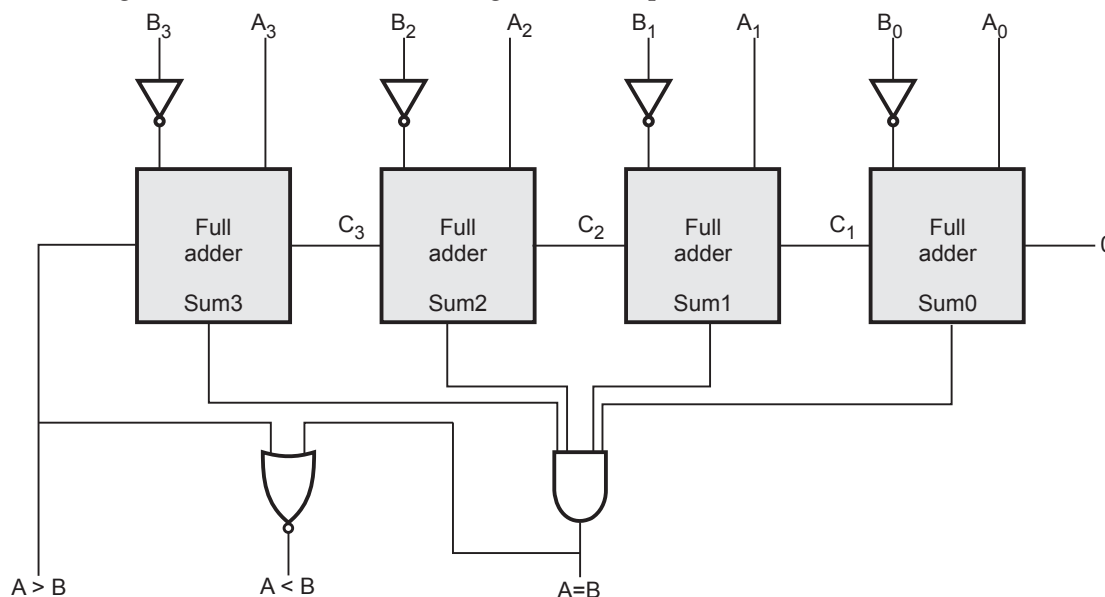


Fig. 3.14.8 4-bit magnitude comparator

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

$$\begin{aligned}
 P &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC \\
 &= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC) \\
 &= \bar{A}(B \oplus C) + A(B \odot C) \\
 &= \bar{A}(B \oplus C) + A(\overline{B \oplus C}) \\
 &= A \oplus (B \oplus C)
 \end{aligned}$$

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

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.

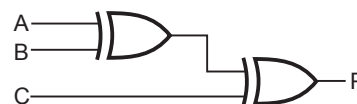


Fig. 3.15.1 (b)

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

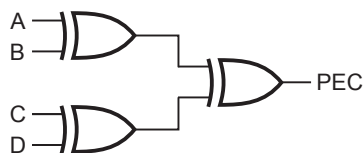
$$\begin{aligned}
 \text{PEC} &= \bar{A}\bar{B}(\bar{C}D + C\bar{D}) + \bar{A}B(\bar{C}\bar{D} + CD) \\
 &\quad + A\bar{B}(\bar{C}D + C\bar{D}) + A\bar{B}(\bar{C}\bar{D} + CD) \\
 &= \bar{A}\bar{B}(C \oplus D) + \bar{A}B(\overline{C \oplus D}) \\
 &\quad + A\bar{B}(C \oplus D) + A\bar{B}(\overline{C \oplus D}) \\
 &= (\bar{A}\bar{B} + A\bar{B})(C \oplus D) + (\bar{A}B + A\bar{B})(\overline{C \oplus D}) \\
 &= (\bar{A} \oplus B)(C \oplus D) + (A \oplus B)(\overline{C \oplus D}) \\
 &= (A \oplus B) \oplus (C \oplus D)
 \end{aligned}$$

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

$$\text{PEC} = (A \oplus B) \oplus (C \oplus D)$$

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.

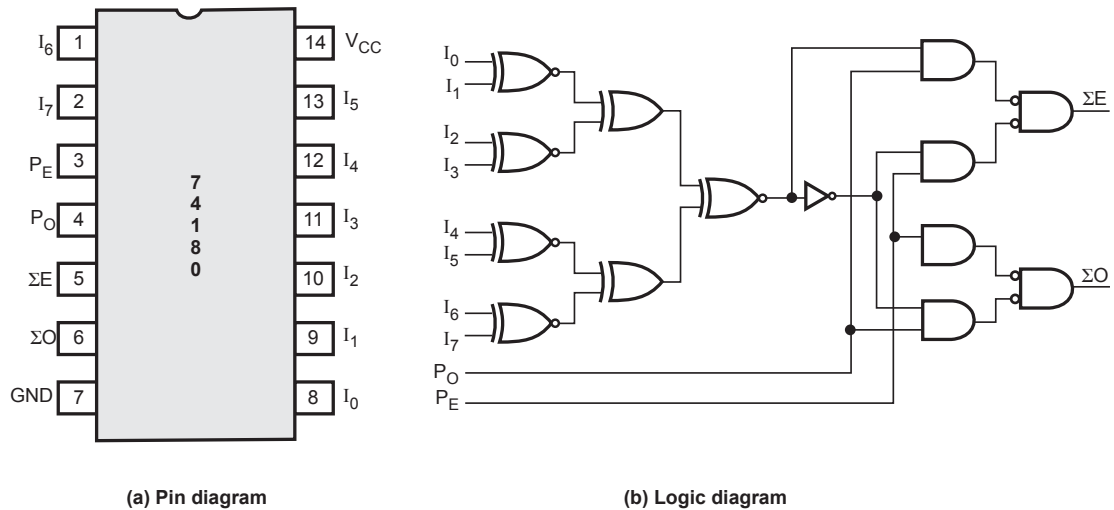


Fig. 3.15.3

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.

Inputs			Outputs	
Number of HIGH data Inputs (I ₀ - I ₇)	P _E	P _O	ΣE	ΣO
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,

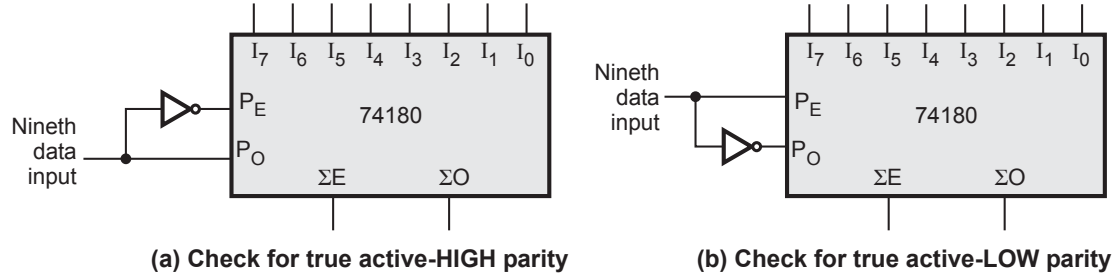


Fig. 3.15.4

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.

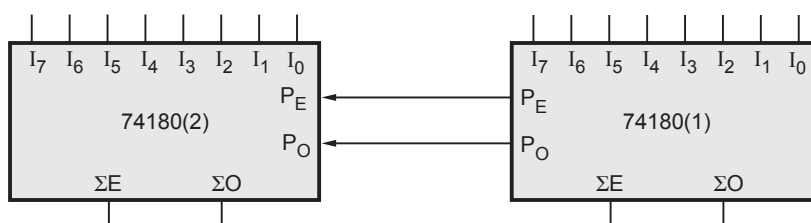


Fig. 3.15.5 Expansion to the larger word

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