

FOR MORE EXCLUSIVE
(Civil, Mechanical, EEE, ECE)
ENGINEERING & GENERAL STUDIES
(Competitive Exams)

TEXT BOOKS, IES GATE PSU's TANCET & GOVT EXAMS
NOTES & ANNA UNIVERSITY STUDY MATERIALS

VISIT

www.EasyEngineering.net

**AN EXCLUSIVE WEBSITE FOR ENGINEERING STUDENTS &
GRADUATES**



****Note:** Other Websites/Blogs Owners Please do not Copy (or) Republish this Materials without Legal Permission of the Publishers.

****Disclimers :** EasyEngineering not the original publisher of this Book/Material on net. This e-book/Material has been collected from other sources of net.

Third Revised Edition - 2008

Digital Electronics

A. P. Godse
D. A. Godse



Technical Publications Pune™

Copyrighted material

Downloaded From : www.EasyEngineering.net



Digital Electronics

ISBN 978-81-89411-32-9

All rights 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 :

Technical Publications Pune[®]

#1, Amit Residency, 412, Shaniwar Peth, Pune - 411 030, India.

Printer :

Alert DTPrinters
Sr no 10/3, Sinhad Road,
Pune - 411 041

Table of Contents

Chapter 1	Number Systems	1
1.1	Introduction	1
1.2	Decimal Number System	1
1.3	Binary Number System	2
1.3.1	Binary Counting	3
1.3.2	Binary-to-Decimal Conversion	3
1.3.3	Fractions	5
1.3.4	Mixed Numbers	5
1.3.5	Decimal to Binary Conversion	6
1.3.6	Fractional Decimal to Binary Conversion	8
1.3.7	Binary Arithmetic	10
1.3.8	Signed Binary Numbers	11
1.3.9	2's Complement Arithmetic	13
1.4	Octal Numbers System	16
1.4.1	Octal to Decimal Conversion	17
1.4.2	Fractions	17
1.4.3	Mixed Numbers	17
1.4.4	Decimal to Octal Conversion	17
1.4.5	Fractional Decimal to Octal Conversion	18
1.4.6	Binary to Octal Conversion	18
1.4.7	Octal to Binary Conversion	19
1.4.8	Usefulness of Octal System	19
1.4.9	Octal Arithmetic	19
1.5	Hexadecimal Number System	23
1.5.1	Hexadecimal to Decimal Conversion	24
1.5.2	Fractions	24
1.5.3	Mixed Numbers	24
1.5.4	Decimal to Hexadecimal Conversion	25
1.5.5	Fractional Decimal to Hexadecimal Conversion	25
1.5.6	Binary to Hexadecimal Conversion	25
1.5.7	Hexadecimal to Binary Conversion	26
1.5.8	Octal to Hexadecimal Conversion	26
1.5.9	Hexadecimal to Octal Conversion	26
1.5.10	Hexadecimal Arithmetic	27

3.2.2 Condition for Error Detection	78
3.3 Parity Bit	78
<u>3.4 Block Parity</u>	<u>80</u>
3.5 Linear Block Codes	81
3.5.1 Matrix Representation of Linear Block Codes	81
3.5.2 Generalized Steps for Construction of Code	83
3.5.3 Decoding the Received Codewords	85
<u>3.5.4 Error Correction</u>	<u>88</u>
3.6 Binary Cyclic Codes	90
3.7 Burst Code	91
<u>3.8 Check-sums and Cyclic Redundancy Checks</u>	<u>92</u>
<u>3.9 Hamming Code</u>	<u>92</u>
<u>3.9.1 Number of Parity Bits</u>	<u>92</u>
3.9.2 Locations of the Parity Bits in the Code	93
3.9.3 Assigning Values to Parity Bits	93
3.9.4 Detecting and Correcting an Error	95
University Questions	97
Chapter 4 Logic Gates and Boolean Algebra	99
4.1 Introduction	99
<u>4.2 Logical Operators</u>	<u>100</u>
<u>4.2.1 Logical Operator NOT/INVERT</u>	<u>100</u>
<u>4.2.2 Logical operator AND</u>	<u>100</u>
4.2.3 Logical Operator OR	101
4.3 Logic Gates	101
4.3.1 Inverter : NOT gate	102
4.3.2 AND gate	103
4.3.3 The OR Gate	106
<u>4.3.4 The NAND gate</u>	<u>108</u>
4.3.5 The NOR Gate	110
<u>4.3.6 The Exclusive-OR Gate</u>	<u>112</u>
4.3.7 The Exclusive-NOR Gate	114
4.4 Boolean Algebra	117
4.4.1 Logic Expressions	117
4.4.2 Laws of Boolean Algebra	118
4.4.3 Rules in Boolean Algebra	121
4.4.4 Duality Theorem	125

7.5.2 Full-Subtractor	348
<u>7.6 n-Bit Parallel Adder</u>	<u>351</u>
7.7 Look Ahead Carry Generator (IC 74182)	352
7.8 Binary Parallel Adder (IC 74LS83/74LS283)	357
7.9 n-Bit Parallel Subtractor	358
<u>7.10 Binary Adder-Subtractor</u>	<u>359</u>
<u>7.11 BCD Adder</u>	<u>359</u>
7.12 BCD Subtractor	363
7.12.1 9's Complement	364
7.12.2 9's Complement Subtraction	364
7.12.3 10's Complement Subtraction	365
7.13 Excess-3 Adder	368
7.14 Excess-3 Subtraction	369
7.15 Digital Comparator	371
<u>University Questions</u>	<u>378</u>
Chapter 8 Combinational Logic with MSI and LSI	381
<hr/>	
8.1 Introduction	381
8.2 Multiplexer	381
8.2.1 Boolean Function Implementation	384
8.2.2 Multiplexer ICs	392
<u>8.3 Demultiplexer</u>	<u>393</u>
8.4 Decoders	396
8.4.1 Boolean Function Implementation	397
<u>8.4.2 Decoder IC 74138</u>	<u>398</u>
8.4.3 Decoder IC 7442	401
8.5 Encoder	401
8.5.1 Priority Encoder	402
8.5.2 Encoder IC (74148)	404
8.6 BCD-To- 7 Segment Display Decoder/Driver	405
8.6.1 Basic Connection for Driving 7-Segment Displays	409
8.6.2 Seven Segment Decoder/Driver for LCD Display	416
8.7 Parity Generator and Checker	418
8.8 Design of ALU	423
8.8.1 Design of Arithmetic Unit	423
8.8.2 Design of Logic Unit	426

12.5.2 Dynamic RAM	689
12.5.2.1 Write Operation	690
12.5.2.2 Read Operation	691
12.5.2.3 Refresh Operation	691
12.5.3 Comparison Between SRAM and DRAM	692
12.6 Memory Structure and Its Requirements	692
12.7 Standard Memory Chips	693
12.7.1 Characteristics of Memory :	693
12.7.2 Standard Memory Chips	694
12.7.3 Memory Cycles	694
12.8 Typical Memory ICs	697
12.8.1 IC 2764 (EPROM)	697
12.8.2 IC 2114 (RAM)	698
12.9 Expanding Word Size	700
12.10 Expanding Memory Capacity	700
<u>University Questions</u>	<u>702</u>
Chapter 13 Programmable Logic Devices	703
13.1 Introduction	703
13.2 Read Only Memory (ROM)	703
13.2.1 Combination Logic Implementation Using ROM	704
13.2.2 Types of ROMs	708
13.3 Programmable Logic Array (PLA)	708
<u>13.4 Programmable Array Logic (PAL)</u>	<u>714</u>
University Questions	723
Appendix - A Typical Digital ICs	724
References	728
Index	729

Streamlined method

We can streamline binary to decimal conversion by performing following steps :

1. Write the binary number
2. Directly under the binary number write 1, 2, 4, 8, 16 ..., working from right to left.
3. If digit is zero, cross out the decimal weight for that position.
4. Add the remaining weight to obtain the decimal equivalent of the binary number.

Ex. 1.3 : Convert binary number 1010 to its decimal equivalent.

Sol. :

Step 1 : 1 0 1 0 (number)

Step 2 : 8 4 2 1 (Binary weights)

Step 3 : 8 ~~4~~ 2 1

Step 4 : 8 + 2 = 10₁₀

Note : The subscript 10 identifies this as a decimal number.

1.3.2.1 Double Dabble Method

A popular way to convert binary numbers to decimal numbers is the double dabble method. In this method, the conversion takes place in the following steps :

1. Multiply most significant digit (leftmost digit) by 2.
2. Add next adjacent digit to the result.
3. Multiply result by 2.
4. Add next adjacent digit to the result.
5. Repeat steps 3 and 4 until all the bits are over.

Ex. 1.4 : Convert binary number 10101 to its decimal equivalent.

Sol. :

1 0 1 0 1

MSB LSB

Step 1 : 1 × 2 = 2

Step 2 : 2 + 0 = 2

Step 3 : 2 × 2 = 4

Step 4 : 4 + 1 = 5

.....

Step 3 : 5 × 2 = 10

Step 4 : 10 + 0 = 10

.....

Step 3 : 10 × 2 = 20

Step 4 : 20 + 1 = 21₁₀

Review Questions

1. Convert 67_{10} to binary using both methods.
2. Convert 813_{10} to binary using both methods and check your answer by converting back to decimal.

1.3.6 Fractional Decimal to Binary Conversion

In case of fractional decimal numbers, numbers are multiplied by 2 and carries are recorded from the integer position. These carries when read downward represent binary equivalent to fractional decimal number. This means that the first carry produced is the MSB and the last carry produced is the LSB.

Ex. 1.11 : Convert 0.8125 decimal number to its binary equivalent.

Sol. :	Fraction	Base	Result	Recorded Carries
	0.8125	$\times 2$	$= 1.625 = 0.625$ with a carry of	1 MSB
	0.625	$\times 2$	$= 1.25 = 0.25$ with a carry of	1
	0.25	$\times 2$	$= 0.5 = 0.5$ with a carry of	0
	0.5	$\times 2$	$= 1.0 = 0.0$ with a carry of	1 LSB

Reading carries downward we get,

binary fraction = 0.1101, which is equivalent to 0.8125 decimal.

Ex. 1.12 : Convert 0.95 decimal number to its binary equivalent

Sol.:	Fraction	Base	Result	Recorded Carries
	0.95	$\times 2$	$= 1.9 = 0.9$ with a carry of	1 MSB
	0.9	$\times 2$	$= 1.8 = 0.8$ with a carry of	1
	0.8	$\times 2$	$= 1.6 = 0.6$ with a carry of	1
	0.6	$\times 2$	$= 1.2 = 0.2$ with a carry of	1
	0.2	$\times 2$	$= 0.4 = 0.4$ with a carry of	0
	0.4	$\times 2$	$= 0.8 = 0.8$ with a carry of	0
	0.8	$\times 2$	$= 1.6 = 0.6$ with a carry of	1 LSB

In this case, 0.8 is repeated and if we multiply further, we will get repeated sequence. If we stop here, we get 7 binary digits, . 1111001. This answer is an approximate answer. To get more accurate answer we have to continue multiplying by 2 until we have as many digits as necessary for our application.

In case of unsigned 8-bit binary numbers the decimal range is 0 to 255. For signed magnitude 8-bit binary numbers the largest magnitude is reduced from 255 to 127 because we need to represent both positive and negative numbers.

Maximum positive number 0111 1111 = +127

Maximum negative number 1111 1111 = -128

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.

Ex. 1.17 : Find 1's complement of $(1\ 1\ 0\ 1)_2$

Sol. : 1 1 0 1 ← number

 0 0 1 0 ← 1's complement

Ex. 1.18 : Find 1's complement of 1 0 1 1 1 0 0 1

Sol. : 1 0 1 1 1 0 0 1 number

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

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

$$2's\ complement = 1's\ complement + 1$$

The 2's complement form is used to represent negative numbers.

Ex. 1.19 : Find 2's complement of $(1\ 0\ 0\ 1)_2$

Sol. : 1 0 0 1 number

 0 1 1 0 1's complement

 + 1

 0 1 1 1 2's complement

Ex. 1.20 : Find 2's complement of $(1\ 0\ 1\ 0\ 0\ 0\ 1\ 1)_2$

Sol. : 1 0 1 0 0 0 1 1 number

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

 + 1

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

CASE 4 : Both negative

Ex. 1.28 : Subtract -68 from -15

$$\begin{array}{r} \text{Sol. :} \\ (-15) \quad (-15) \\ - \quad \equiv \quad + \\ \hline (-68) \quad 68 \\ \hline 53 \quad 53 \end{array}$$

2's complement of 15

	0 0 0 0 1 1 1 1	number
	1 1 1 1 0 0 0 0	1's complement
+	1	
	1 1 1 1 0 0 0 1	2's complement
Addition :	1	← carry
	1 1 1 1 0 0 0 1	2's complement of 15
	+ 0 1 0 0 0 1 0 0	decimal 68
	① 0 0 1 1 0 1 0 1	decimal 53

In 8-bit addition carry after 8th bit is ignored. After addition of 2's complement of 15 (1 1 1 1 0 0 0 1) and 68 we get result (0 0 1 1 0 1 0 1)₂, which is equivalent to decimal 53 and our answer is verified.

Review Questions

- Determine the 1's complement of each binary number.
(a) 10011_2 (b) 111011_2 (c) 10001_2
- Determine the 2's complement of each binary number.
(a) 100_2 (b) 10000_2 (c) 111001_2
- Perform the following subtraction using the 2's complement method.
(a) $111011_2 - 101001_2$ (b) $100111_2 - 100010_2$

1.4 Octal Numbers System

We know that the base of the decimal number system is 10 because it uses the digits 0 to 9, and the base of binary number system is 2 because it uses digits 0 and 1. 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.

When we write an octal number say, 567, it can be represented in power of 8 as

$$\begin{array}{c} \boxed{5 \times 8^2} + \boxed{6 \times 8^1} + \boxed{7 \times 8^0} \\ \swarrow \quad \downarrow \quad \searrow \\ 5 \quad 6 \quad 7 \end{array}$$

To obtain the sum of multi-digit octal numbers, the procedure just described is applied to each column of digits as illustrated in the following example.

Ex. 1.38 : Add 167_8 and 325_8

$$\begin{array}{r} \text{Sol. :} \\ 11 \quad \leftarrow \text{Carry} \\ 167 \\ + 325 \\ \hline 514_8 \end{array}$$

If the decimal sum of several octal digits is 16 or greater, subtract 16 and set carry equal to 2. In general, we can express any decimal sum in octal by repeatedly subtracting 8 until the result is one of the octal digits through 0 to 7. Each time 8 is subtracted, the amount of the carry is increased by 1. This procedure is illustrated in the following example.

Ex. 1.39 : Add the octal numbers 341_8 , 125_8 , 472_8 and 577_8

$$\begin{array}{r} \text{Sol. :} \\ 121 \quad \leftarrow \text{Carry} \\ 341 \\ + 125 \\ + 472 \\ + 577 \\ \hline 1757_8 \end{array}$$

The better way to perform octal subtraction is to convert the numbers to binary, perform the subtraction, and convert the result back to octal. However, the complement methods already described for binary can also be used. The 7's and 8's complements for octal numbers are found and used like 1's and 2's complements to perform subtraction.

Subtraction with 7's complement

The 7's complement of an octal number is found by subtracting each digit from 7, as illustrated in Ex. 1.40.

Ex. 1.40 : Find 7's complement of 612_8

$$\begin{array}{r} \text{Sol. :} \\ 777 \\ - 612 \\ \hline 165_8 \end{array}$$

The steps for octal subtraction using 7's complement method are as given below :

Step 1 : Find 7's complement of subtrahend

Step 2 : Add two octal numbers (first number and 7's complement of the second number)

Step 3 : If carry is produced in the addition, add carry in the least significant bit of the sum; otherwise find 7's complement of the sum as a result with negative sign.

Ex. 1.41 : Use the 7's complement method of subtraction to compute $176_8 - 157_8$

$$\begin{array}{r} \text{Sol. :} \\ \text{Step 1 :} \\ 777 \\ - 157 \\ \hline 620 \end{array} \quad \text{7's complement}$$

$$\begin{aligned}
 &= 3 \times 256 + F(15) \times 16 + D(13) \times 1 \\
 &= 768 + 240 + 13 \\
 &= 1021_{10}
 \end{aligned}$$

1.5.1 Hexadecimal to Decimal Conversion

The hexadecimal number can be converted into decimal number using following steps.

1. Multiply most significant digit (leftmost digit) by 16.
2. Add next adjacent digit to the result.
3. Multiply result by 16.
4. Add next adjacent digit to the result.
5. Repeat steps 3 and 4 until all the digits are over.

Example 1.46 : Convert hexadecimal number 7E4H to its decimal equivalent.

Sol. :

$$\begin{aligned}
 \text{Step 1 :} & \quad 7 \times 16 = 112 \\
 \text{Step 2 :} & \quad 112 + E \text{ (Decimal 14)} = 126 \\
 \text{Step 3 :} & \quad 126 \times 16 = 2016 \\
 \text{Step 4 :} & \quad 2016 + 4 = 2020_{10}
 \end{aligned}$$

1.5.2 Fractions

In the above section we have considered only hexadecimal integer numbers. In case of hexadecimal fraction numbers, we have to see the weights of digit positions to the right of the decimal point. These are given as $1/16$, $1/256$, $1/4096$ and so on. In power of 16, the weights are 16^{-1} , 16^{-2} , 16^{-3} , 16^{-4} and so on. These weights may be written in decimal form as 0.0625, 0.0039 and so on.

Ex. 1.47 Convert hexadecimal number 0.582_H to its decimal equivalent

$$\begin{aligned}
 \text{Sol. :} \quad 0.582_H &= 5 \times 16^{-1} + 8 \times 16^{-2} + 2 \times 16^{-3} \\
 &= 0.3125 + 0.03125 + 4.8828 \times 10^{-4} \\
 &= 0.34424_{10}
 \end{aligned}$$

1.5.3 Mixed Numbers

In general mixed (numbers with an integer and a fractional part) hexadecimal numbers can be represented as

$$N = A_n 16^n + A_{n-1} 16^{n-1} + \dots + A_1 16^1 + A_0 16^0 + A_{-1} 16^{-1} + A_{-2} 16^{-2} + \dots$$

where $N = \text{Number}$

$A = \text{Digit}$

$n = \text{Digit number}$

Ex. 1.48 : Convert $(9B2.1A)_H$ to its decimal equivalent

$$\begin{aligned}
 \text{Sol. :} \quad N &= 9 \times 16^2 + B(11) \times 16^1 + 2 \times 16^0 + 1 \times 16^{-1} + A(10) \times 16^{-2} \\
 &= 2304 + 176 + 2 + 0.0625 + 0.039 \\
 &= 2482.1_{10}
 \end{aligned}$$

$$\begin{array}{r}
 \text{Sol. :} \quad 2 \ 1 \ 2 \quad \leftarrow \text{Carry} \\
 \quad \quad 4 \ F \ B \\
 \quad \quad 7 \ 5 \ D \\
 \quad \quad A \ 1 \ 2 \\
 \quad \quad C \ 3 \ 9 \\
 \hline
 \quad \quad 2 \ 2 \ A \ 3_H
 \end{array}$$

Hexadecimal subtraction is best accomplished using the complement method. The 15's and 16's complements for hex numbers are found and used like 1's and 2's complements to perform subtraction.

Subtraction with 15's Complement

The 15's complement of a hexadecimal number is found by subtracting each digit from 15, as illustrated in Ex. 1.59.

Ex. 1.59 : Find 15's complement of $A9B_H$

$$\begin{array}{r}
 \text{Sol. :} \quad 15 \ 15 \ 15 \\
 \quad \quad - \ A \ 9 \ B \\
 \hline
 \quad \quad 5 \ 6 \ 4_H
 \end{array}$$

The steps for hexadecimal subtraction using 15's complement method are as given below :

Step 1 : Find 15's complement of subtrahend

Step 2 : Add two hexadecimal numbers (first number and 15's complement of the second number.)

Step 3 : If carry is produced in the addition, add carry to the least significant bit of the sum; otherwise find 15's complement of the sum as a result with a negative sign.

Ex. 1.60 : Use the 15's complement method of subtraction to compute $B02_H - 98F_H$

$$\begin{array}{r}
 \text{Sol. :} \quad \text{Step 1:} \quad 15 \ 15 \ 15 \\
 \quad \quad \quad - \ 9 \ 8 \ F \\
 \quad \quad \quad \hline
 \quad \quad \quad 6 \ 7 \ 0 \quad \leftarrow \text{15's complement} \\
 \\
 \text{Step 2:} \quad \quad \quad B \ 0 \ 2 \\
 \quad \quad \quad + \ 6 \ 7 \ 0 \\
 \quad \quad \quad \hline
 \text{carry} \rightarrow 1 \ 1 \ 7 \ 2 \\
 \\
 \text{Step 3:} \quad \quad \quad 1 \ 7 \ 2 \\
 \quad \quad \quad + \quad \quad 1 \\
 \quad \quad \quad \hline
 \quad \quad \quad 1 \ 7 \ 3
 \end{array}$$

$$\therefore B02_H - 98F_H \rightarrow 173_H$$

Ex. 1.68 : Divide 110101101_2 by 101_2

Sol. :

$$\begin{array}{r}
 1010101 \\
 101 \overline{) 110101101} \\
 \underline{-101} \\
 00110 \\
 \underline{-101} \\
 00111 \\
 \underline{-101} \\
 01001 \\
 \underline{-101} \\
 0100
 \end{array}$$

Solved Examples

Ex. 1.69 : Convert the following binary numbers to Decimal, Hexadecimal and Octal form :

i) $(101101.1101)_2$

ii) $(11011011.100101)_2$

Sol. : i) $(101101.1101)_2$

Decimal Conversion

$$\begin{aligned}
 &= 2^5 \times 1 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 \\
 &\quad + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\
 &= 32 + 8 + 4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} \\
 &= 45 + 0.5 + 0.25 + 0.0625 \\
 &= 45 + 0.8125 \\
 &= 45.8125
 \end{aligned}$$

$$(101101.1101)_2 = (45.8125)_{10}$$

Hex Conversion : 10 1101 . 1101

$$2 \quad D \quad . \quad D = 2D.DH$$

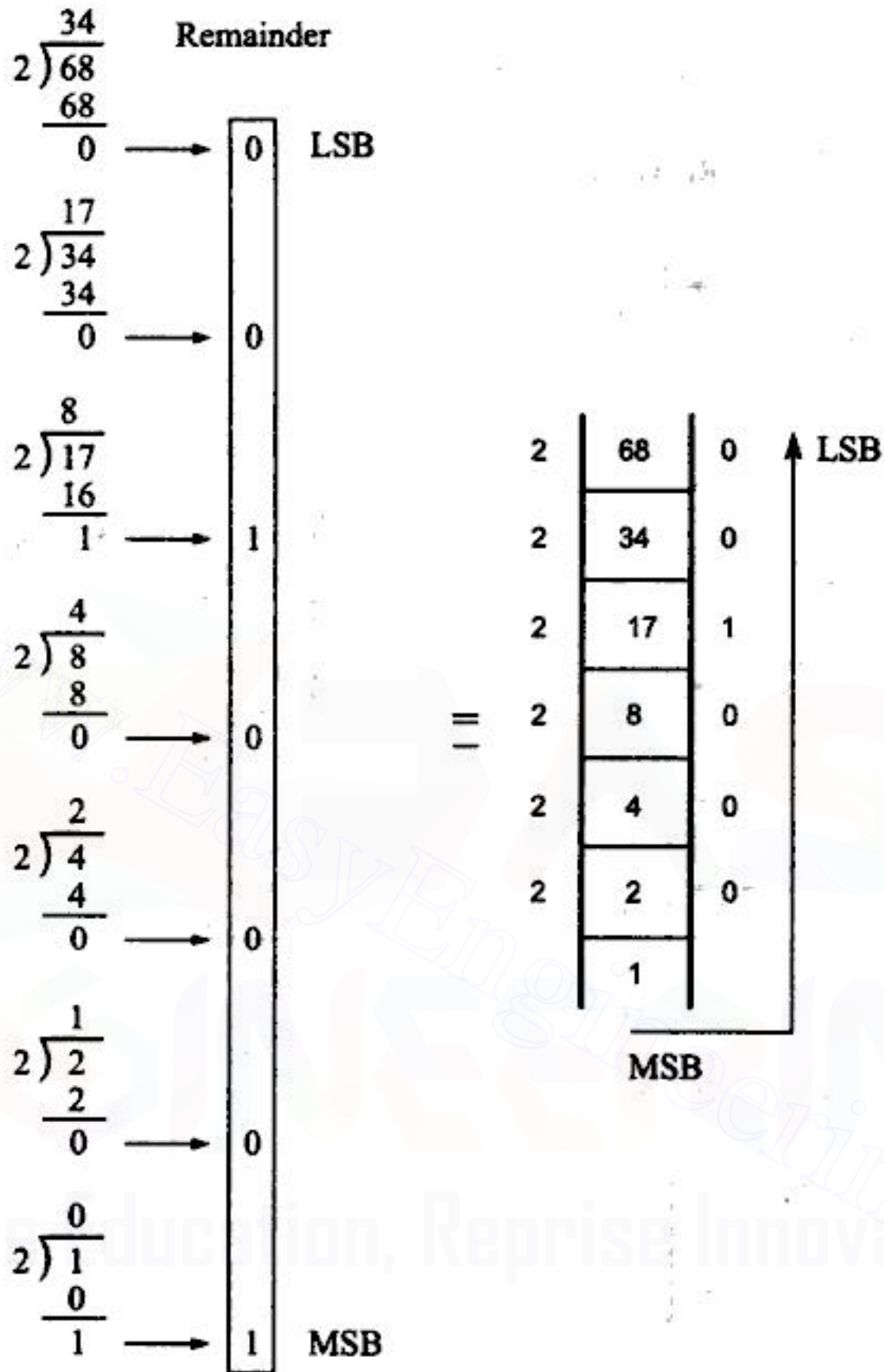
Octal Conversion : 101 101 . 110 1

$$5 \quad 5 \quad . \quad 6 \quad 4 = 55.64_8$$

ii) $(11011011.100101)_2$

Decimal Conversion :

$$\begin{aligned}
 &= 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 \\
 &\quad + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \\
 &\quad + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\
 &\quad + 0 \times 2^{-5} + 1 \times 2^{-6} \\
 &= 128 + 64 + 0 + 16 + 8 + 0 + 2 + 1
 \end{aligned}$$



Step 4: Add $(1\ 0\ 1\ 0\ 1\ 0)_2$ and $(0\ 1\ 1\ 1\ 1\ 0\ 0)_2$

	111	← Carry
	0101010	
+	0111100	
	1100110	

Since we have subtracted larger number from smaller number result is negative and hence it is necessary to take 2's complement of this result.

Since final carry is zero, so the answer is negative and in 2's complement form.

$$\therefore (1000100)_2 - (1010100)_2 = (0010000)_2$$

Ex. 1.82 : Perform the following operation

$$(737)_8 - (123)_{16} + (100)_{10}$$

Write the answer in hexadecimal form

(Dec-99)

Sol. : $(737)_8 = (111011\ 11)_2 = (1\ D\ F)_{16}$

16	100	4
	6	

$$\therefore (100)_{10} = 64_H$$

Now

$$\begin{array}{r} 1\ D\ F \\ -1\ 2\ 3 \\ \hline 0\ B\ C \\ +\ 6\ 4 \\ \hline 1\ 2\ 0 \end{array}$$

$$\therefore (737)_8 - (123)_{16} + (100)_{10} = 120_H$$

Ex. 1.83 : Convert the following decimal numbers to binary

i) $(12.0625)_{10}$

ii) $(41.375)_{10}$

(Dec-99)

Sol. : i)

2	12	0	LSB
2	6	0	↑
2	3	1	↑
	1		↑
			↑

MSB

$$0.0625 \times 2 = 0.125 \quad 0$$

$$0.125 \times 2 = 0.25 \quad 0$$

$$0.25 \times 2 = 0.5 \quad 0$$

$$0.5 \times 2 = 0 \quad 1$$

$$\therefore (12.0625)_{10} = (1\ 1\ 0\ 0.0\ 0\ 0\ 1)_2$$

6. Assign a binary code in some orderly manner to the 52 playing cards. Use the minimum number of bits.
(May-92)
7. Convert the following decimal numbers into binary numbers.
i) -128 ii) 257
(Dec-92)
Ans. : i) 10000000 (in 2's complement form), ii) 10000001
8. Solve the following
i) $A5_H + 2C_H = ()_{10}$ ii) $1101 \times 1001 = \dots$
(Dec-92)
Ans. : i) 209, ii) 111 01 01
9. Determine 2's complement of
i) $8F_H$ ii) FF_H
(Dec-92)
Ans. : i) 1000 0001, ii) 0000 0001
10. Determine decimal equivalent of the following in sign magnitude form
i) $8D_H$ ii) FF_H
(Dec-92)
Ans. : i) -13, ii) -127
11. i) Add the hexadecimal numbers :
 $FE1 + CA9$
ii) Perform the subtraction of hexadecimal numbers
 $FA7 - 1FD$
(May-93)
Ans. : i) 1C 8A, ii) DAA
12. Carry out the following :
i) Convert $(11001011.01110)_2$ into decimal
ii) Convert decimal number (61.3) to binary (upto 5 bit binary)
iii) Convert octal number 574 to binary and decimal.
iv) Convert hex number A92 to octal.
(May-93)
Ans. : $(203.4375)_{10}$, ii) $(111101.010011001)_2$, iii) $(101\ 111\ 100)_2$, 380 iv) $(5222)_8$
13. Determine 2's complement of
i) 77_H , ii) FE_H
(May-93)
Ans. : i) 89_H , ii) 02_H
14. Carry out the following
 $(756)_8 + (125)_8 - (176)_8 = (\dots ? \dots)_8$
(May-93)
Ans. : $(705)_8$
15. Convert the following numbers to hexadecimal.
i) $(365)_8$ ii) $(22.62)_{10}$ iii) $(11011.1101)_2$ iv) $(10.1)_2$
(Dec-93)
Ans. : i) $(0F5)_{16}$, ii) $(16.9)_{16}$, iii) $(1B.D)_{16}$ iv) $(2.8)_{16}$
16. Carry out the following arithmetic operations and represent result in the same format.
i) $(621)_8 - (217)_8$ ii) $(F4\ D2)_{16} - (91\ FA)_{16}$
(Dec-93)
Ans. : i) $(402)_8$, ii) $(62\ D8)_{16}$
17. Using 2's complement perform the following :
i) $(42)_{10} - (68)_{10}$ ii) $(25) - (16)_{10}$
(Dec-93)
What are the advantages of using 2's complement method over 1's complement
Ans. : $(-26)_{10}$, ii) $(9)_{10}$
18. Solve the following.
i) 1101.01×100.1 ii) $111110.1 + 0101$
(Dec-93)
Ans. : i) 1110111.01, ii) 1100.1

64. Perform the following subtraction using 1's complement method :
 $(0011.1001) - (0001.1110)$ (May-2000)
65. (a) Perform the following :
 (i) Convert to binary : $(19.75)_{10} = (\quad)_2$
 (ii) Convert to decimal : $(F4D2)_{16} = (\quad)_{10}$ (Dec-2000, May-2002)
66. Performing the following : $(5531)_8 - (3261)_8 + (100)_{10}$
 Write the answer in hexadecimal form. (Dec-2000, May-2002)
67. Convert the following numbers from the given base to other three bases indicated :
 1) Decimal 52.45 to binary, octal and hex. **Ans. : $(64.3463)_8, (110100.011100110011)_2, (34.733)_{16}$**
 2) Hexadecimal 2AC5 to binary, octal and decimal. **Ans. : $(0010101011000101)_2, (25305)_8, (10949)_{10}$**
(May-2001)
68. Perform the following subtraction using 1s complement and 2s complement method $45 + (-17)$
 Give the result in sign-magnitude form. (May-2001)
69. Determine the values of base x if : (Dec-2001)
 (i) $(211)_x = (152)_8$ (ii) $(193)_x = (623)_8$ **Ans. : i) $x = 7$ ii) $x = 16$**
70. Using 2's complement method, perform : (Dec-2001)
 (i) $(156)_{10} - (99)_{10}$ (ii) $(16)_{10} - (25)_{10}$
71. Carry out the following conversions : $(268.75)_{10}$ into binary, octal and hexadecimal.
Ans. : $(100001100.11)_2, (414.6)_8, (10C.C)_{16}$
(Dec-2001)
72. Determine the base x : $(211)_x = (152)_8$ (May-2002)
73. What do you mean by sign magnitude form of representation ? (May-2002)
74. Convert the following numbers into decimal :
 i) 100100111000.0111 (BCD), ii) 11001101.111_2 , iii) $CF.5_{16}$, iv) 234_5 . (Dec.-2002)
75. Write sign-magnitude, 1's complement and 2's complement forms for the following decimal numbers :
 i) $+ 9.5$ ii) $- 17.0$ (May-2003)
76. Perform the indicated operations : i) $(FDC)_{16} \div (A29)_{16}$ ii) $(241)_8 - (176)_8$ (May-2003)
77. Perform the following subtractions using 2's complement method :
 i) $0011.1001 - 0001.1110$ ii) $(3F)_{16} - (5C)_{16}$ (Dec.-2003)
78. Give 4-bit sign-magnitude, 1's complement and 2's complement representation of a number in a tabular form. (Dec.-2003)



2.3.1 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.

Sum equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

$$\begin{array}{r}
 6 \quad 0110 \quad \leftarrow \text{BCD for 6} \\
 + 3 \quad 0011 \quad \leftarrow \text{BCD for 3} \\
 \hline
 9 \quad 1001 \quad \leftarrow \text{BCD for 9}
 \end{array}$$

The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

$$\begin{array}{r}
 6 \quad 0110 \quad \leftarrow \text{BCD for 6} \\
 + 8 \quad 1000 \quad \leftarrow \text{BCD for 8} \\
 \hline
 14 \quad 1110 \quad \leftarrow \text{Invalid BCD number } (1110) > 9
 \end{array}$$

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

$$\begin{array}{r}
 6 \quad 0110 \quad \leftarrow \text{BCD for 6} \\
 + 8 \quad 1000 \quad \leftarrow \text{BCD for 8} \\
 \hline
 14 \quad 1110 \quad \leftarrow \text{Invalid BCD number} \\
 + \quad 0110 \quad \leftarrow \text{Add 6 for correction} \\
 \hline
 \underbrace{0001}_1 \quad \underbrace{0100}_4 \quad \leftarrow \text{BCD for 14}
 \end{array}$$

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

Sum equals 9 or less with carry 1

Let us consider addition of 8 and 9 in BCD

$$\begin{array}{r}
 8 \quad 1000 \quad \leftarrow \text{BCD for 8} \\
 + 9 \quad 1001 \quad \leftarrow \text{BCD for 9} \\
 \hline
 17 \quad 0001 \quad 0001 \quad \leftarrow \text{Incorrect BCD result}
 \end{array}$$

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

From the above examples we can summarize steps for 9's complement BCD subtraction as follows :

- Find the 9's complement of a negative number
- Add two numbers using BCD addition
- If carry is generated add carry to the result otherwise find the 9's complement of the result.

Ex. 2.2 : Perform each of the following decimal subtractions in 8-4-2-1 BCD using 9's complement method.

$$\begin{array}{r} a) 79 \\ - 26 \\ \hline \end{array}$$

$$\begin{array}{r} b) 89 \\ - 54 \\ \hline \end{array}$$

Sol. : a) $\begin{array}{r} 79 \\ - 26 \\ \hline \end{array}$ $\begin{array}{r} 0111 \\ + 0111 \\ \hline \end{array}$ $\begin{array}{r} 1001 \\ 0011 \\ \hline \end{array}$ 73 – 9's complement for BCD 26

$$\begin{array}{r} 53 \\ \hline \end{array}$$

$$\begin{array}{r} 1110 \\ 1100 \\ \hline \end{array}$$

0110 1100 > 9 so add 6

$$\begin{array}{r} 1110 \\ + 1 \\ \hline \end{array}$$

$$\begin{array}{r} 10010 \\ + 1 \\ \hline \end{array}$$

Propagate carry

$$\begin{array}{r} 1111 \\ + 0110 \\ \hline \end{array}$$

$$\begin{array}{r} 10101 \\ + 0010 \\ \hline \end{array}$$

Add 6

$$\begin{array}{r} 10101 \\ + 1 \\ \hline \end{array}$$

$$\begin{array}{r} 0101 \\ + 1 \\ \hline \end{array}$$

End around carry

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

BCD for 53

b) $\begin{array}{r} 89 \\ - 54 \\ \hline \end{array}$ $\begin{array}{r} 1000 \\ 0100 \\ \hline \end{array}$ $\begin{array}{r} 1001 \\ 0101 \\ \hline \end{array}$

0100 (45) 9's complement of 54 BCD

$$\begin{array}{r} 35 \\ \hline \end{array}$$

$$\begin{array}{r} 1100 \\ 1110 \\ \hline \end{array}$$

$$\begin{array}{r} + 0110 \\ \hline \end{array}$$

1110 > 9 so add 6

$$\begin{array}{r} 1100 \\ + 1 \\ \hline \end{array}$$

$$\begin{array}{r} 10100 \\ + 1 \\ \hline \end{array}$$

Propagate carry

$$\begin{array}{r} 1101 \\ + 0110 \\ \hline \end{array}$$

$$\begin{array}{r} 0110 \\ \hline \end{array}$$

Add 6

$$\begin{array}{r} 10011 \\ + 1 \\ \hline \end{array}$$

$$\begin{array}{r} 00100 \\ + 1 \\ \hline \end{array}$$

End around carry

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

BCD for 35

Ex. 2.4 :

a)

$$\begin{array}{r}
 1011 \rightarrow \text{Excess-3 for 8} \\
 + 1001 \rightarrow \text{Excess-3 for 6} \\
 \hline
 10100 \\
 \text{Add 3} \quad 00110011 \\
 \hline
 01000111 \rightarrow \text{Excess-3 for 14} \\
 \text{(1) \quad (4)}
 \end{array}$$

b)

$$\begin{array}{r}
 0100 \rightarrow \text{Excess-3 for 1} \\
 + 0101 \rightarrow \text{Excess-3 for 2} \\
 \hline
 1001 \\
 \text{Sub 3} \quad - 0011 \\
 \hline
 0110 \rightarrow \text{Excess-3 for 3}
 \end{array}$$

2.5.2 Excess-3 Subtraction

To perform Excess-3 subtraction we have to

- Complement the subtrahend
- Add complemented subtrahend to minuend
- If carry = 1 Result is positive. Add 3 and end around carry
- If carry = 0 Result is negative. Subtract 3.

Ex. 2.5 : a) 8 - 5

$$\begin{array}{r}
 1011 \rightarrow \text{Excess-3 for 8} \\
 + 0111 \rightarrow \text{Complement of 5 in Excess-3} \\
 \hline
 10010 \\
 \text{Add 3} \quad 0011 \\
 \hline
 0101 \\
 \text{Add end around carry} \quad \rightarrow 1 \\
 \hline
 0110 \rightarrow \text{Excess-3 for 3}
 \end{array}$$

b) 5-8

$$\begin{array}{r}
 1000 \rightarrow \text{Excess-3 for 5} \\
 + 0100 \rightarrow \text{Complement of 8 in Excess-3} \\
 \hline
 1100 \\
 - 0011 \\
 \hline
 1001 \rightarrow \text{Excess-3 for } -3
 \end{array}$$

180° error in the disk position. Since it is physically impossible to have all the brushes precisely aligned, some error will always be present at the edges of the sectors.

If we use gray code to represent disk position then error due to improper brush alignment can be reduced. This is because the gray code assures that only one bit will change each time the decimal number is incremented. So in 3-bit code, error may occur due to one bit position. Other two bits positions of two adjacent sectors are always same and hence there is no possibility of error. Therefore in 3-bit code probability of error is reduced upto 66%. In case of 4-bit code it is reduced upto 75%.

2.6.2 Gray-to-Binary Conversion

The gray to binary code conversion can be achieved using following steps.

1. The most significant bit of the binary number is the same as the most significant bit of the gray code number. So write it down.
2. 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.
3. Repeat step 2 until all gray code bits have been exclusive-ORed with binary digits. The sequence of bits that have been written down is the binary equivalent of the gray-code number.

Ex. 2.8 : Convert gray code 101011 into its binary equivalent.

Sol. :

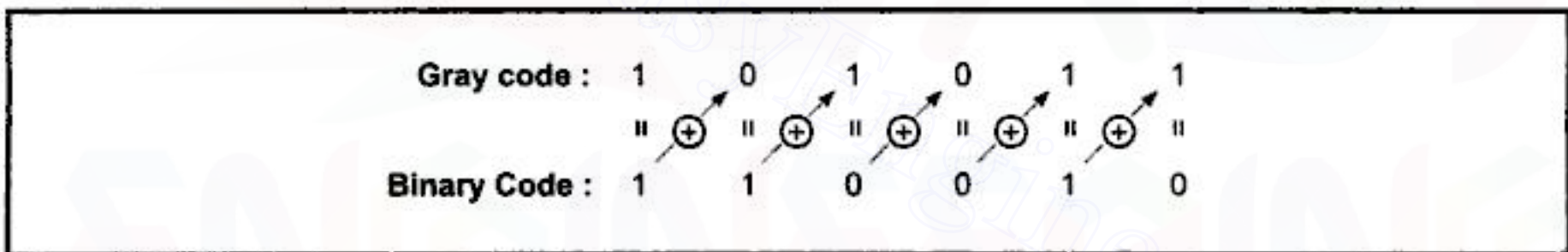


Fig. 2.3

2.6.3 Binary to Gray Conversion

Let us represent binary number as

$B_1 B_2 B_3 B_4 \dots B_n$ and its equivalent gray code as

$G_1 G_2 G_3 G_4 \dots G_n$.

With this representation gray code bits are obtained from the binary bits as follows :

$$\begin{aligned}
 G_1 &= B_1 \\
 G_2 &= B_1 \oplus B_2 \\
 G_3 &= B_2 \oplus B_3 \\
 G_4 &= B_3 \oplus B_4 \\
 &\vdots \\
 G_n &= B_{n-1} \oplus B_n
 \end{aligned}$$

1010		CC	SM		α	!	:								
1011						\$,	#							
1100					<	*	%	@							
1101					()	-	'							
1110					+	;	>	=							
1111	CU1	CU2	CU3				?	"							

Table 2.9 Partial EBCDIC table

Definitions of control abbreviations :

BS	Backspace	LC	Lowercase	meanings of unfamiliar symbols : Vertical bar : logical OR ¬ Logical NOT - Hyphen or minus sign _ Underscore (01101101) Example of code format : 01234567 Bit positions 11000110 is F
BYP	Bypass	LF	Line feed	
CC	Cursor control	NL	New line	
CU1	Customer use	PF	Punch off	
CU2	Customer use	PN	Punch on	
CU3	Customer use	PRE	Prefix	
DL	Delete	RES	Restore	
DS	Digit select	RS	Reader stop	
EOB	End of block	SM	Set Mode	
EOT	End of transmission	SP	Space	
FS	Field separator	TM	Tape mark	
HT	Horizontal tab	UC	Uppercase	
IL	Idle			

2.8.3 Hollerith Code

Herman Hollerith developed a way to feed information into the digital computer using punched cards. The code used in the system to represent alphanumeric information is known as **Hollerith code**. A Hollerith card consists of 80 columns and 12 rows as shown in the Fig. 2.5. Each column represents an alphanumeric character with holes in the appropriate rows. A hole is sensed as a 1 and the absence of a hole is sensed as a 0 by the circuit in the card reader. The 12 rows are marked starting from the top as 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Each row is a 1-bit information. Thus the Hollerith code is 12-bit code. The first three rows of the card are called the **zone punch rows** and remaining nine row are called the **numeric punch rows**. The numbers are represented in the column by a single punch. For example, punch on 5 represents number 5. The alphabets are represented using two punches. A letter A is a 12-1 punch, i.e. punch in row 1 and row 12. For example,

$$B = 12 - 2, C = 12 - 3, \dots, I = 12 - 9,$$

$$J = 11 - 1, K = 11 - 2, \dots, R = 11 - 9.$$

Ex. 2.4 : Convert each pair of decimal number to BCD and add.

i) $(65)_{10} + (58)_{10}$

ii) $(113)_{10} + (101)_{10}$

Sol. :

65	0 1 1 0	0 1 0 1	BCD for 65
+ 58	+ 0 1 0 1	1 0 0 0	BCD for 58
123	1 0 1 1	1 1 0 1	1101 > 9 so
	+	0 1 1 0	Add 6
	1 0 1 1	1 0 0 1 1	
	+ 1		Propagate carry to next digit
	1 1 0 0	0 0 1 1	1100 > 9 so
	+	0 1 1 0	Add 6
000 1 0 0 1 0	0 0 1 1		BCD for 123

ii) 113	0 0 0 1	0 0 0 1	0 0 1 1	BCD for 113
+ 101	0 0 0 1	0 0 0 0	0 0 0 1	BCD for 101
214	0 0 1 0	0 0 0 1	0 1 0 0	BCD for 214

Ex. 2.5 : Convert each of the following decimal numbers to excess-3 code

i) $(18)_{10}$ ii) $(56)_{10}$

Sol. : i) $18_{10} = 0100 \ 1011$ in excess-3

ii) $56_{10} = 1000 \ 1001$ in excess-3

Ex. 2.6 : Convert gray number 1110 to its BCD equivalent.

Sol. :

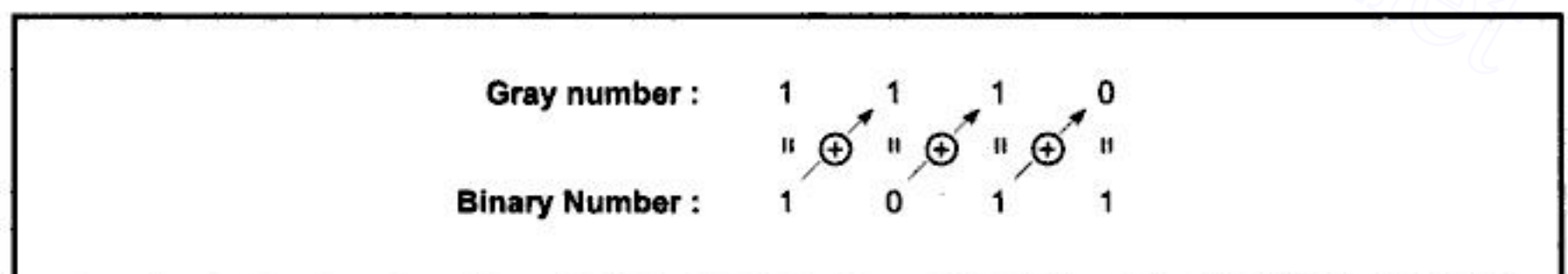


Fig. 2.8

$$\therefore 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11_{10}$$

$$11_{10} = 0001 \ 0001 \text{ in BCD}$$

$$\therefore 1110 \text{ in gray} = 0001 \ 0001 \text{ in BCD}$$

29. Convert the given numbers into BCD and subtract (35) – (19) (May-97)
30. Add the following Excess-3 numbers expressing the result in Excess-3.
1) (0011) + (0100) 2) (0101) + (1001). (May-97)
31. Convert Gray code (10010111) to Binary. (May-97)
Ans. : 11100101
32. Convert Binary (110101011) to Gray code. (May-97)
Ans. : 101111110
33. Convert (567) to 5211 BCD code and 8 4 $\bar{2}$ 1 code. (May-97)
Ans. : (1000 1010 1100) 5211, (1011 1010 1001) 84 2 1
34. i) Convert a Binary number 10010011 to Gray code. (Dec-97)
ii) Convert a gray 1001011 to binary number. Ans. : i) 11011010, ii) 1110010
35. Perform the following subtraction (Dec-97)
(46) – (18) Ans. : 28 (BCD)
36. For the weighted code 4, 4, 3, – 2 for decimal digits, determine all possible tables so that 9's complement of each decimal digit is obtained by changing 1's to 0's and 0's to 1's. (May-98)
37. Using Excess-3 code, add the following decimal numbers and express the answer in Excess-3 form : (May-98)
(i) 432 + 127 (ii) 372 + 437.
Ans. : i) 1000 1000 1100, ii) 10110011 1100
38. Convert following numbers into Gray code : (May-98)
i) (47) ii) (AB)
iii) (16) iv) (1110111) Ans. : i) 110100, ii) 11111110, iii) 1100, iv) 1001100
39. What are excess-3 and gray code ? Express decimal 1 to 9 in gray code excess-3 code and 8421 code. (Dec-98)
40. What is Gray code ? Give the advantage of gray code over binary code. (May-99)
41. Write the ASCII code for 'ELECTRONICS' (May-99)
42. Convert Binary number 1011100010 to its Gray Code. (May-2000)
43. Encode the word 'BINARY' in ASCII form. (May-2000)
44. Write short note on BCD Addition (May-2000)
45. Explain giving a suitable example the difference between weighted code and non-weighted code. (Dec-2000)
46. What do you mean by ASCII codes ? Give one application. (Dec-2000)
47. With suitable examples explain weighted and non-weighted codes. (May-2001)
48. What are excess-3 and gray code? Express decimal 1 to 9 in gray code, excess-3 code and 8221 code. (Dec-2001)
49. What do you mean by self complementing code ? Write two self complementing codes. (May-2002)
50. Represent the following decimal numbers in each of the given codes using 12-bit notation :
i) 491 ii) 27
Codes : i) 8421 ii) 5421 iii) 2421 (May-2003)



3.4 Block Parity

When several binary words are transmitted or received in succession, the resulting collection of bits can be regarded as a block of data, having rows and columns. For example, four eight bit words in succession form an 4×8 block. Parity bits can then be assigned to both rows and columns, as shown in Fig. 3.1 (a). This scheme is known as **block parity**. It makes it possible to correct any single error occurring in a data word and to detect any two errors in a word. Let us see how we can do this.

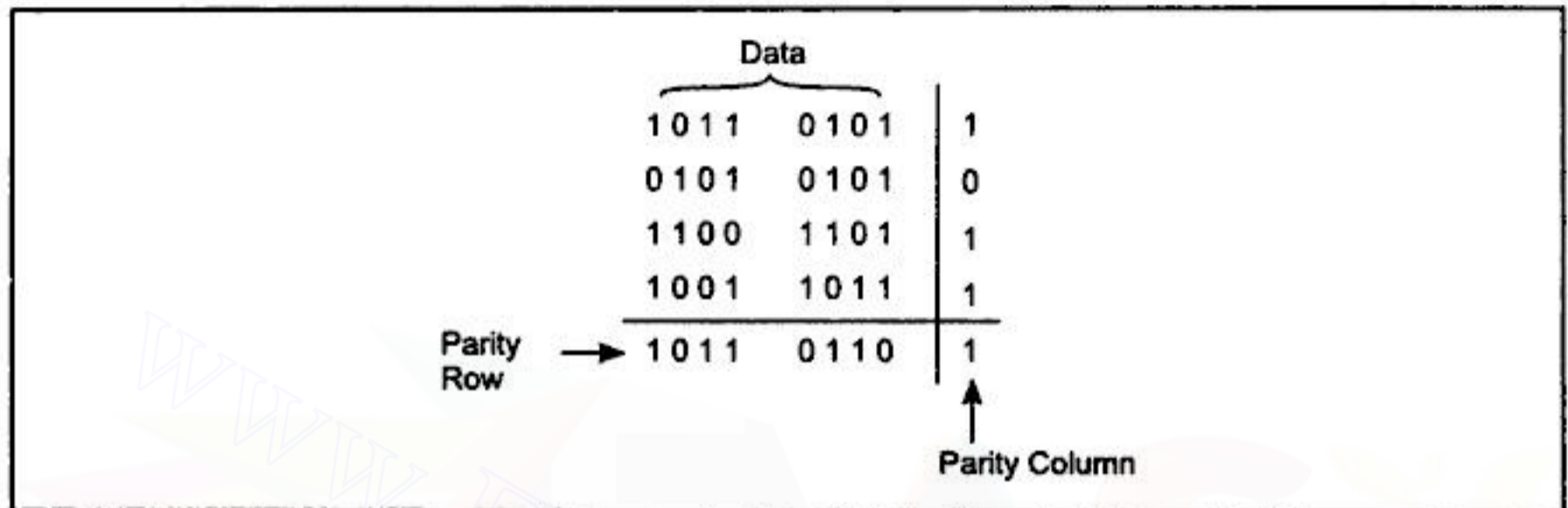


Fig. 3.1 (a) Adding even parity bits to the rows and columns of a 4×8 data block

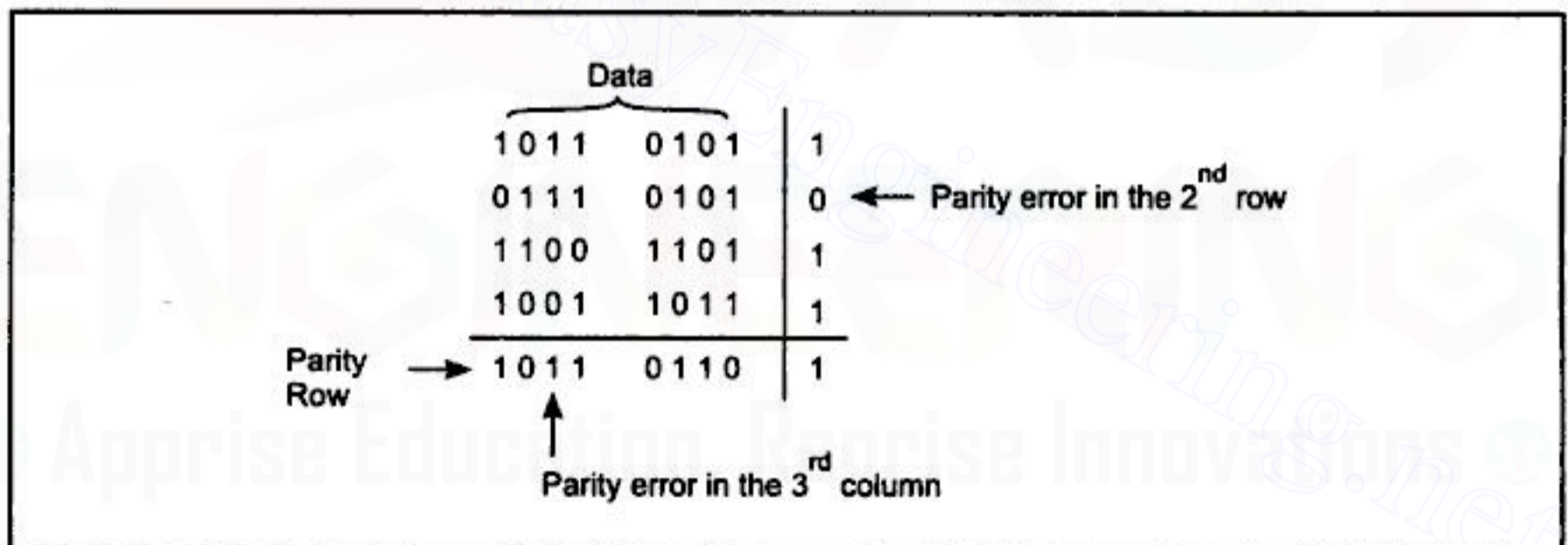


Fig. 3.1 (b) Parity errors in the third column and second row mean that the third bit in the second word is in error, so it can be corrected.

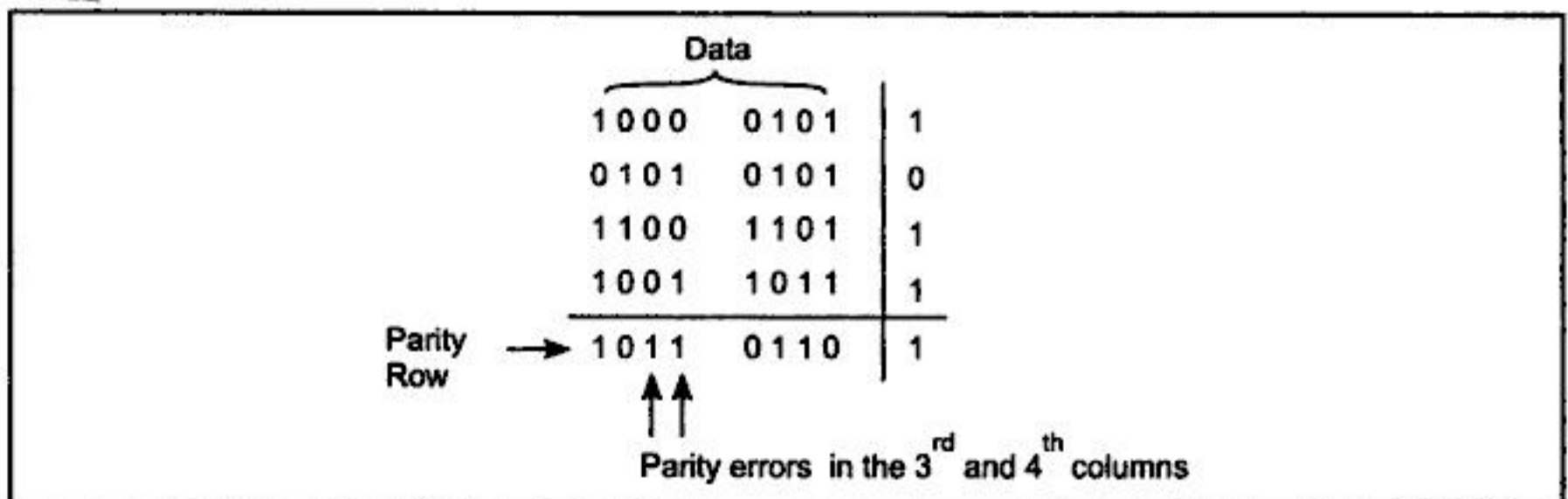


Fig. 3.1 (c) Parity errors in two columns mean that two errors have occurred in one data word

Sol.: Here $n = 6$ and $k = 3$. Therefore message code is 3 bit having eight combinations (000 - 111). As $n = 6$, check bits are $n - k = 3$.

Code for message 0 0 0 :

We know that,

$$\begin{aligned}
 C &= DG \\
 &= [000] \begin{bmatrix} 1 & 0 & 0 & : & 1 & 0 & 0 \\ 0 & 1 & 0 & : & 1 & 0 & 1 \\ 0 & 0 & 1 & : & 1 & 1 & 1 \end{bmatrix} \\
 &= [0 \cdot 1 \oplus 0 \cdot 0 \oplus 0 \cdot 0 \quad 0 \cdot 0 \oplus 0 \cdot 1 \oplus 0 \cdot 0 \quad 0 \cdot 0 \oplus 1 \cdot 0 \\
 &\quad \oplus 0 \cdot 1 \quad 0 \cdot 1 \oplus 0 \cdot 1 \oplus 0 \cdot 1 \quad 0 \cdot 0 \oplus 0 \cdot 0 \oplus 0 \cdot 1 \\
 &\quad 0 \cdot 0 \oplus 1 \cdot 1 \oplus 0 \cdot 1] \\
 &= [000000]
 \end{aligned}$$

Similarly we have,

Code for message 0 0 1 :

$$\begin{aligned}
 C &= [001] \begin{bmatrix} 1 & 0 & 0 & : & 1 & 0 & 0 \\ 0 & 1 & 0 & : & 1 & 0 & 1 \\ 0 & 0 & 1 & : & 1 & 1 & 1 \end{bmatrix} \\
 &= 001 \ 111
 \end{aligned}$$

Code for message 0 1 0 :

$$\begin{aligned}
 C &= [010] \begin{bmatrix} 1 & 0 & 0 & : & 1 & 0 & 0 \\ 0 & 1 & 0 & : & 1 & 0 & 1 \\ 0 & 0 & 1 & : & 1 & 1 & 1 \end{bmatrix} \\
 &= 010 \ 101
 \end{aligned}$$

Code for message 0 1 1 :

$$\begin{aligned}
 C &= [011] \begin{bmatrix} 1 & 0 & 0 & : & 1 & 0 & 0 \\ 0 & 1 & 0 & : & 1 & 0 & 1 \\ 0 & 0 & 1 & : & 1 & 1 & 1 \end{bmatrix} \\
 &= 011010
 \end{aligned}$$

Code for message 1 0 0 :

$$\begin{aligned}
 C &= [100] \begin{bmatrix} 1 & 0 & 0 & : & 1 & 0 & 0 \\ 0 & 1 & 0 & : & 1 & 0 & 1 \\ 0 & 0 & 1 & : & 1 & 1 & 1 \end{bmatrix} \\
 &= 100 \ 100
 \end{aligned}$$

Code for message 1 0 1 :

$$\begin{aligned}
 C &= [101] \begin{bmatrix} 1 & 0 & 0 & : & 1 & 0 & 0 \\ 0 & 1 & 0 & : & 1 & 0 & 1 \\ 0 & 0 & 1 & : & 1 & 1 & 1 \end{bmatrix} \\
 &= 101011
 \end{aligned}$$

$$\therefore R^T = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{aligned} \therefore HR^T &= \begin{bmatrix} 1 & 1 & 1 & : & 1 & 0 & 0 \\ 0 & 0 & 1 & : & 0 & 1 & 0 \\ 0 & 1 & 1 & : & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 \cdot 1 \oplus 1 \cdot 1 \oplus 1 \cdot 0 \oplus 1 \cdot 0 \oplus 0 \cdot 1 \oplus 0 \cdot 0 \\ 0 \cdot 1 \oplus 0 \cdot 1 \oplus 1 \cdot 0 \oplus 0 \cdot 0 \oplus 1 \cdot 1 \oplus 0 \cdot 0 \\ 0 \cdot 1 \oplus 1 \cdot 1 \oplus 1 \cdot 0 \oplus 0 \cdot 0 \oplus 0 \cdot 1 \oplus 1 \cdot 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \end{aligned}$$

Since $HR^T \neq 0$, received code is wrong.

3.5.4 Error Correction

It is assumed that the coding/decoding system has been designed to correct single error only. In order to correct the codeword we multiply received codeword with transpose of parity-check matrix to get **syndrome**. Then result of RH^T , i.e. syndrome is compared with the row of transpose of parity-check matrix (H^T). Matching row number is the number of bit in error. Error bit is then inverted to get the correct code.

The procedure is given below :

1. Find $S = RH^T$

where

R : Received code

H^T : Transpose of T

$S = [S_1, S_2, S_3 \dots]$ is called **syndrome**.

2. Match the result, i.e. S with row of H^T . The number of row where the match occur gives the number of bit in error. This bit is inverted to correct the error.

Ex. 3.8 : For a (6, 3) block code received code is 110100. If $A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$. Find the received code

is correct or wrong. If wrong correct it.

The decoding technique for the burst code is similar to the technique used for cyclic codes.

3.8 Check-sums and Cyclic Redundancy Checks

Check-sums and cyclic redundancy checks are used to detect error in the transmission of block of data. Check-sum is formed by adding all the data bytes in the block, ignoring any carries. It is the arithmetic sum of all data bytes. The resulting sum is stored after the data block. When the data is read, it is re-added and the sum is compared with the recorded check-sum. There is no error in the data transmission if both the sum matches; otherwise data has to be re-transmitted.

The cyclic redundancy check (CRC) involves dividing the string of data bytes by a constant. Any remainder from this division is written in as 2 CRC bytes, or characters. When the data is read out, these 2 bytes are subtracted from the data string. The result is divided by the original constant. This division gives a zero remainder if the data contains no errors. The CRC method is usually used for error checking of disk storage.

Ex. 3.9 : What would be the checksum of the following characters ?

(May-2000)

```

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

```

Sol. : The checksum is

```

      0 1 0 1 1 0 1 0
+     1 1 0 0 0 1 0 1
+     1 1 0 1 1 0 0 1
-----
      1 1 1 1 1 0 0 0

```

In hexadecimal it is represented as $(F8)_{16}$

3.9 Hamming Code

Hamming code not only provides the detection of a bit error, but also identifies which bit is in error so that it can be corrected. Thus Hamming code is called error detecting and correcting code. The code uses a number of parity bits (dependent on the number of information bits) located at certain positions in the code group. Follows sections describe how Hamming code can be constructed for single error correction.

3.9.1 Number of Parity Bits

As mentioned earlier, number of parity bits depend on the number of information bits. If the number of information bits is designed x , then the number of parity bits, P is determined by the following relationship :

$$2^p \geq x + p + 1 \quad \dots(1)$$

For example, if we have four information bit, i.e. $x = 4$, then P is found by trial and error using equation 1. Let $p = 2$. Then

$$2^p = 2^2 = 4$$

Step 2: Check for parity bits

For P₁: P₁ checks locations 1, 3, 5, and 7.

There is one 1 in the group.

∴ Parity checks for even parity is wrong → 1 (LSB)

For P₂: P₂ checks locations 2, 3, 6 and 7.

There are two 1s in the group

∴ Parity check for even parity is correct → 0

For P₄: P₄ checks locations 4, 5, 6 and 7.

There are one 1 in the group

∴ Parity check for even parity is wrong → 1

The resultant word is 1 0 1. This says that the bit in the number 5 location is in error. It is 0 and should be a 1. Therefore, the correct code is 0 1 1 0 0 1 1, which agrees with the transmitted code.

Ex. 3.13: The Hamming code 1 0 1 1 0 1 1 0 1 is received. Correct it if any errors. There are four parity bits and odd parity is used.

Sol.: **Step 1:** Construct a bit location table

Bit designation	D ₉	P ₈	D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Received code	1	0	1	1	0	1	1	0	1

Step 2: Check for parity bits

For P₁: P₁ checks locations 1, 3, 5, 7 and 9.

There are four 1s in the group

∴ Parity check for odd parity is wrong → 1 (LSB)

For P₂: P₂ checks locations 2, 3, 6 and 7.

There are three 1s in the group

∴ Parity check for odd parity is correct → 0

For P₄: P₄ checks locations 4, 5, 6 and 7.

There are three 1s in the group

∴ Parity check for odd parity is correct → 0

For P₈: P₈ checks locations 8 and 9.

There is one 1 in the group

∴ Parity check for odd parity is correct → 0

The resultant word is 0001. This says that the bit in the number 1 location is in error. It is 1 and should be a 0. Therefore, the correct code is 1 0 1 1 0 1 1 0 0.

basic elements forming the building blocks for complex digital system such as the computer.

Irish logician and mathematician George Boole developed a mathematical system for formulating logical statements with symbols, so that problems could be written and solved in a manner similar to ordinary algebra. Boolean algebra, as it is known today, finds application in the analysis and design of digital systems.

In this chapter we will study logic gates and we will see how their operation can be described using Boolean algebra. We will also see how logic gates can be combined to produce logic circuits, and how these circuits can be described and analyzed using Boolean algebra.

4.2 Logical Operators

We know that, to represent and solve arithmetic expressions we use arithmetic operators such as $+$, $-$, \times and \div . Similarly, we can use logical operators to represent and solve logical expressions. There are three basic logical operators : NOT/INVERT, AND and OR.

4.2.1 Logical Operator NOT/INVERT

The inversion (or complementing or negation) operator is written as a bar over its argument. Sometimes it is written "NOT". Thus the inverse of A is \bar{A} or NOT A. The logic operator NOT can be better understood if the switching-circuit realization of NOT function, as shown in Fig. 4.1, is considered.

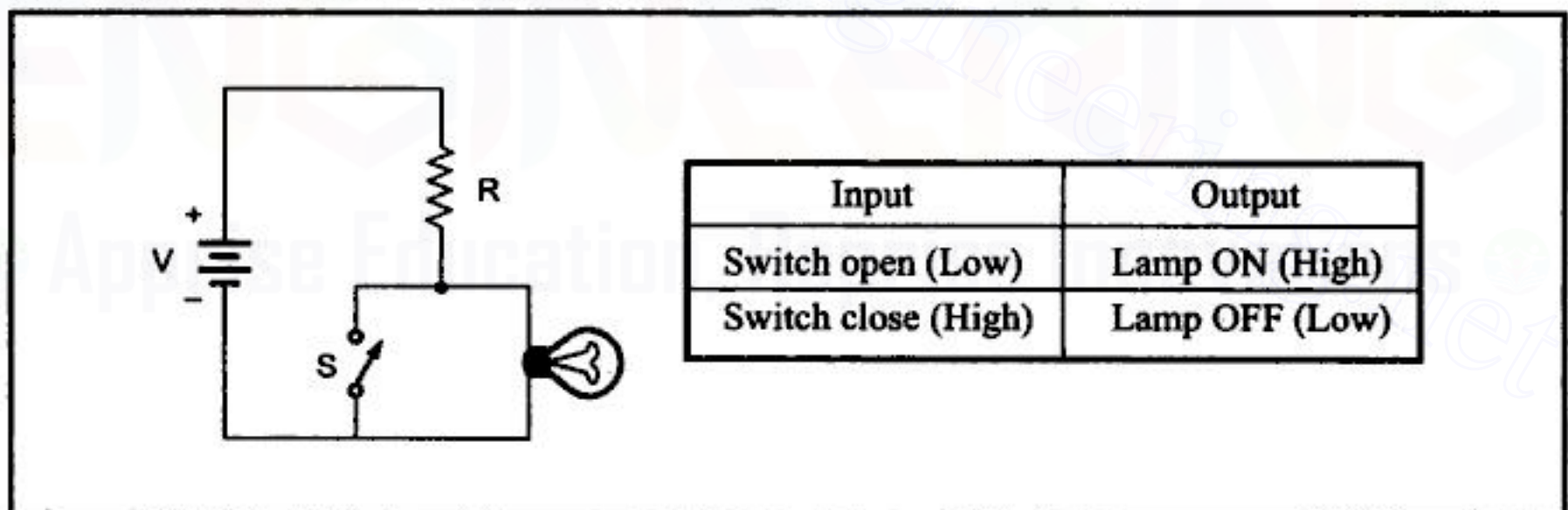


Fig. 4.1 Switching circuit analogy of NOT function

When the switch S is open, lamp is 'ON' and when the switch S is close, lamp is 'OFF'.

4.2.2 Logical operator AND

It is denoted by ' \times ' or ' \cdot '. These signs may be omitted, similar to ordinary algebra. For example: $A \cdot B$, $A \times B$ or AB has a same meaning. $A \cdot B$ may be read as 'A and B' or 'A times B'. $A \cdot B$ is high if A and B are both high and otherwise low. The logic operator AND can be better understood if the switching-circuit realization of AND function as shown in Fig. 4.2, is considered. *AND gate denotes lowest value of the input variables.*

The truth table for a two-input AND gate is shown in Table 4.2. This table can be expanded for any number of inputs. For any AND gate, regardless of the number of inputs, the output is high only when all inputs are HIGH.

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Table 4.2 Truth table for 2 input AND gate

Fig. 4.9 shows one way to build a 2-input AND gate. The inputs are labelled A and B, while the output is Y. Let us assume a supply voltage V_{CC} of +5 V. Also we will assume the input voltages are either 0 V (Low) or +5 V (High). With 2 inputs, there are four possible input cases and we will now observe the output for all four input cases.

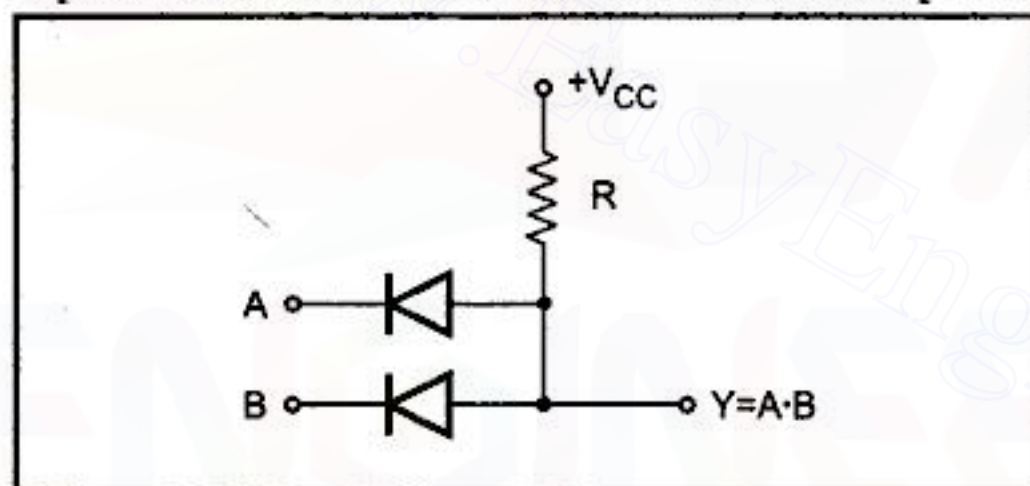


Fig. 4.9 2-input AND gate

CASE 1 : A is low and B is low : When both input voltages are low, the cathode of each diode is grounded. Therefore, the positive supply forward-biases both diodes in parallel. Because of this, the output voltage is ideally zero (practically 0.7 V for Si). This means Y is low.

CASE 2 : A is low and B is high : When A is low, the upper diode is forward-biased (ON), and it pulls the output down to a low voltage, i.e. $Y = 0$. With the B input high, the lower diode goes into reverse bias (OFF).

CASE 3: A is high and B is low : Because of the symmetry of the circuit, the circuit operation is similar to case 2. But in this case, upper diode is reverse biased (OFF), lower diode is forward biased (ON), and Y is low.

CASE 4 : A is high and B is high : When both inputs are at +5 V, both diodes are reverse biased and there is no current through diodes and resistor R. This pulls up the output Y to the supply voltage. Therefore, Y is high.

Pulsed Operation :

In a majority of applications, the inputs to a gate are not constant levels but are **voltages** that change with time between two logic levels and that can be classified as **pulse waveforms**. Let us study the operation of AND gate with pulsed input waveforms. Keep in mind that an AND gate obeys the truth table operation regardless of whether its inputs are constant levels or pulsed levels. In studying the pulsed operation of the AND gate, we

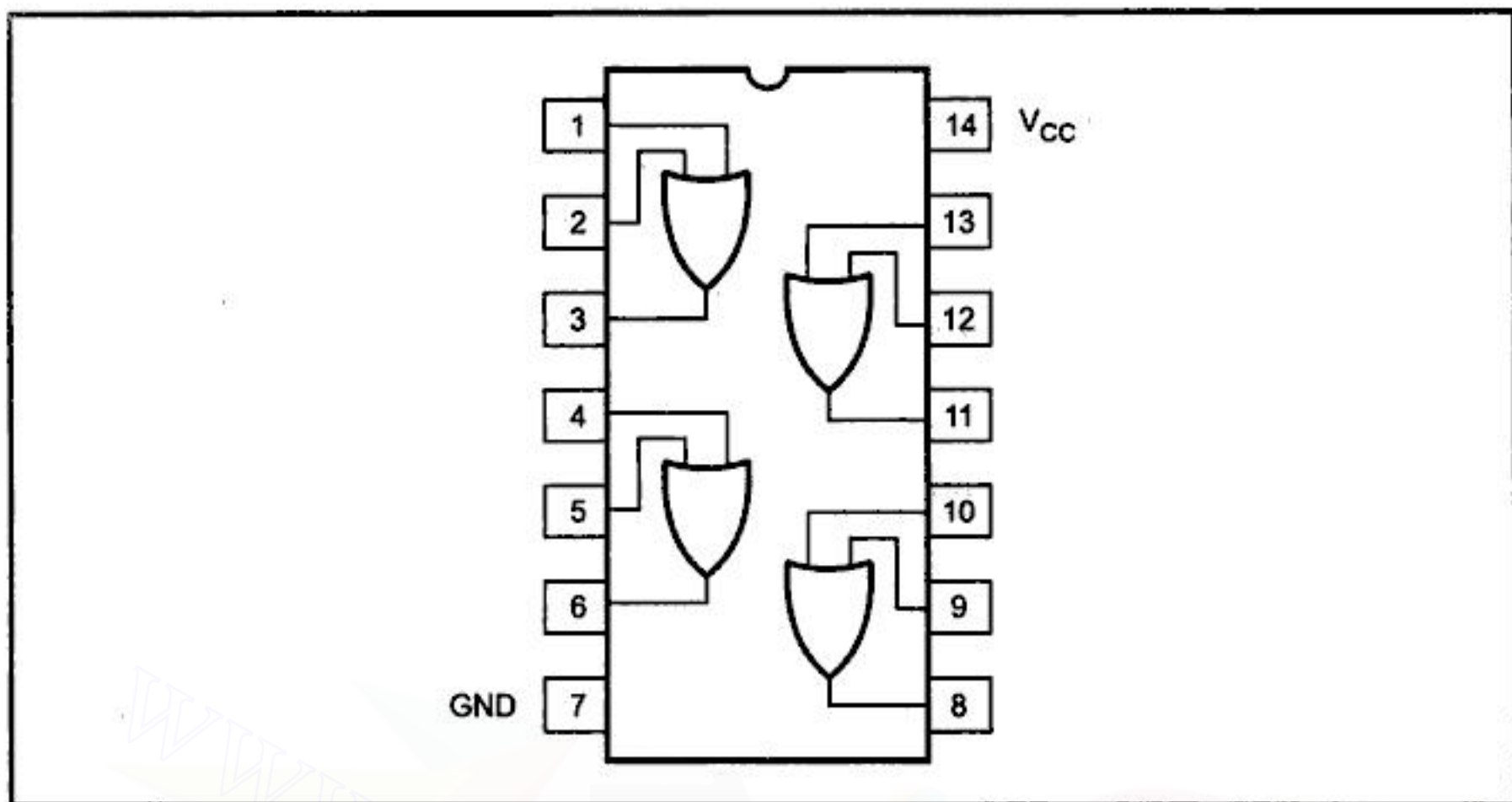


Fig. 4.16 Pin diagram of a 7432

4.3.4 The NAND gate

The term NAND is a contraction of NOT-AND and implies an AND function with a complemented (inverted) output. A standard logic symbol for a two-input NAND gate and its equivalency to an AND gate followed by an inverter are shown in Fig. 4.17.

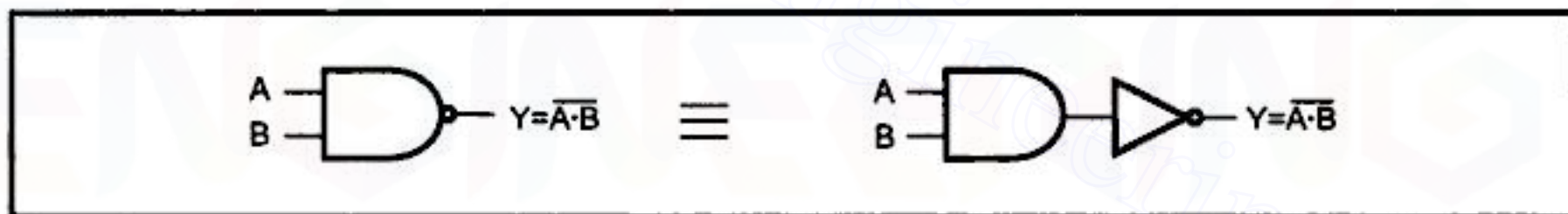


Fig. 4.17 NAND gate symbol and equivalent circuit

The NAND gate is a **universal** gate as it can be used to construct an AND gate, an OR gate an inverter, or any combination of these functions. The logical operation of the NAND gate is such that a LOW output occurs only when **all** inputs are HIGH. When **any** of the inputs is LOW, the output will be HIGH. Note that this operation is opposite to that of the AND as far as output is concerned. Fig. 4.18 illustrates the logical operation of a two-input NAND gate for all four input combinations.

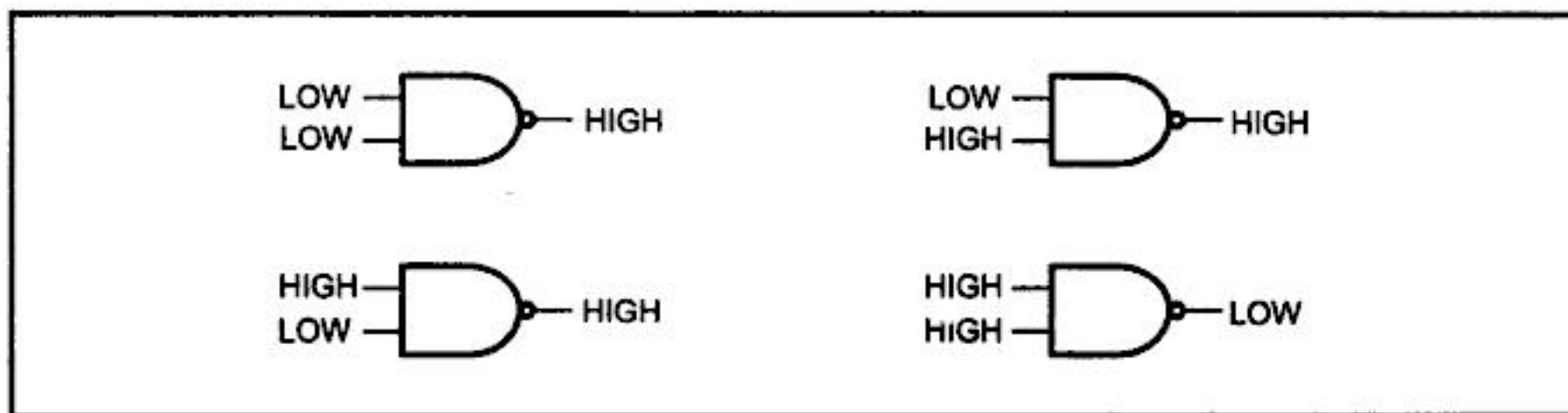


Fig. 4.18 Four possible inputs for two input NAND Gate and resulting outputs

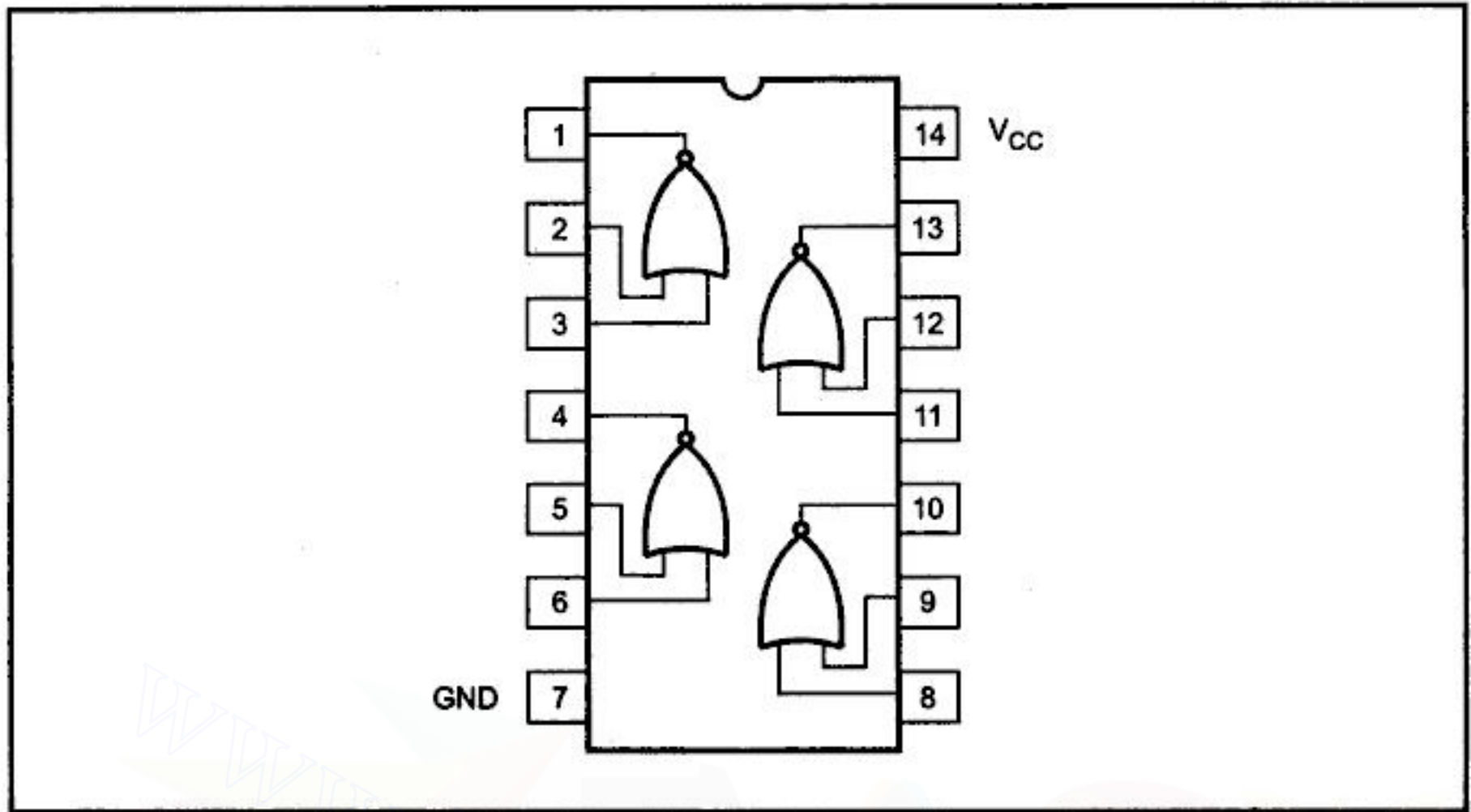


Fig. 4.24 Pin diagram of a 7402

4.3.6 The Exclusive-OR Gate

The EX-OR gate is an abbreviation for Exclusive-OR gate. An EX-OR gate has two or more inputs and one output, as indicated by the standard logic symbol in Fig. 4.25 where EX-OR gates with two and four inputs are shown.

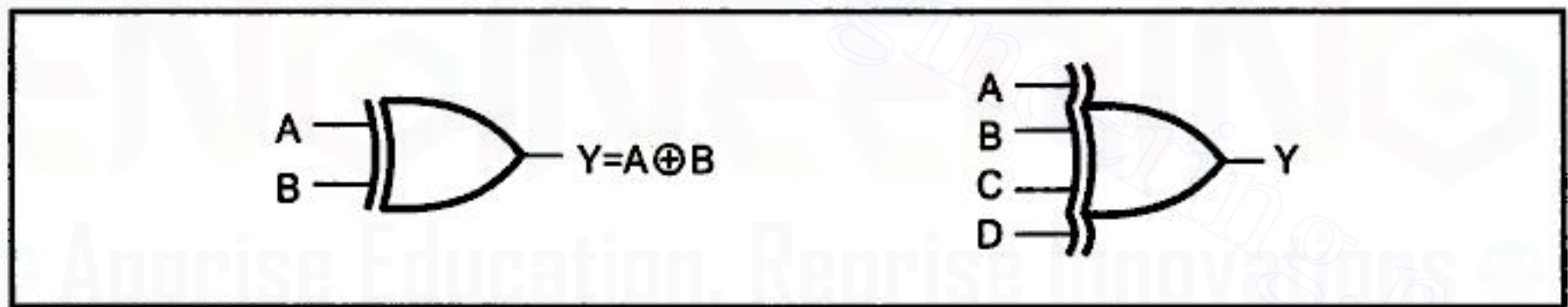


Fig. 4.25 (a) Two input EX-OR

Fig. 4.25 (b) Four input EX-OR

It recognizes only the words that have an odd number of ones. This means that for odd number of ones, output of EX-OR gate is high. The Fig. 4.26 illustrates the logic operation for a two-input EX-OR gate for all four possible input combinations.

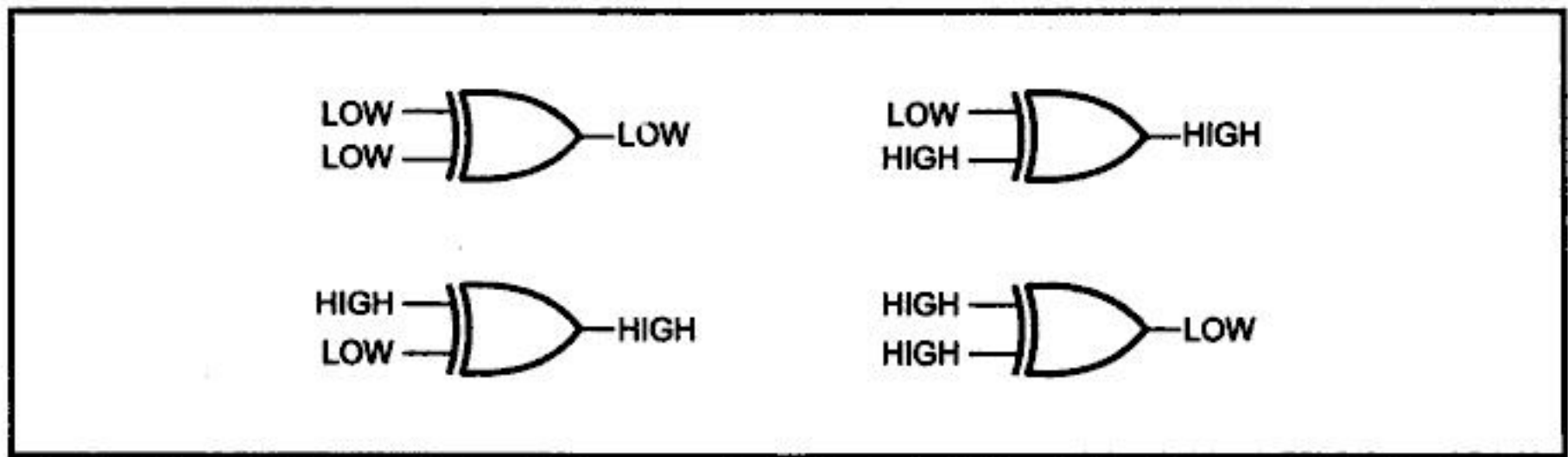


Fig. 4.26 Four possible inputs for two input EX-OR gate and resulting outputs

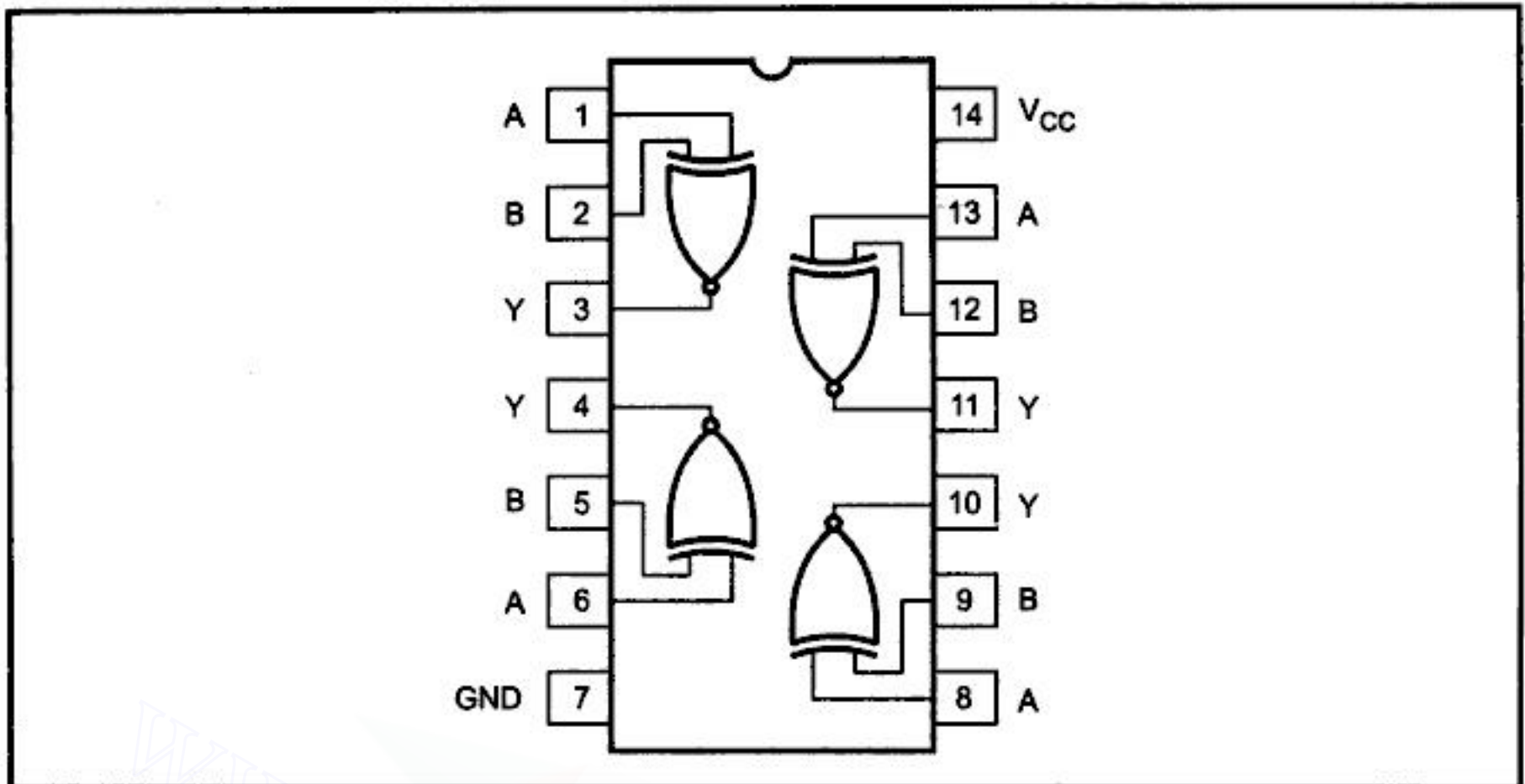


Fig. 4.32 Pin diagram of a 74266

Points to Remember

1. In digital systems, two voltage levels represent the two binary digits, 1 and 0. If the higher of the two voltages represents 1 and the lower voltage represents 0, the system is called a positive logic system. On the other hand, if the lower voltage represents a 1 and the higher voltage represents a 0, the system is called a negative logic system.
2. Three basic logical operators are : AND, OR and NOT.
3. Logic gates are the basic elements that make up a digital system.
4. The logic gate is a digital circuit with one or more inputs but only one output.
5. The inverter performs a basic logic function called inversion. It changes one logic level to its opposite level.
6. For any AND gate, regardless of the number of inputs, the output is HIGH only when all inputs are HIGH.
7. An OR gate produces a HIGH on the output when any of the inputs is HIGH. The output is LOW only when all of the inputs are LOW.
8. The logical operation of NAND gate is such that a low output occurs only when all inputs are HIGH.
9. The logic operation of the NOR gate is such that a LOW output occurs when any of its inputs is HIGH. The output is HIGH, only when all of its inputs are LOW.
10. For odd number of ones, at the input, the output of EX-OR gate is HIGH.
11. For even number of ones at the input, or inputs having all zeroes, the output of EX-NOR gate is HIGH.

A	B	C	A + B	(A + B) + C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

≡

A	B	C	B + C	A + (B + C)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Table 4.11 Truth tables for associative law for OR gates

LAW 2 : (AB) C = A (BC) : The associative law of multiplication states that it makes no difference in what order the variables are grouped when ANDing several variables. For three variables, A AND B ANDed with C is the same as A ANDed with B and C. The Fig. 4.35 illustrates this law as applied to AND gates.

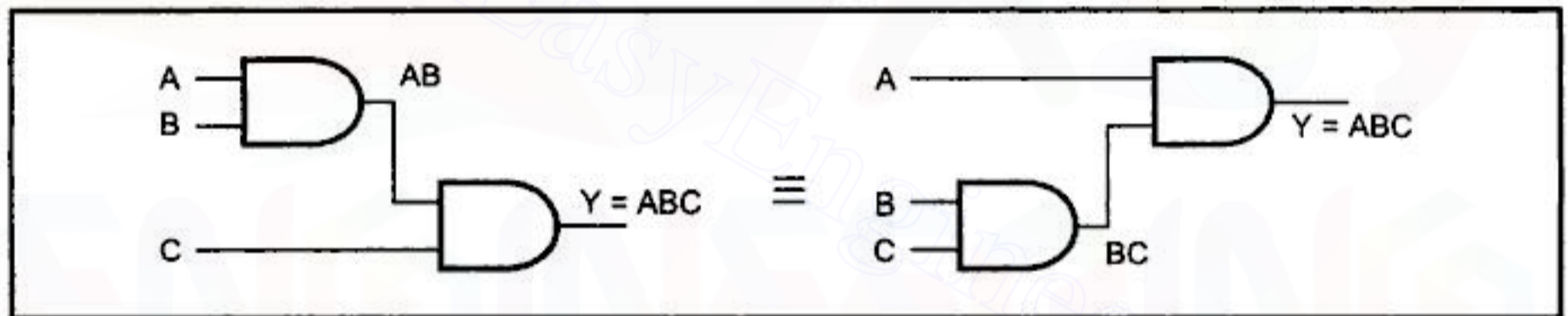


Fig. 4.35 Associative law applied to AND gates

A	B	C	AB	(AB) C
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	1	0
1	1	1	1	1

≡

A	B	C	BC	A (BC)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 4.12 Truth table for associative law for AND gates

Illustration of Rule 6

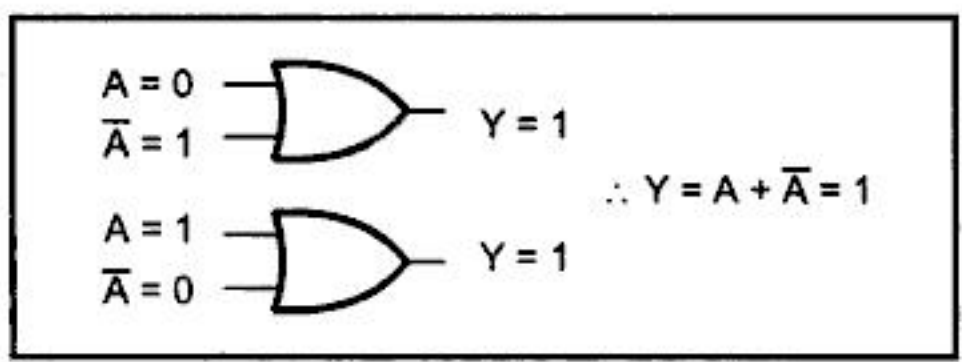


Fig. 4.42

Rule 6 says that when variable and its complement are ORed, the result is always 1. As shown in Fig. 4.42, when $A = 0$, $A + \bar{A} = 0 + 1 = 1$ and when $A = 1$, $A + \bar{A} = 1 + 0 = 1$.

Illustration of Rule 7

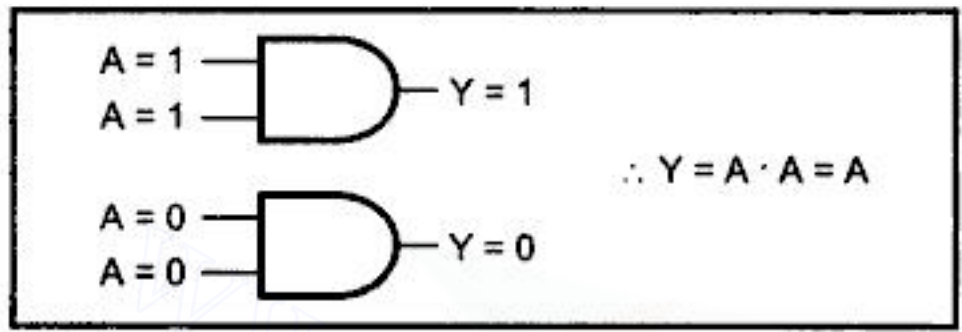


Fig. 4.43

Rule 7 says that if variable is ANDed with itself, the result is equal to the variable. As shown in Fig. 4.43, when $A = 1$, $1 \cdot 1 = 1$ and when $A = 0$, $0 \cdot 0 = 0$.

Illustration of Rule 8

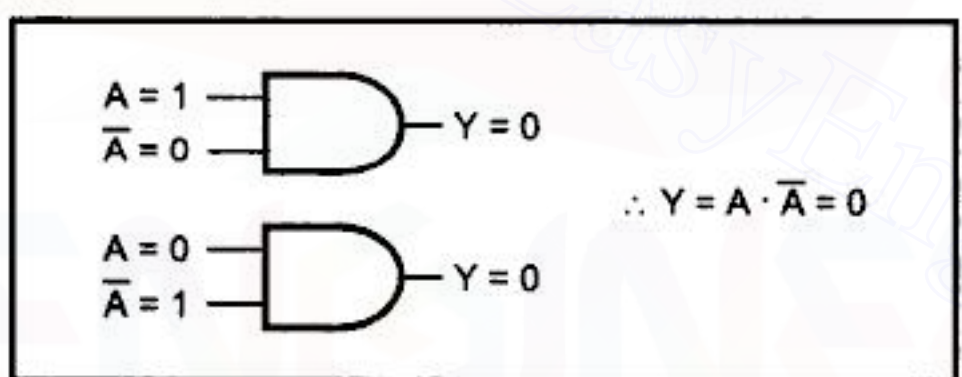


Fig. 4.44

Rule 8 says that when variable and its complement are ANDed, the result is always 0. As shown in Fig. 4.44, when $A = 1$, $A \cdot \bar{A} = 1 \cdot 0 = 0$, and when $A = 0$, $A \cdot \bar{A} = 0 \cdot 1 = 0$.

Illustration of Rule 9

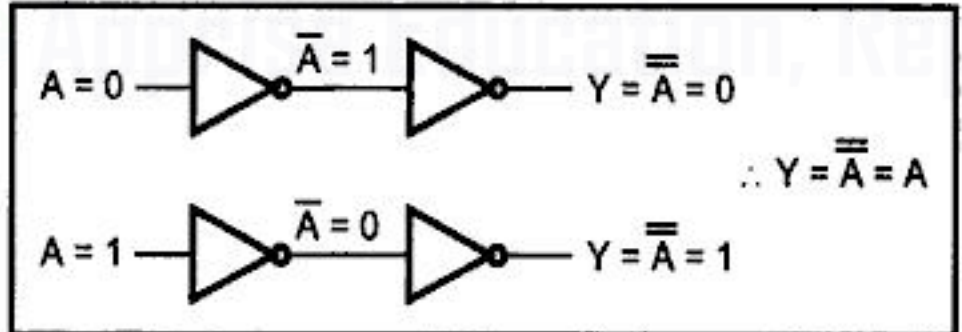


Fig. 4.45

Rule 9 says that if a variable is complemented twice, the result is the variable itself. When $A = 0$, $\bar{\bar{A}} = 0$ and when $A = 1$, $\bar{\bar{A}} = 1$.

Proof of Rule 10

Rule 10 is given as $A + AB = A$

$$\begin{aligned} \text{L.H.S.} &= A + AB \\ &= A(1 + B) \\ &= A(1) \\ &= A \end{aligned}$$

Distributive law
 Rule 2 : $(1 + A = 1)$
 Rule 4 $(A \cdot 1 = A)$... proved

Proof of Rule 11

Rule 11 is given as $A + \bar{A}B = A + B$

Ex. 4.11 : $\overline{A}B\overline{C}D + \overline{A}BCD + ABD$

$$= \overline{A}BD (\overline{C} + C) + ABD$$

Distributive law

$$= \overline{A}BD + ABD$$

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

$$= BD (\overline{A} + A)$$

Distributive law

$$= BD$$

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

Ex. 4.12 : $A + \overline{A}B + A\overline{B} = A + B$

$$\text{L.H.S} = A + \overline{A}B + A\overline{B}$$

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

Rule 11 : $[A + \overline{A}B = A + B]$

$$= A + A + B$$

Rule 11 : $[A + \overline{A}B = A + B]$

$$= A + B$$

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

Ex. 4.13 :

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

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

Theorem 1 : $[\overline{AB} = \overline{A} + \overline{B}]$

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

Rule 11 : $[A + \overline{A}B = A + B]$

$$= \overline{\overline{A} + 1}$$

Rule 5 : $[A + A = A]$ andRule 6 $[A + \overline{A} = 1]$

$$= \overline{1}$$

Rule 2 : $[A + 1 = 1]$

$$= 0$$

Ex. 4.14 : $AB + \overline{A}C + A\overline{B}C (AB + C)$

$$= AB + \overline{A}C + A\overline{A}B\overline{B}C + A\overline{B}CC$$

Distributive law

$$= AB + \overline{A}C + A\overline{B}CC$$

Rule 8 : $[A \cdot \overline{A} = 0]$

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

Rule 7 : $[A \cdot A = A]$

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

Theorem 1 : $[\overline{AB} = \overline{A} + \overline{B}]$

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

Rule 11 : $[A + \overline{A}B = A + B]$

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

Commutative law

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

Rule 11 : $[A + \overline{A}B = A + B]$ Here $B = \overline{B}C$

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

Commutative law

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

Rule 11 : $[A + \overline{A}B = A + B]$

$$= \overline{A} + \overline{C} + 1$$

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

$$= 1$$

Rule 2 : $[A + 1 = 1]$

4.5 Universal Property

The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates. This is illustrated in following sections

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. $A + B = \overline{\overline{A + B}}$. Fig. 4.53 shows the two input OR gate using NOR gates.

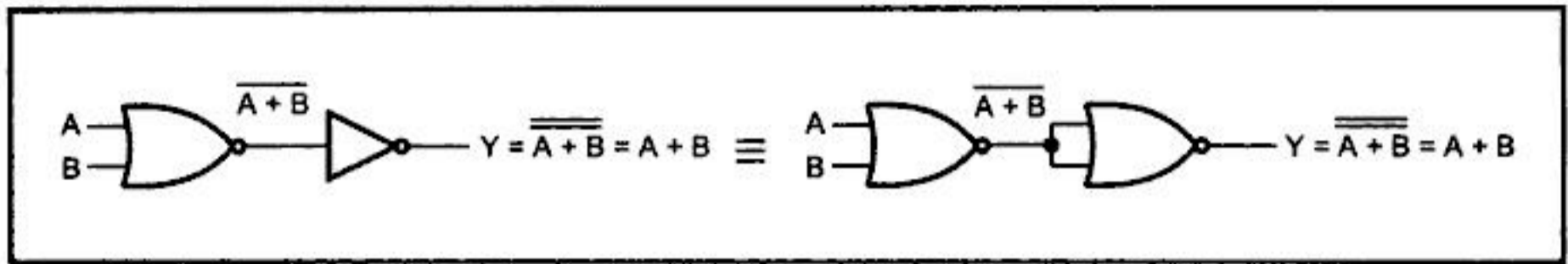


Fig. 4.53 OR function using NOR gates

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

=

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

Table 4.20 Truth table

AND Function :

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

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

Rule 9 : $[\overline{\overline{A}} = A]$

DeMorgan's Theorem 2

The above equation is implemented using only NOR gates as shown in the Fig. 5.54.

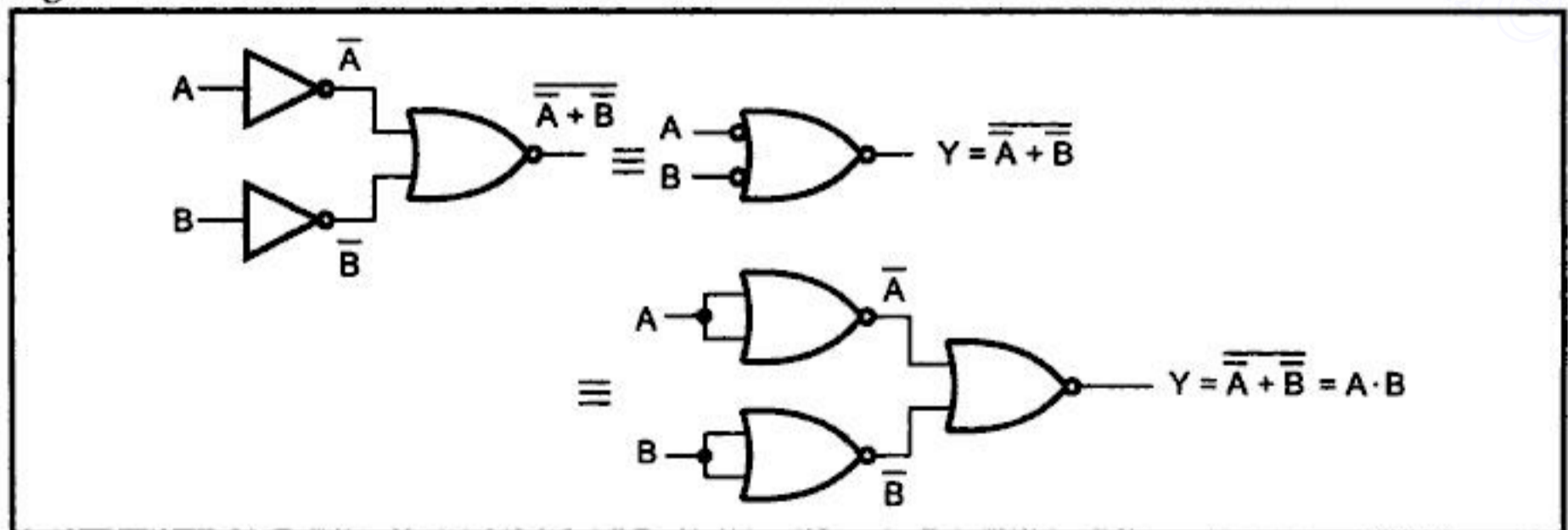


Fig. 4.54 AND function using NOR gates

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

NOR Circuit

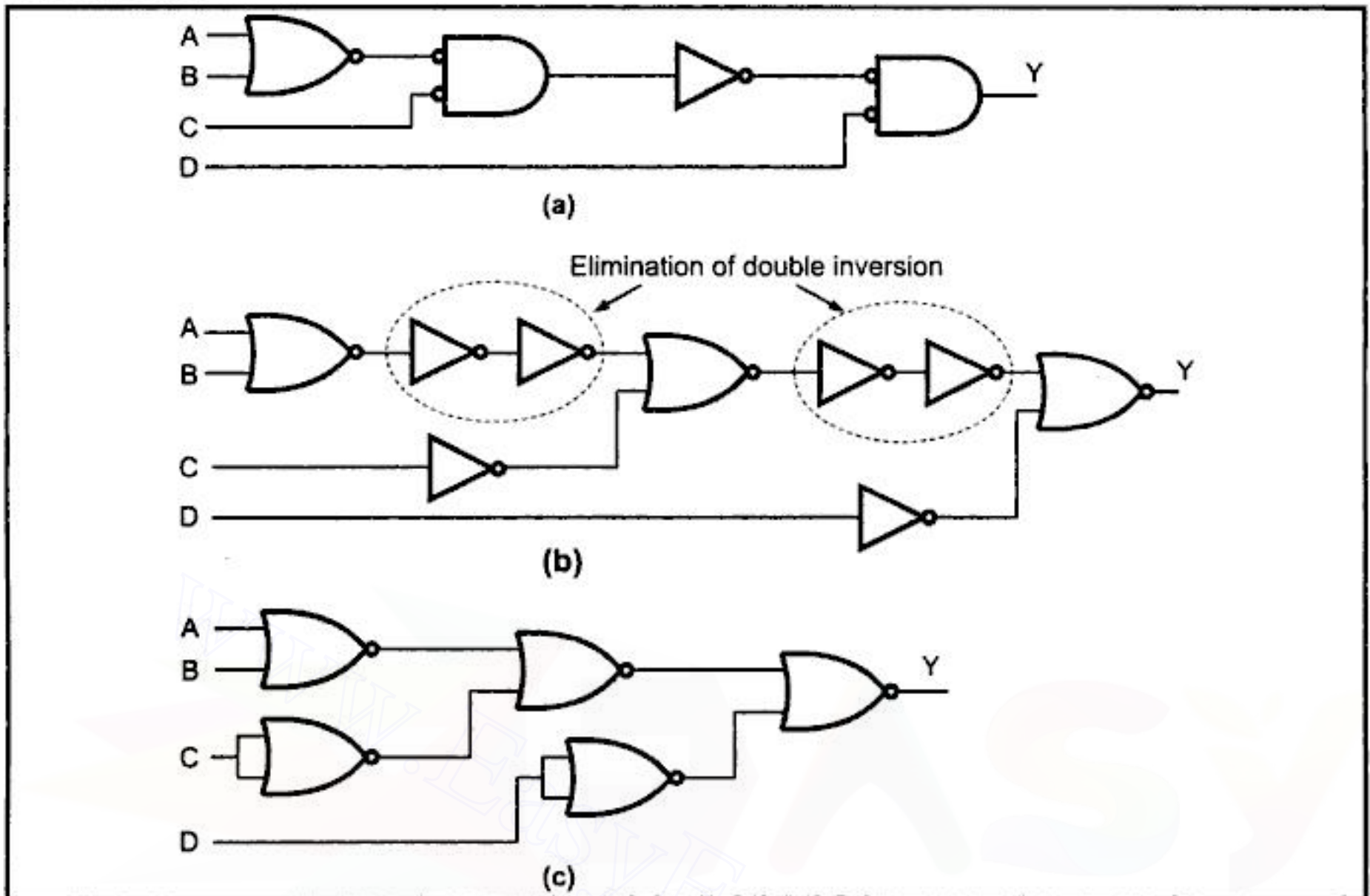


Fig. 4.59

The straight forward method for implementing the given expression uses two AND gates, one OR gate and one inverter, as shown in the Fig. 4.57. Now we will see implementation of this circuit accomplished by replacing each AND gate, OR gate and inverter by its equivalent NAND gate using steps mentioned earlier. This is illustrated in Fig. 4.58. If we now compare original circuit and the circuit with NAND gates, we find that three ICs are required (AND, OR, and INVERTER) to implement original circuit, whereas only two NAND ICs are required for the circuit with NAND gates. (With two NAND ICs we have 8 NAND gates and only 6 NANDs are required to implement the logic circuits).

Applying similar steps for NOR gate we get NOR circuit as shown in Fig. 4.59. Here, we need only two ICs of NOR gate.

Ex. 4.16: Implement Boolean expression for EX-OR gate using NAND gates. (Dec-99, Dec-2002)

Sol.: Boolean expression for EX-OR gate is $AB + \bar{A}\bar{B}$. Using AND/OR/INV logic the circuit can be implemented as shown in Fig. 4.60.

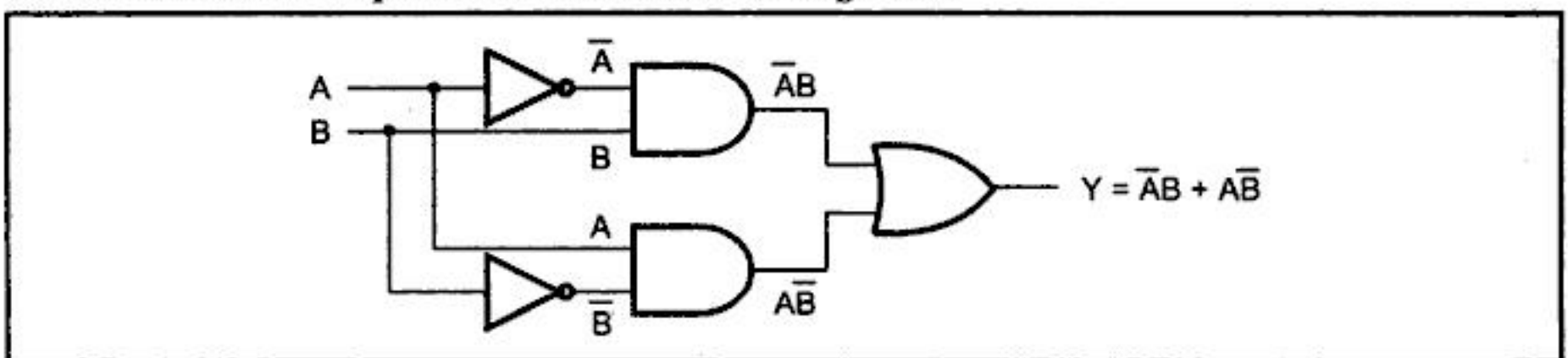


Fig. 4.60

The symbols used to represent logic gates in this chapter have been used in the digital industry for many years. These symbols work well for the logic gates because each gate symbol has distinctive shape, and each gate input has the same function. They do not provide enough useful information, however, for the more complex logic devices such as Flip-Flops, counters decoders etc. In order to provide more useful information about these complex logic devices, a new set of standards was introduced jointly by the American National standards Institute (ANSI) and the institute of Electrical and Electronics Engineers (IEEE) in 1984. This standard is commonly known as standard 91-1984 and is now being used by many industries and manufacturers of electronic devices.

The principal difference in the new standard is that it uses rectangular symbols for all devices instead of different symbol shapes for each device. A special dependency notation system is used to indicate how the outputs depend on the inputs. These dependency notations are also called as qualifying symbols, which are placed inside the rectangle to indicate the type of logic operations.

Fig. 4.64 (See Fig. on previous page) shows the newer rectangular symbols for various gates.

The ANSI/IEEE logic symbols for integrated circuits containing multiple logic gates are drawn by stacking rectangles on top of each other, one rectangle for each gate. Fig. 4.65 shows the ANSI/IEEE symbols for each of the 7400- series circuits whose pin diagrams are already shown in section 4.3.

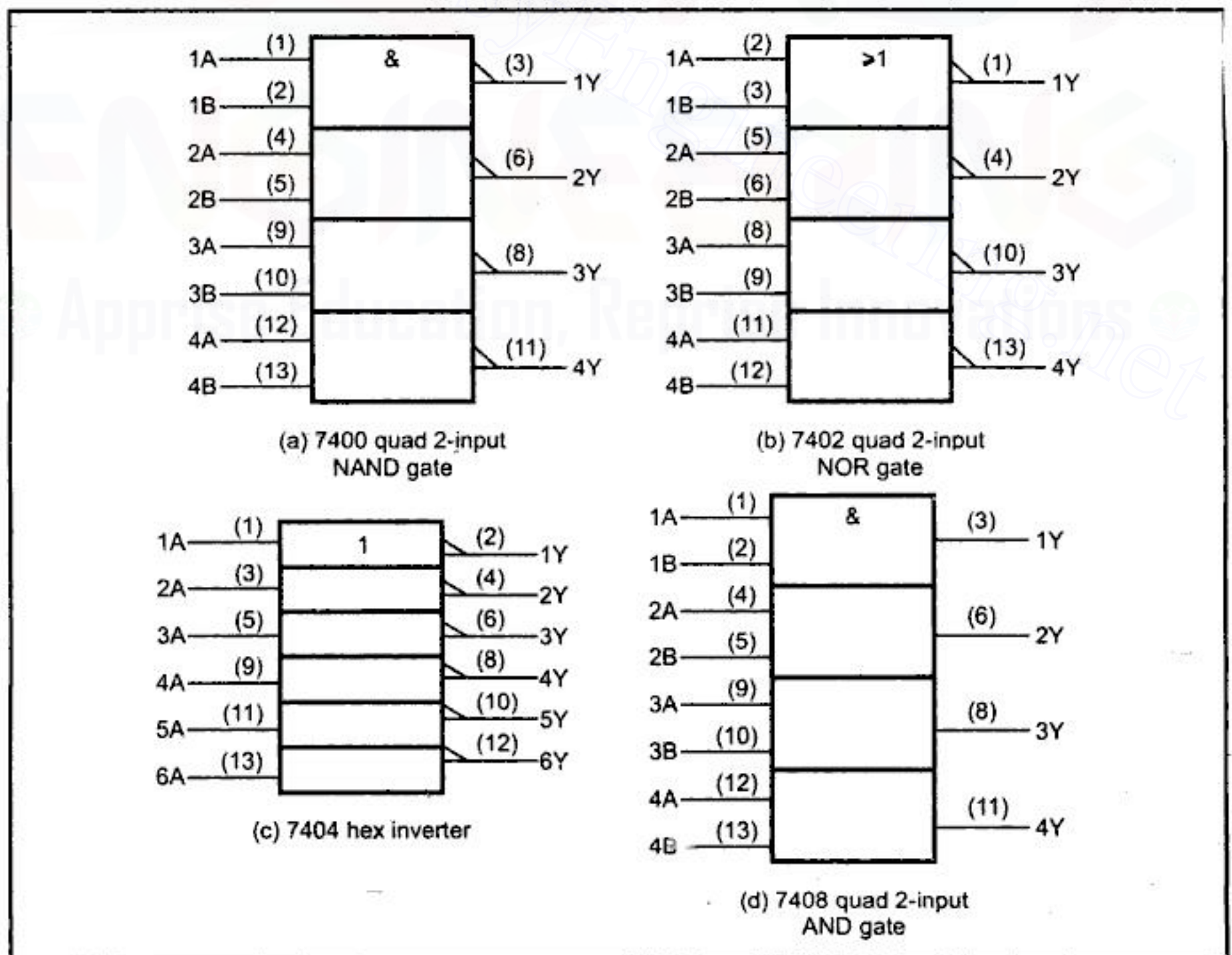


Fig. 4.65 (continued on next page)

Ex. 4.21 : Determine the truth table for the circuit shown in Fig. 4.71.

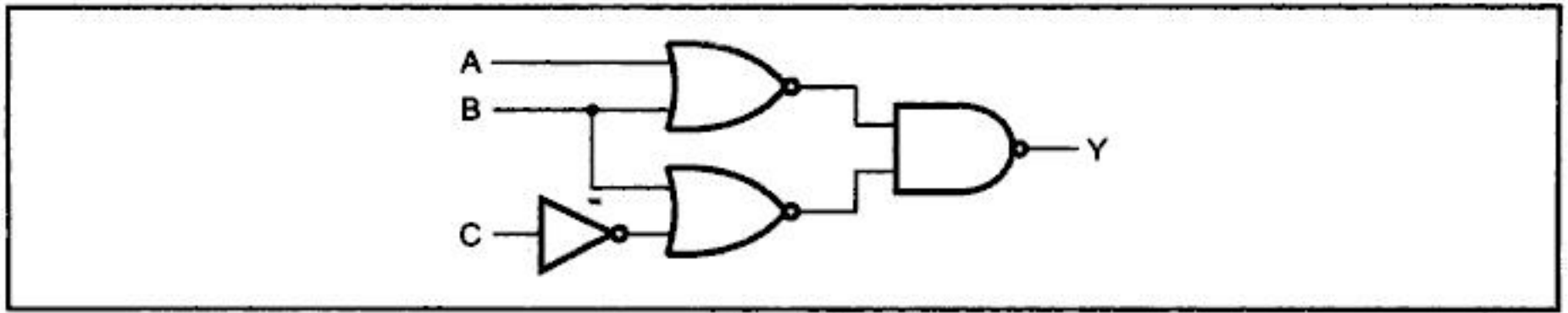


Fig. 4.71

Sol. :

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

De Morgan's Theorem 1

Rule 9 : $\overline{\overline{A}} = A$

Rule 5 : $A + A = A$

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

Truth table 4.24

Ex. 4.22 : Simplify the following expression

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

Rule 8 : $A\overline{A} = 0$

Rule 8 : $A\overline{A} = 0$

Ex. 4.23 : Simplify each of the following using Demorgan's theorem

a) $\overline{(A+B)}(\overline{A+B})$ b) $\overline{\overline{\overline{A}BCD}}$

Therefore logic circuit is

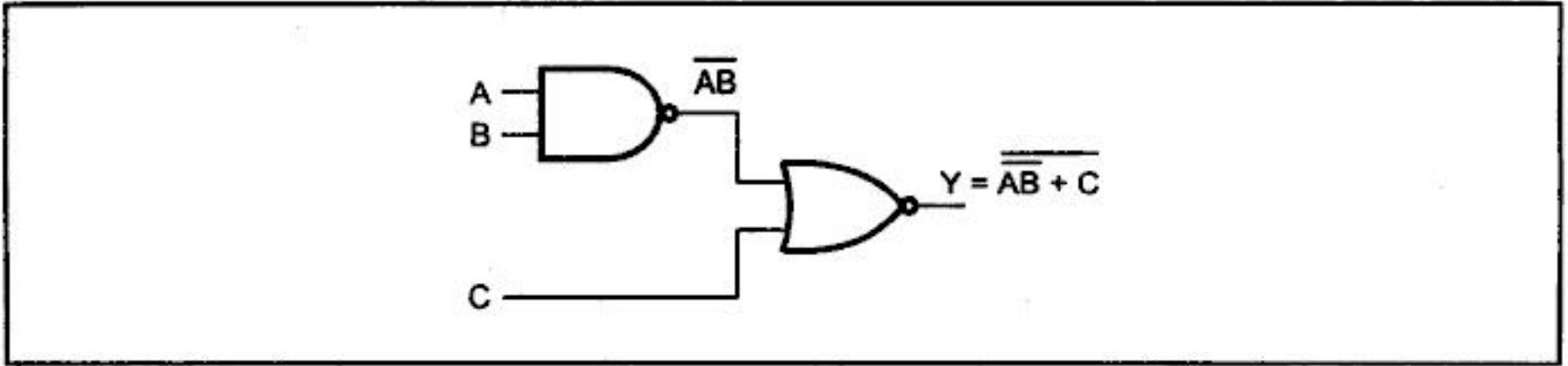


Fig. 4.78

Ex. 4.27 : Implement $Y = ABCD$ using two inputs NAND gates

Sol. :

$$\begin{aligned}
 Y &= ABCD \\
 &= \overline{\overline{ABCD}} \\
 &= \overline{\overline{AB} + \overline{CD}} \\
 &= \overline{\overline{AB} \cdot \overline{CD}} \\
 &= \overline{\overline{\overline{AB} \cdot \overline{CD}}}
 \end{aligned}$$

Rule 9 : $\overline{\overline{A}} = A$

DeMorgan's Theorem 1

DeMorgan's Theorem 2

Therefore the logic circuit is

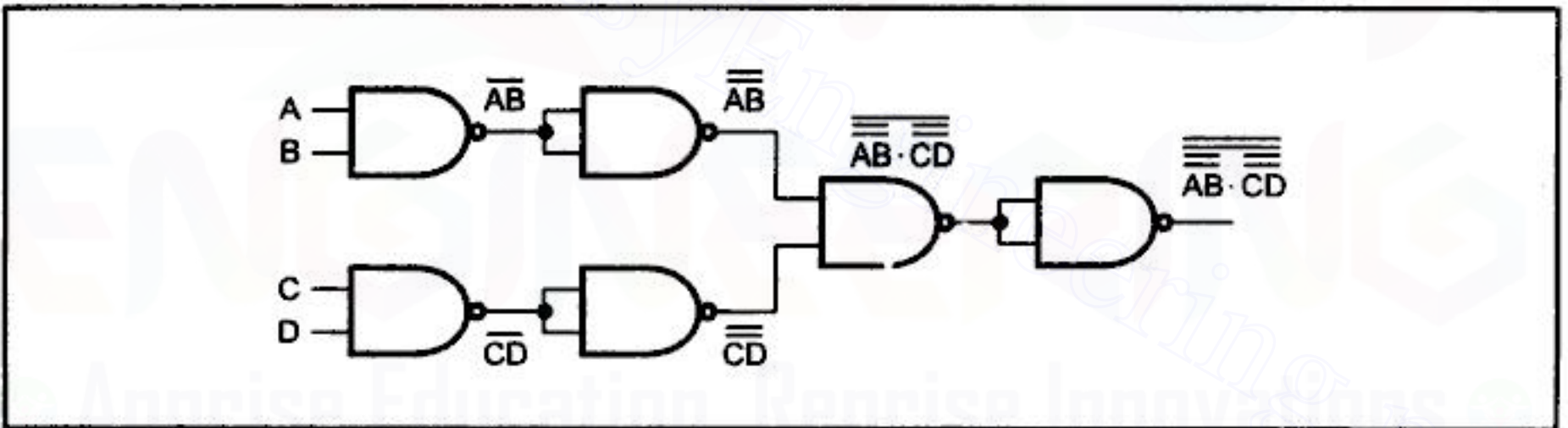


Fig. 4.79

Ex. 4.28 : Draw the circuit shown in Fig. 4.80 using ANSI/IEEE symbols

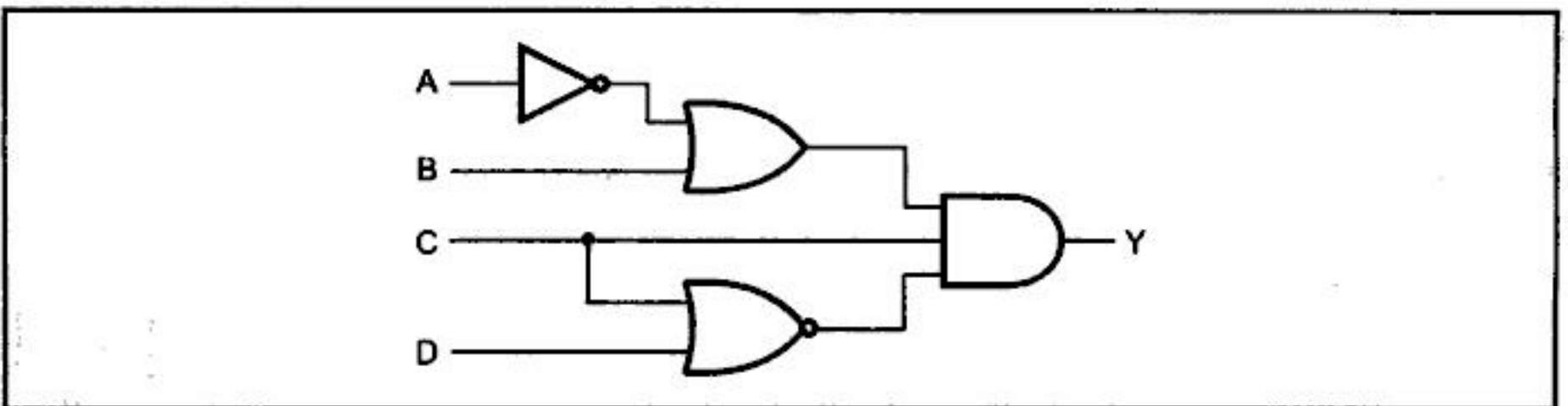


Fig. 4.80

Sol. :

i)

$$F_1 = \bar{x} y \bar{z} + \bar{x} \bar{y} z$$

∴

$$\bar{F}_1 = \overline{\bar{x} y \bar{z} + \bar{x} \bar{y} z}$$

$$= \overline{\bar{x} y \bar{z}} \cdot \overline{\bar{x} \bar{y} z}$$

$$= (\bar{x} + \bar{y} + z) \cdot (\bar{x} \bar{y} \bar{z})$$

$$= (x + \bar{y} + z) \cdot (x + y + \bar{z})$$

$$= xx + xy + x\bar{z} + x\bar{y} + y\bar{y} + \bar{y}\bar{z} + xz + yz + z\bar{z}$$

$$= x + xy + x\bar{z} + x\bar{y} + \bar{y}\bar{z} + xz + yz$$

$$= x(1 + y + \bar{z} + \bar{y} + z) + \bar{y}\bar{z} + yz$$

$$= x + \bar{y}\bar{z} + yz$$

$$\text{ii) } F_2 = x(\bar{y}\bar{z} + yz)$$

∴

$$\bar{F}_2 = \overline{x(\bar{y}\bar{z} + yz)}$$

$$= \bar{x} + \overline{(\bar{y}\bar{z} + yz)}$$

$$= \bar{x} + (\bar{y}\bar{z} \cdot \overline{yz})$$

$$= \bar{x} + [(\bar{y} + \bar{z}) \cdot (\bar{y} + \bar{z})]$$

$$= \bar{x} + [(y + z) \cdot (\bar{y} + \bar{z})]$$

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

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

Review Questions

1. Draw a typical digital signal waveform and properly identify the two operating levels.
2. Define positive logic and negative logic.
3. Write the names of basic logical operators.
4. What is a logic gate ?
5. Draw the logic symbol and construct the truth table for each of the following gates :
 - a) Two input NAND gate
 - b) Three input OR gate
 - c) Two input EX-OR gate
 - d) Three input EX-NOR gate
 - e) NOT gate
6. Write the names of Universal Gates.
7. Why are NAND and NOR gates known as Universal gates ?
8. Give the Boolean expressions used for following gates
 - a) AND
 - b) NOR
 - c) EX-OR
 - d) OR
 - e) NOT

34. 'NAND and NOR gates are called universal gates.' Justify. (Dec-96)
35. Implement EX-OR gate using only four NAND gates. (May-97)
36. Verify the Boolean function

$$AB + \bar{A}C + BC = AB + \bar{A}C$$
 (May-97)
37. State and explain principle of duality. (May-97)
38. Using Boolean Algebra prove :
- i) $A + BC = (A + B)(A + C)$
- ii) $(\bar{A}C)(B + \bar{C})(\bar{B}) = (\overline{A + B + C}) + \overline{B + C}$
- iii) $\overline{(AB + BC) \cdot AC} = \bar{A} + \bar{C}$ (May-97)
39. By using Boolean algebra simplify the following : (Dec-97)
- i) $\overline{AB + AB\bar{C}} + \bar{A}(\overline{B + AB})$
- ii) $(AB + \bar{A}\bar{C} + A\bar{B}C)(AB + C)$
- iii) $\overline{(CD + A)} + A + CD + AB$
- iv) $\overline{A + B} \cdot \overline{A + B}$
- Ans. : i) $\bar{A} + B\bar{C}$, ii) 1, iii) $CD + B$, iv) 0
40. Implement OR Gate using Diodes and verify its truth table. (Dec-97)
41. Demonstrate by means of truth tables the validity of the following theorems of Boolean algebra :
- (i) The associative law
- (ii) The distributive law
- (iii) De Morgan's theorem for three variables. (May-98)
42. Simplify the following Boolean functions :
- (i) $\bar{W}\bar{X}YZ + \bar{W}XYZ + W\bar{X}YZ + WXYZ + WXY\bar{Z}$
- (ii) $\overline{ABC} + AB + \overline{ABC} + \bar{B}C$. (May-98)
- Ans. : i) $Y(z + WX)$, ii) 1
43. By using the rules of Boolean algebra reduce the following expressions :
- (i) $Y = (A + \bar{B} + \bar{C})(\bar{A} + B)(B + \bar{C})$.
- (ii) $L = \overline{PQRS} + \overline{PQRS} + \overline{PQRS} + PQRS + PQRS$. (Dec-98)
- Ans. : (i) $AB + \bar{C}(\bar{A} + B)$, (ii) $R(S + PQ)$
44. Implement the EX-OR logic operation with only NAND gates. (Dec-99)
45. State and explain the De Morgan's theorem. (Dec-99, May-2000, Dec-2000)
46. Using Boolean Algebra, show that (Dec-99)
- i) $\bar{a}\bar{b} + bc + \bar{a}b\bar{c} = \bar{a} + bc$
- ii) $D(\bar{A} + \bar{B}) + \bar{B}(C + AD) = D + \bar{B}C$
- iii) $(A + B)(A + C) = A + BC$
- iv) $\bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + xyz = y$
47. Using Boolean Algebra show that
- (i) $AB + AC + \bar{B}C = AB + \bar{B}C$

Let us see one standard sum of products expression :

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

Here, the original expression is standard sum of products, but the simplified expression is not a standard sum of products. It is only sum of products.

Similarly, we can say that a product of sums is a standard (canonical) product of sums if every sum term involves every literal or its complement. For example,

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

5.2.4 Converting Expressions in Standard SOP or POS Forms

Sum of products form can be converted to standard 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 $(C + \overline{C})$ and AND it with AB. Therefore, we get $AB(C + \overline{C}) = ABC + AB\overline{C}$

Ex. 5.1 : Convert the given expression in standard SOP form.

$$\begin{aligned} Y &= AC + AB + BC \\ &= AC(B + \overline{B}) + AB(C + \overline{C}) + BC(A + \overline{A}) \\ &= ABC + A\overline{B}C + ABC + AB\overline{C} + ABC + \overline{A}BC \\ &= ABC + A\overline{B}C + AB\overline{C} + \overline{A}BC \quad \text{Rule 5 : } [A + A = A] \end{aligned}$$

Ex. 5.2 : Convert the given expression in standard SOP form

$$\begin{aligned} Y &= A + AB + ABC \\ &= A \cdot (B + \overline{B}) \cdot (C + \overline{C}) + AB \cdot (C + \overline{C}) + ABC \\ &= (AB + A\overline{B}) \cdot (C + \overline{C}) + ABC + AB\overline{C} + ABC \\ &= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + ABC + AB\overline{C} + ABC \\ &= ABC + AB\overline{C} + A\overline{B}\overline{C} + A\overline{B}C \quad \text{Rule 5 : } [A + A = A] \end{aligned}$$

Similarly product of sums form can be converted to standard product of sums by ORing the terms in the expression with terms formed by ANDing the variable 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 $A + B$ where literal C is missing, then form term $C \cdot \overline{C}$ and OR $A + B$ with it. Therefore, we get,

$$A + B + C \cdot \overline{C} = (A + B + C) (A + B + \overline{C}) \quad \text{Rule 12 : } (A + BC) = (A + B) (A + C)$$

Ex. 5.3 : Convert the given expression in standard POS form

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

Ex. 5.4 : Convert the given expression in standard POS form

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

5.4 Algebraic Simplification

The Boolean algebra theorems that we studied in the previous section can be used to simplify expressions for a logic circuit. The steps for simplification of boolean expression using boolean algebra can be generalized as follows :

1. The original expression is put into the sum of products form by repeated application of DeMorgan's theorems and multiplication of terms.
2. Once it is in the sum of products form, the product terms are checked for common factors and factoring is performed whenever possible.

Boolean rules are used simultaneously which results in the elimination of one or more terms.

Now we will see some examples to illustrate the above mentioned technique.

Ex. 5.5 : Simplify the expression $Z = A B + A \bar{B} \cdot (\bar{A} \bar{C})$

Sol. :

Step 1 : Apply the DeMorgan's theorem and multiply out all terms to get expression in sum of products form

$$\begin{aligned}
 Z &= A B + A \bar{B} \cdot (\bar{A} \bar{C}) \\
 &= A B + A \bar{B} \cdot (\bar{A} + \bar{C}) && \text{De Morgan's theorem 1} \\
 &= A B + A \bar{B} \cdot (A + C) && \text{Rule : } [\bar{\bar{A}} = A] \\
 &= A B + A \bar{B} A + A \bar{B} C && \text{Distributive law}
 \end{aligned}$$

Step 2 : Search for common terms for factorization and apply boolean rules

$$\begin{aligned}
 Z &= A B + A \bar{B} A + A \bar{B} C \\
 &= A B + A \bar{B} + A \bar{B} C && \text{Rule 7 : } [AA = A] \\
 &= A B + A \bar{B} (1 + C) \\
 &= A B + A \bar{B} && \text{Rule 2 : } [1 + C = 1] \\
 &= A (B + \bar{B}) && \text{Rule 6 : } [A + \bar{A} = 1] \\
 &= A
 \end{aligned}$$

Ex. 5.6 : Simplify the logic circuit shown in Fig. 5.2 and implement simplified logic circuit using logic gates.

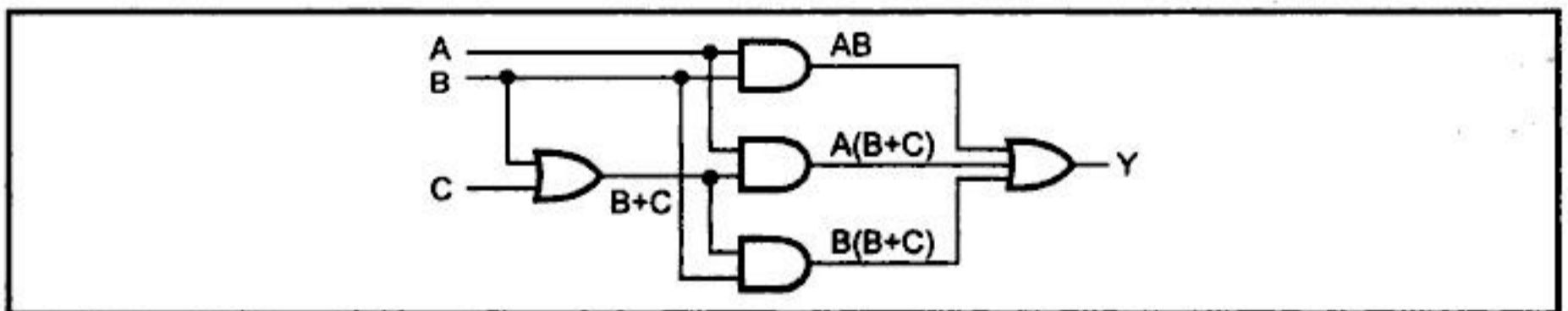


Fig. 5.2

Sol. : The expression for given logic circuit is

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

Step 1 : Multiply out all terms to get expression in sum of product form

example, in Fig. 5.5 (a) the only change that occurs in moving along the bottom row from $A\bar{B}$ to AB is the change from \bar{B} to B . Similarly, the only change that occurs in moving down the right column from $\bar{A}B$ to AB is the change from \bar{A} to A . Irrespective of number of variables the labels along each row and column must conform to the single-change rule.

The Fig. 5.6 shows another way to label the rows and columns of a 2, 3 and 4 variable maps 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 0s and 7s instead of variables.

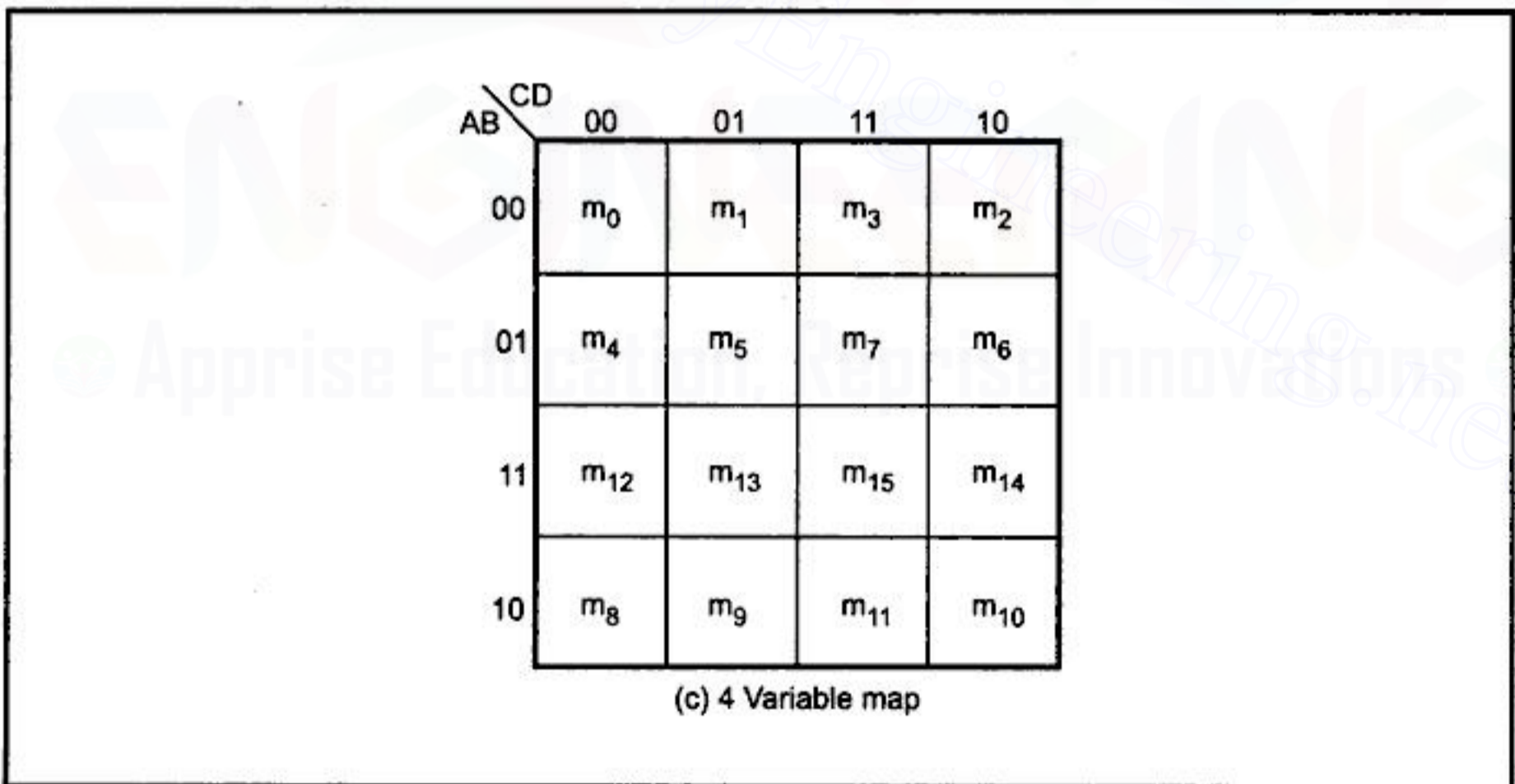
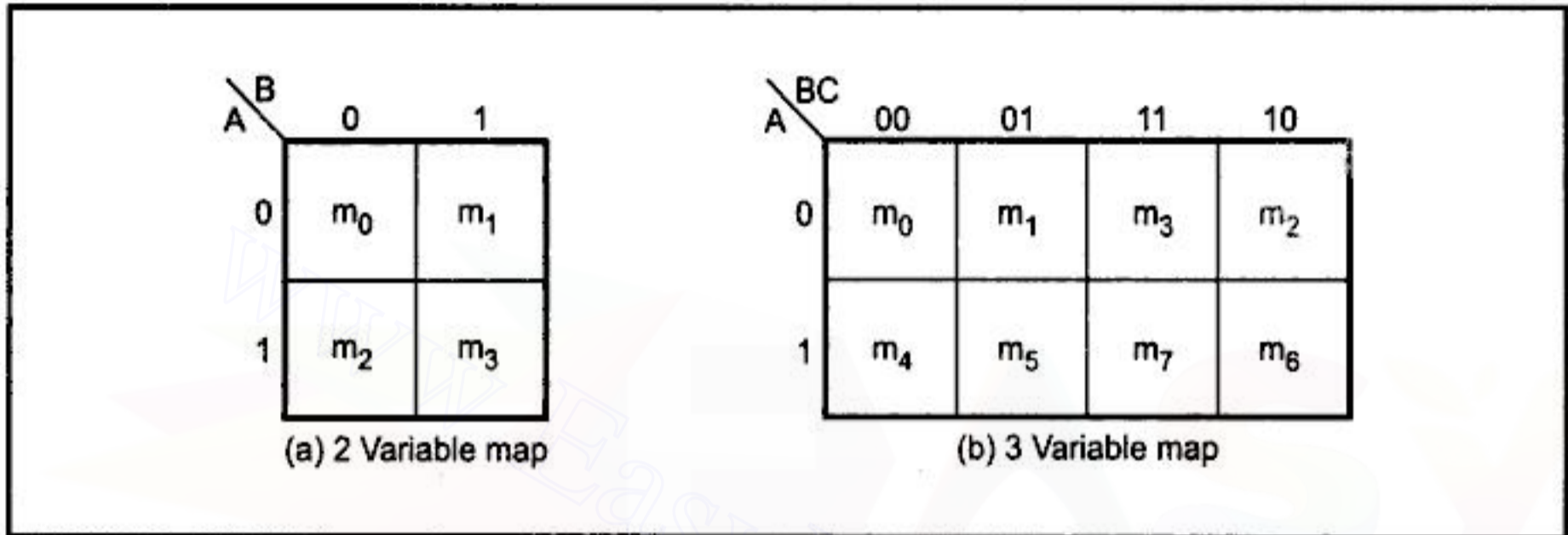


Fig. 5.6 Another way to represent 2, 3 and 4 variable maps

5.5.2 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.

Sol. : The expression has 3 variables and hence it can be plotted using 3-variable map as shown below.

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

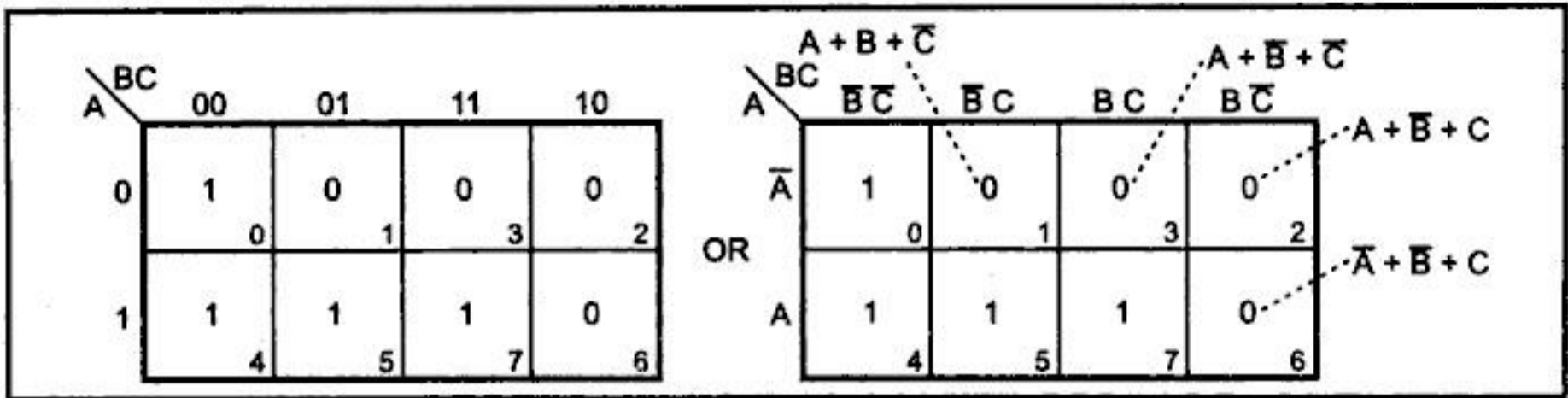


Fig. 5.10

Ex. 5.13 : Plot Boolean expression

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

Sol. : The expression has 4 variables and hence it can be plotted using 4-variable map as shown below

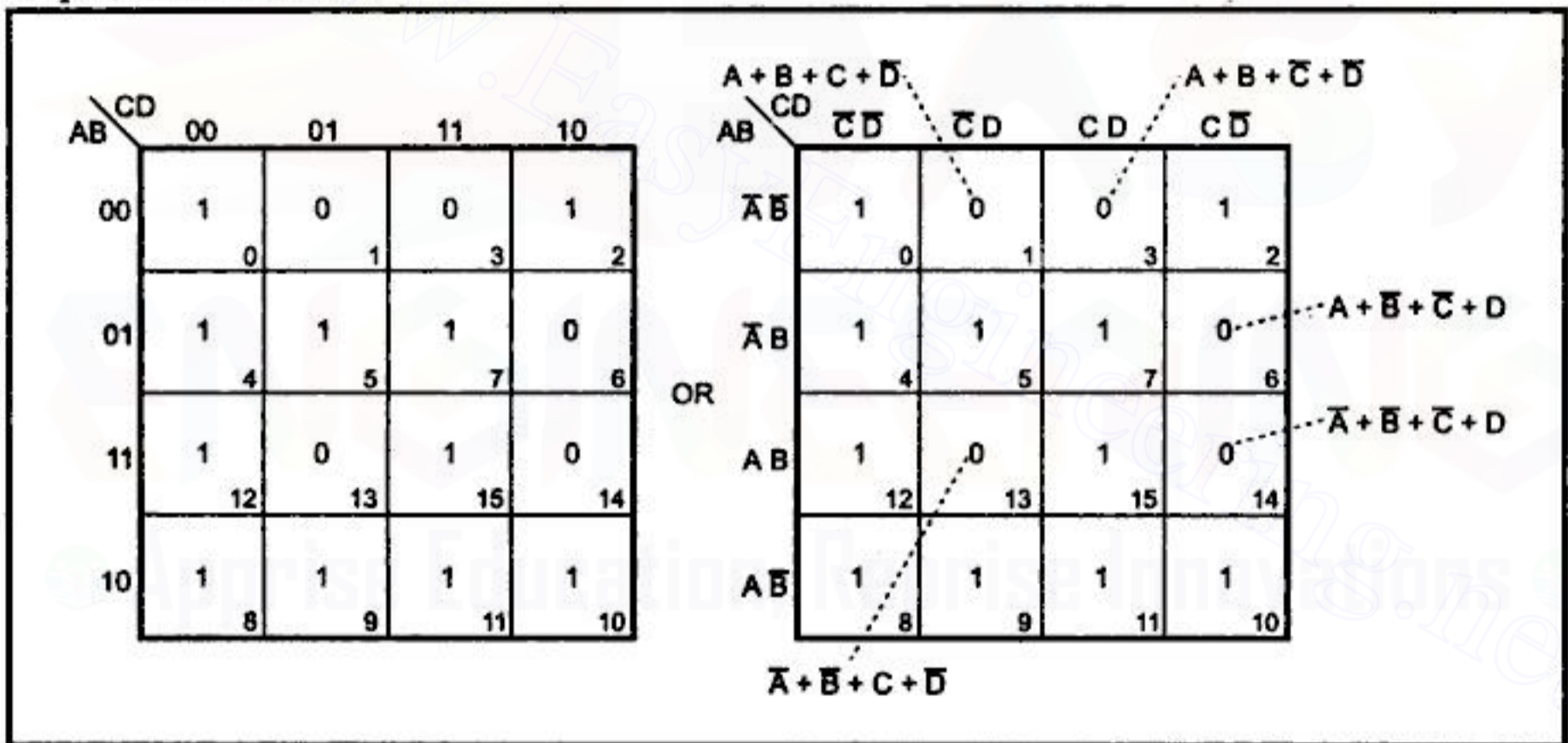


Fig. 5.11

$$(A + B + C + \bar{D}) = M_1, (A + \bar{B} + \bar{C} + D) = M_6, (A + B + \bar{C} + \bar{D}) = M_3, (\bar{A} + \bar{B} + C + \bar{D}) = M_{13}, (\bar{A} + \bar{B} + \bar{C} + D) = M_{14}$$

5.5.3 Grouping Cells for Simplification

In the last section we have seen representation of logic function on the Karnaugh map. We have also seen that minterms are marked by 1s and maxterms are marked by 0s. Once the logic function is plotted on the Karnaugh map we have to use grouping technique to simplify the logic 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. The simplification is achieved by grouping adjacent 1s or 0s in

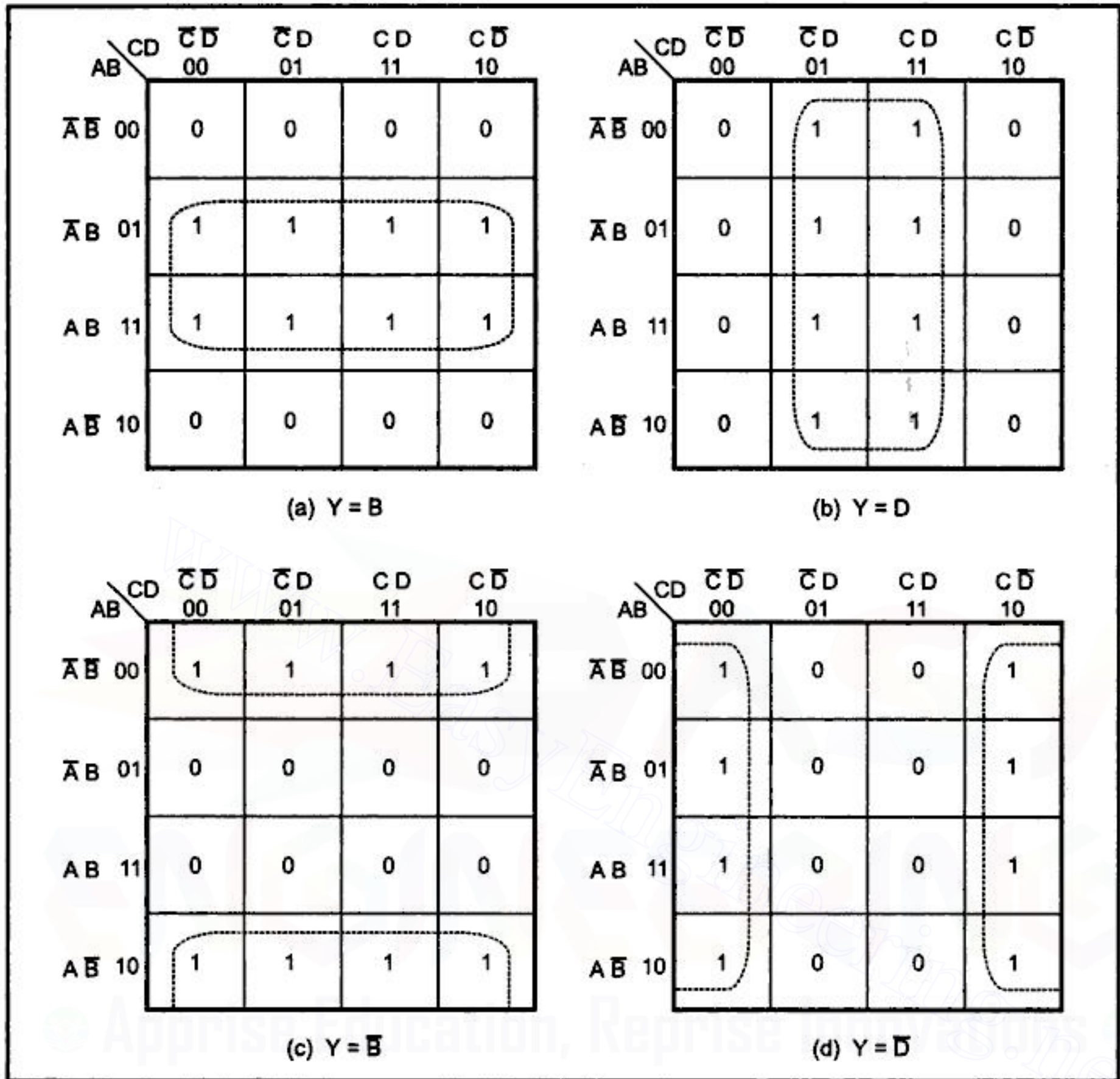


Fig. 5.14 Examples of combining octets of adjacent ones

5.5.4 Simplification of Sum of Products Expressions

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.

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.

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

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

Ex. 5.17 : Reduce the following function to its minimum sum of products form :

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

Sol. :

Step 1 : Fig. 5.18 (a) shows the K-map for four variables and it is plotted according to the given expression.

Step 2 : There are no isolated 1s

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

Fig. 5.17 (b)

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

Fig. 5.18 (a)

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

Step 5 : All 1s have already been grouped.

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

$$Y = ABD + \bar{A}D$$

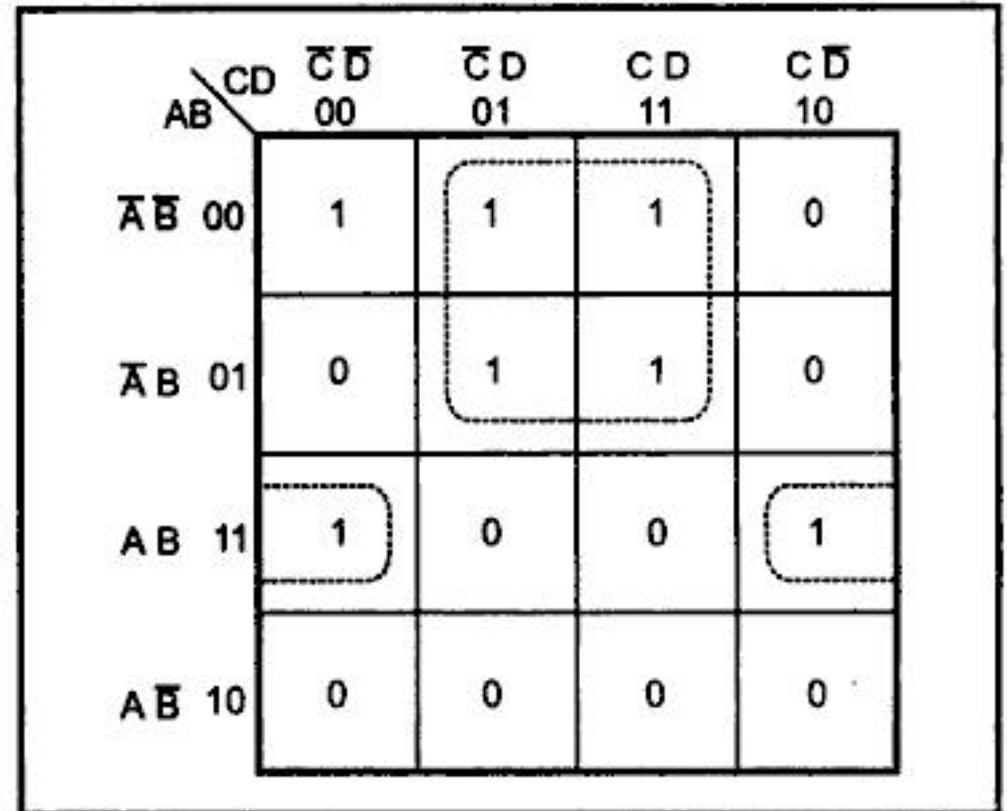


Fig. 5.21 (c)

Implementation :

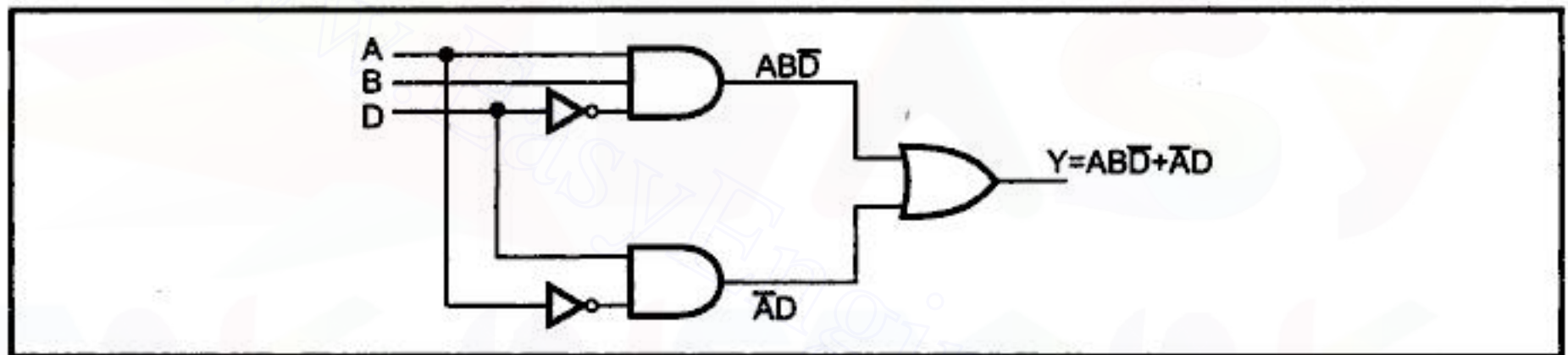


Fig. 5.22

Ex. 5.20 : Reduce the following function using K-map technique

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

Sol. :

Step 1 : Fig 5.23 (a) shows the K-map for four variables and it is plotted according to given minterms.

Step 2 : There are no isolated 1s.

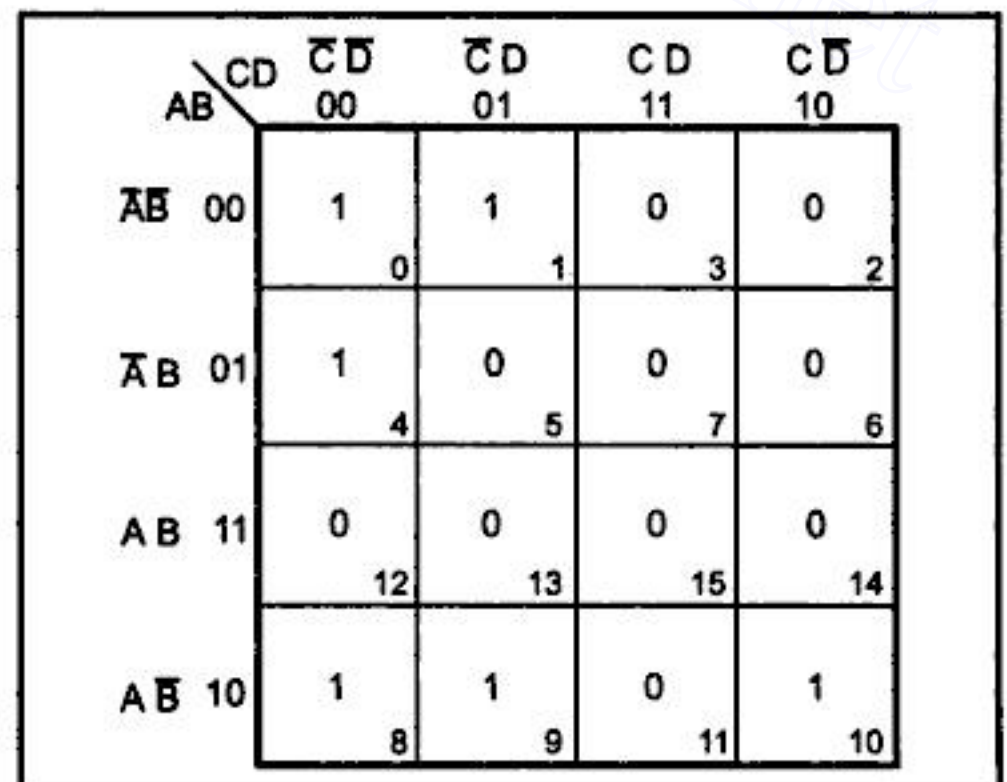


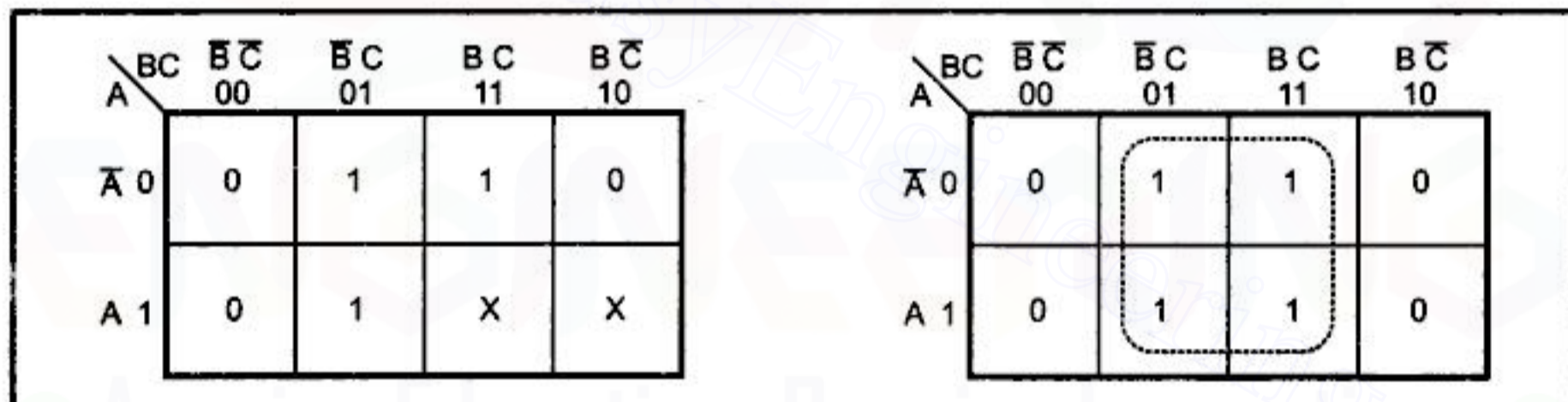
Fig. 5.23 (a)

input conditions from 0 0 0 to 1 0 1. For remaining two conditions of input, output is not defined, hence these are called don't care conditions for this truth table.

A	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 5.5

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. For example, the K-map for above truth table is shown in Fig. 5.26 (a) with X placed in the ABC and ABC cells.



(a) Fig. 5.26 Use of don't care conditions (b)

It is not always advisable to put don't cares as 1s. This is illustrated in Fig. 5.26(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.

Ex. 5.21 : Find the reduced SOP form of the following function.

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

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.

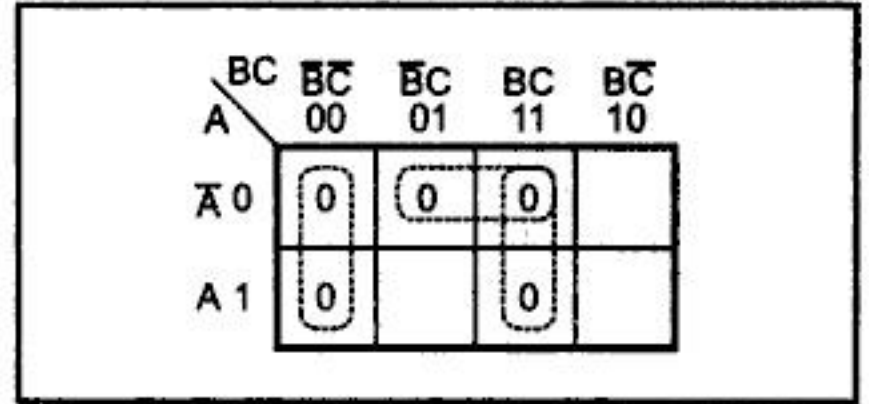


Fig. 5.31 (c)

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

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

Step 7 : $Y = \bar{Y} = \overline{\bar{B}\bar{C} + BC + \bar{A}C}$

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

It is possible to directly write the expression for Y by using DeMorgans theorem for each minterm as follows :

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

Ex. 5.26 : Minimize the following expression in the POS form

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

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

Sol. :

$$(\bar{A} + \bar{B} + C + D) = M_{12}, (\bar{A} + \bar{B} + \bar{C} + D) = M_{14},$$

$$(\bar{A} + \bar{B} + \bar{C} + \bar{D}) = M_{15}$$

$$(\bar{A} + B + C + D) = M_8, (A + \bar{B} + \bar{C} + D) = M_6, (A + \bar{B} + \bar{C} + \bar{D}) = M_7$$

$$(A + B + C + D) = M_0 \text{ and } (\bar{A} + \bar{B} + C + \bar{D}) = M_{13}$$

Step 1 : Fig. 5.32 (a) shows the K-map for four variable and it is plotted according to given maxterms.

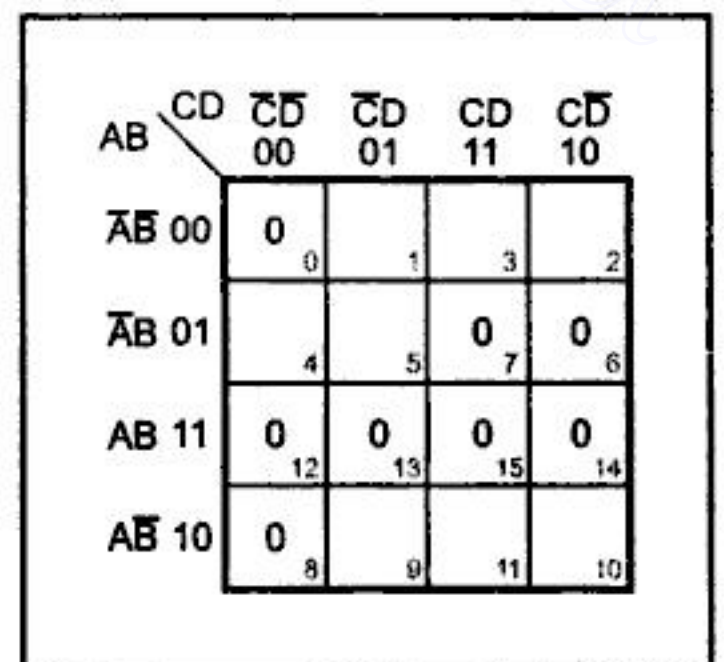


Fig. 5.32 (a)

Step 2 : There are no isolated 0s

Step 5 : All 0s have already been grouped.

Step 6 : The group 1 eliminates two variables B and D and group 2 eliminates three variables A, B and C. Then simplified expression in SOP form can be written as

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

Step 7 :

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

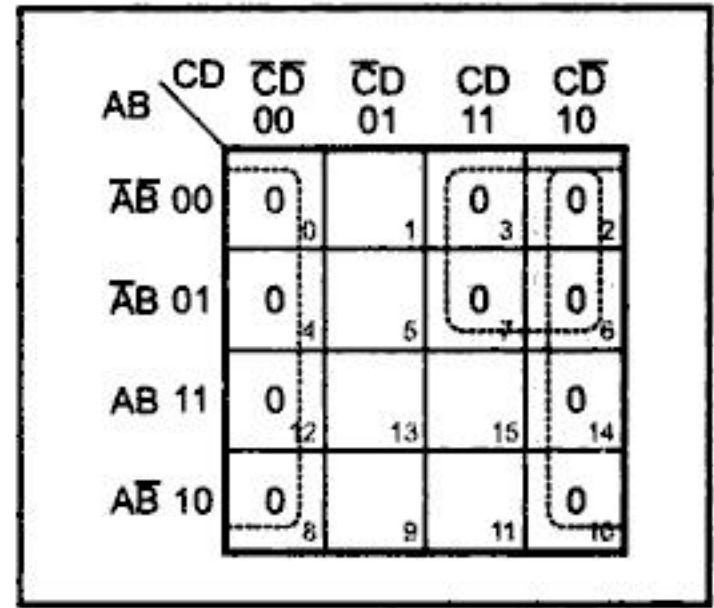


Fig. 5.34 (b)

It is possible to directly write the expression for Y by using DeMorgans theorem for each minterm as follows : $\bar{f} = \bar{A} C + \bar{D} \rightarrow f = (A + \bar{C}) D$

5.5.8 Five and Six Variable K-Maps

Karnaugh maps are most conveniently used to simplify 2, 3 and 4 variable expressions. They can also be used to simplify 5 and 6 variable expressions.

5.5.8.1 Five variable K-map

A 5-variable K-map requires $2^5 = 32$ cells, but adjacent cells are difficult to identify on a single 32-cell map. Therefore, two 16-cell K-maps are generally used. If the variables are A, B, C, D and E two identical 16-cell maps containing B, C, D and E can be constructed. One map is then used for A and the other for \bar{A} . See Fig. 5.35. It is important to note that in order to identify the adjacent grouping in the five variable map, we must imagine the two maps superimposed on one another; not "hinged" or

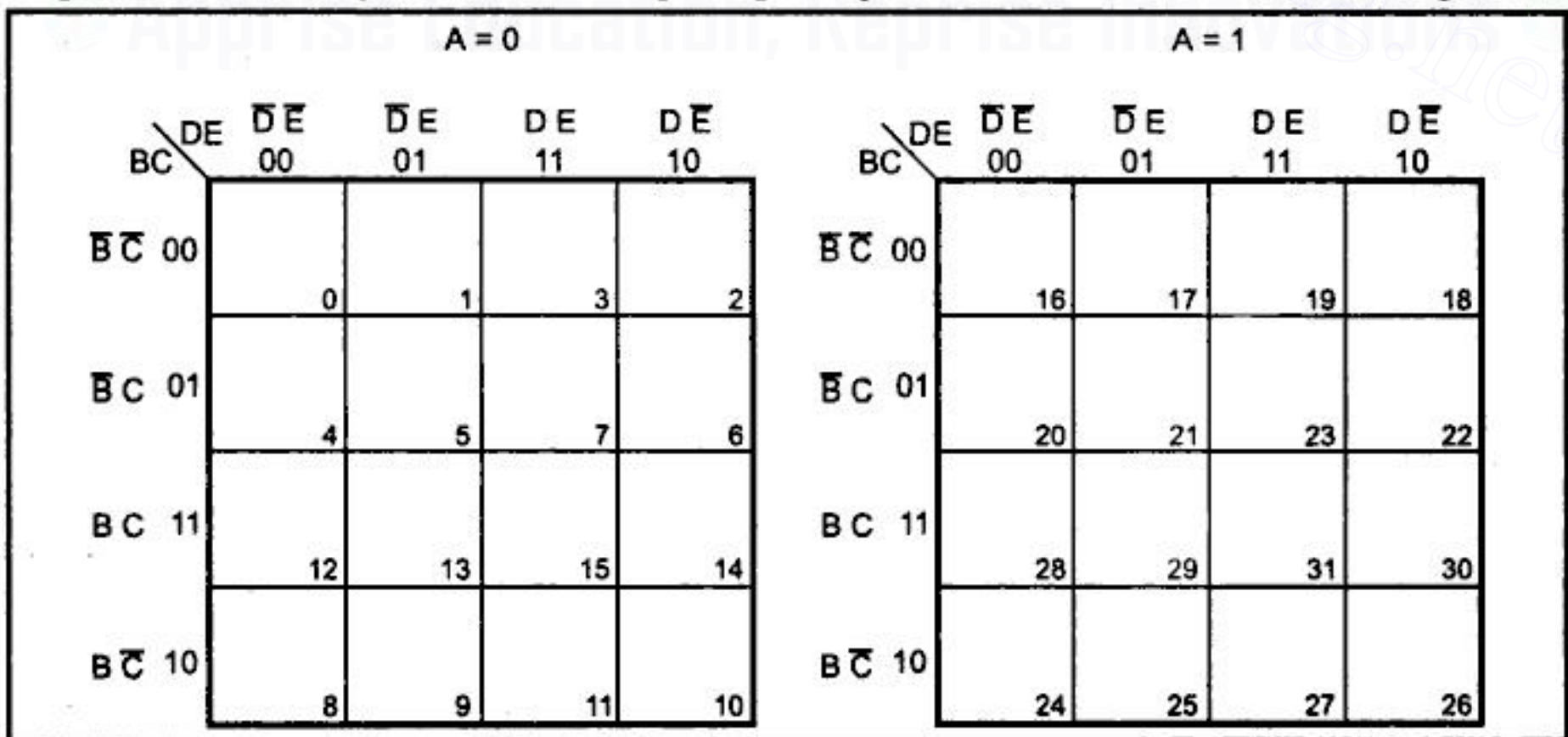


Fig. 5.35 Blank five-variable Karnaugh map

A 6-variable K-map requires $2^6 = 64$ cells. These cells are divided into four identical 16-cell maps as shown in the Fig. 5.40.

If the variables are A, B, C, D, E and F, 16-cell maps contain C, D, E and F and each 16-cell map represents four combinations of A and B. The adjacencies between entries in each cell (16-cell map) are visualized identical to that of a four-variable map.

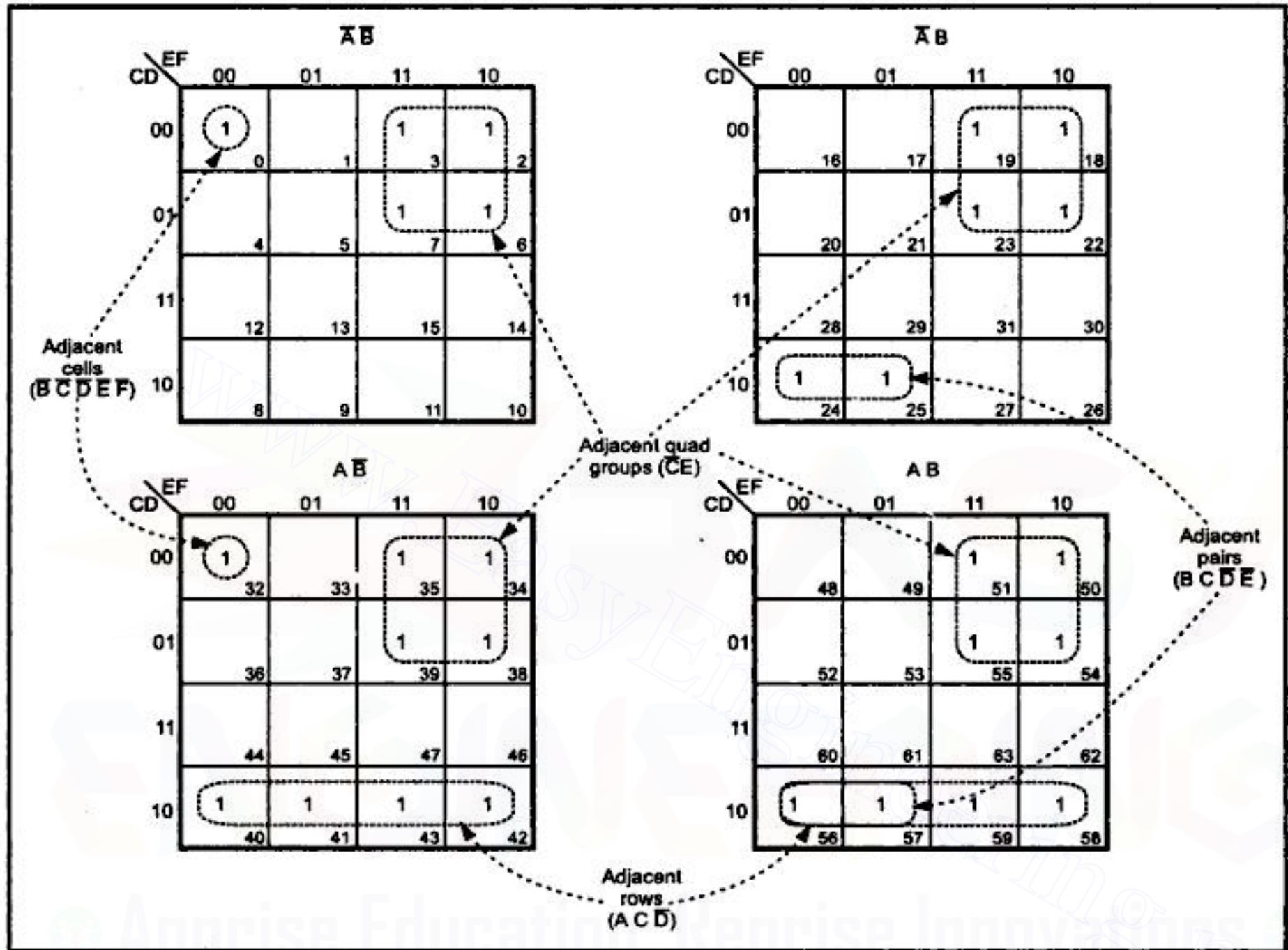


Fig. 5.41 A six-variable Karnaugh map with examples of adjacencies

Ex. 5.32 : Simplify the boolean function

$$F(A, B, C, D, E, F) =$$

$$\sum m(0, 5, 7, 8, 9, 12, 13, 23, 24, 25, 28, 29, 37, 40, 42, 44, 46, 55, 56, 57, 60, 61)$$

Sol. : Group 1 and group 2 are two pairs of 1s in the first 16-cell map. Group 3 is formed by two isolated 1s from first 16-cell map and third 16-cell map.

Group 4 is a combination of two quads from first 16-cell and second 16-cell map. Similarly group 5 is a combination of two quads from second 16-cell map and fourth 16-cell map. Group 6 is again a combination of isolated 1s from second and fourth 16-cell maps. Finally group 7 is a quad within the third 16-cell map. This gives

simplified expression as

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

Table 5.9 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.

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

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

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

Ex. 5.34 : Minimize the expression using Quine McCluskey Method.

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

Sol. :

Step 1 : List all minterms in the binary form, as shown in Table 5.10 (column (a))

Step 2 : Arrange minterms according to categories of 1s, as shown in the table (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 5.10

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.

11, 15								•			•
14, 15 ✓										•	•
1, 3, 9, 11 ✓	•		•				•	•			

Table 5.17 Prime implicant selection chart

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

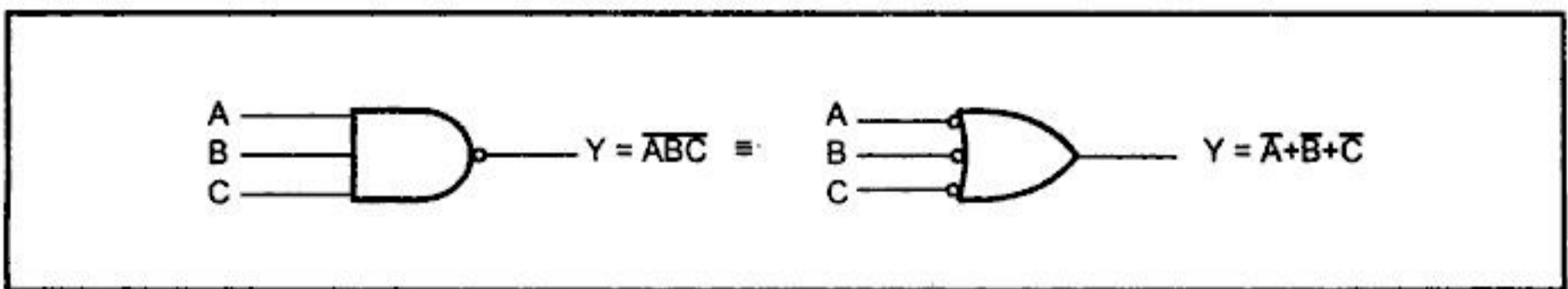
$$Y = (0 - 0 1) + (0 0 1 -) + (1 1 - 0) + (1 1 1 -) + (- 0 - 1)$$

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

5.7 NAND AND NOR Implementation

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. However, many digital circuits are constructed using NAND or NOR gates. For such implementation it is necessary to convert logic diagram from AND-OR logic to NAND- NAND or NOR-NOR logic. The implementation of Boolean function using NAND-NAND or NOR-NOR logic is presented in this section.

To facilitate the conversion to NAND-NAND or NOR-NOR logic, it is convenient to define to other graphic symbols for these gates. Fig. 5.43 shows two equivalent symbols for NAND and NOR gates. As shown in the Fig. 5.43, NAND can be represented by bubbled OR gate and NOR gate can be represented by bubbled AND gate.



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

Demorgan's Theorem 1

(a) Two graphic symbols for NAND gate

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

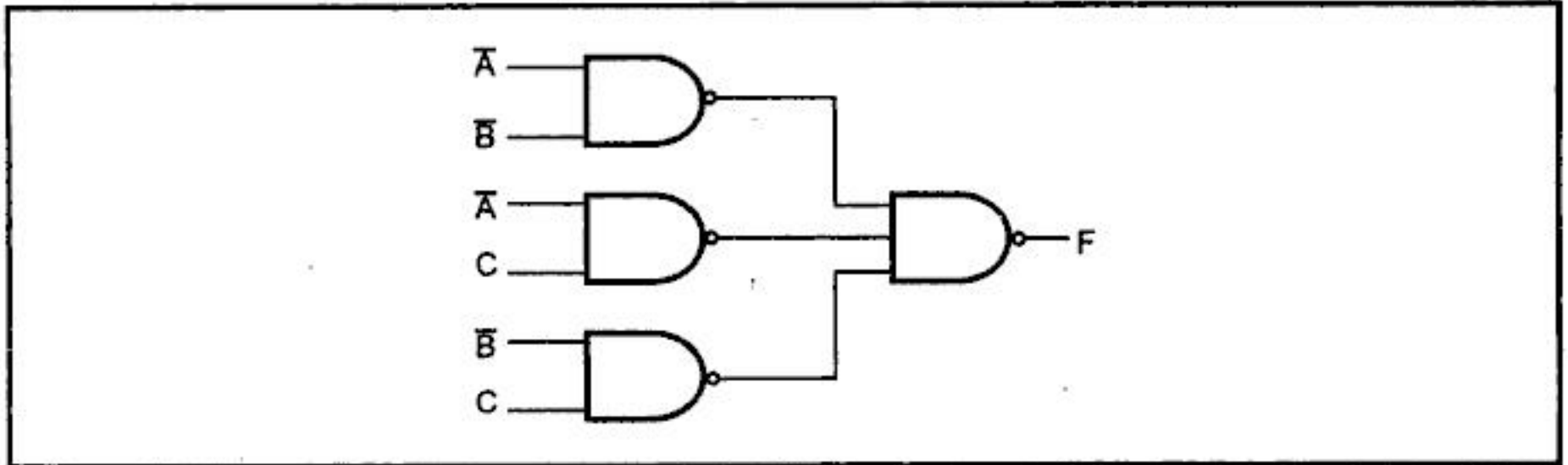


Fig. 5.49

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

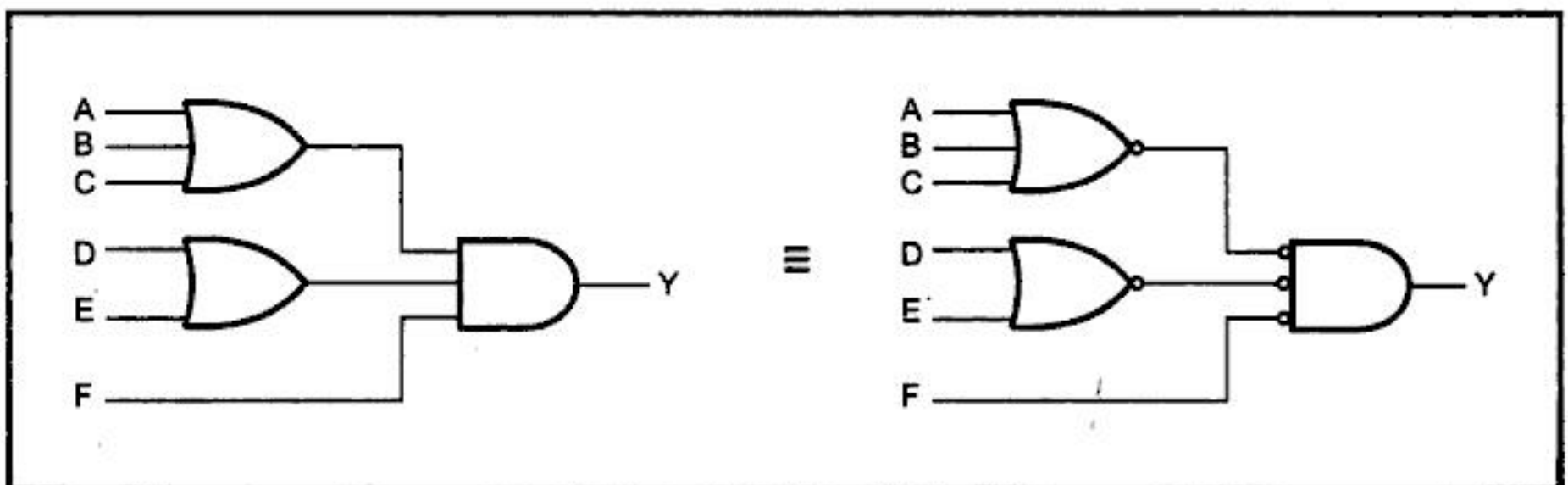
5.7.2 NOR-NOR Implementation

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. 5.50 (a). Fig. 5.50 (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. 5.50 (b) is equivalent to implementation shown in Fig. 5.50 (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 again complemented by bubble to produce the normal value of F .



(a) OR-AND

(b) NOR-Bubbled AND

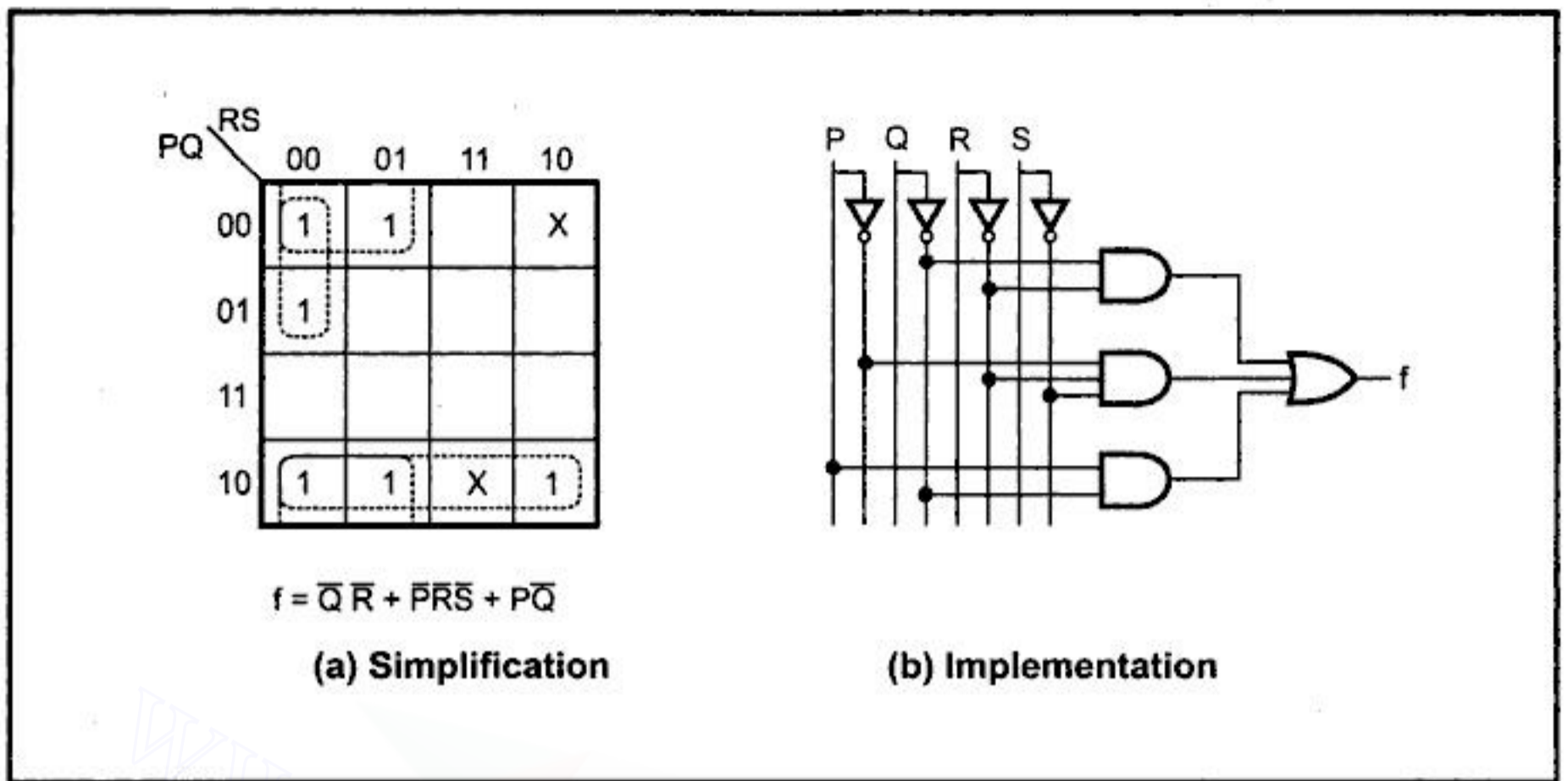


Fig. 5.58

ii) $f(A, B, C, D) = \pi M(0, 2, 4, 10, 11, 14, 15)$
 $f = (\bar{A} + \bar{C})(A + C + D)(A + B + D)$

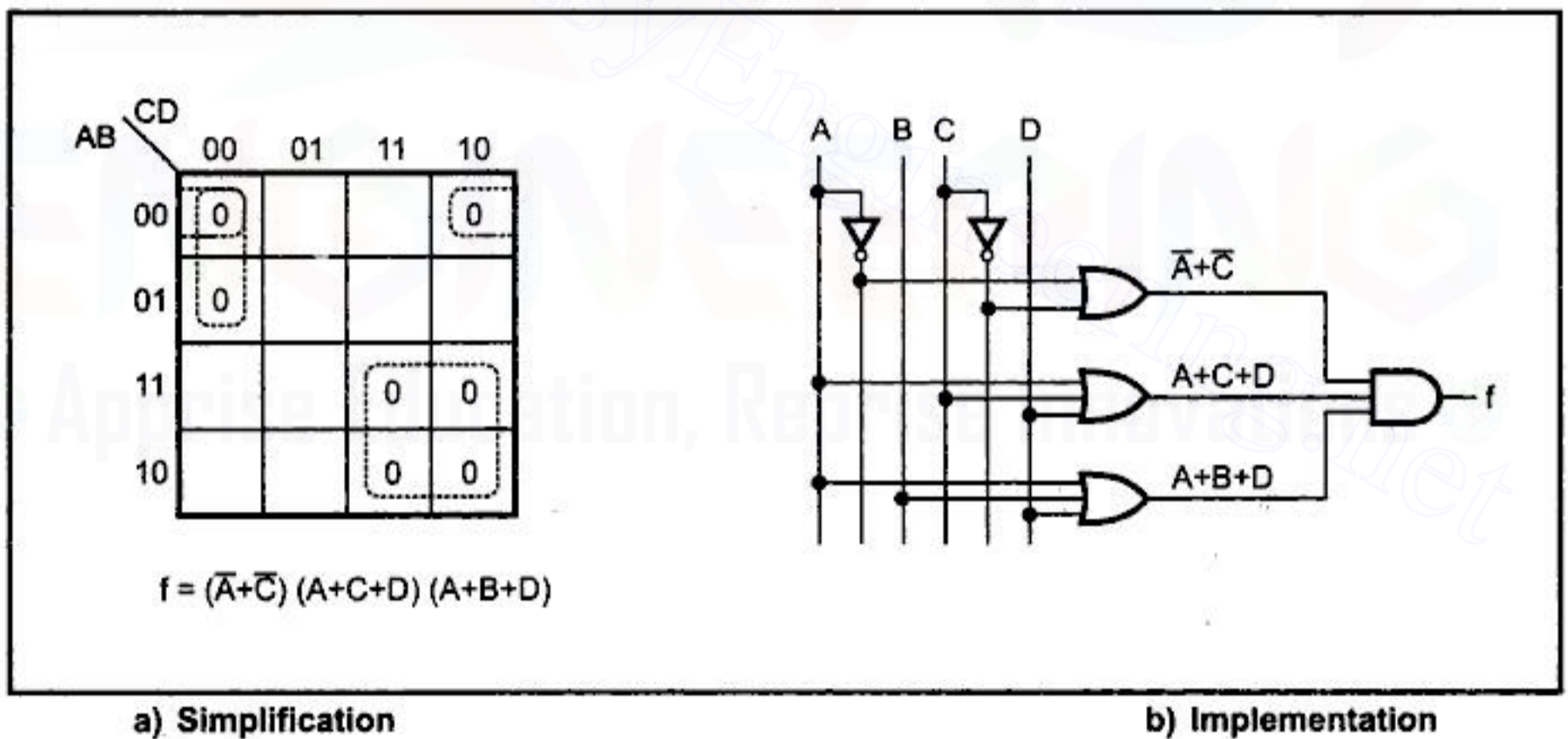


Fig. 5.59

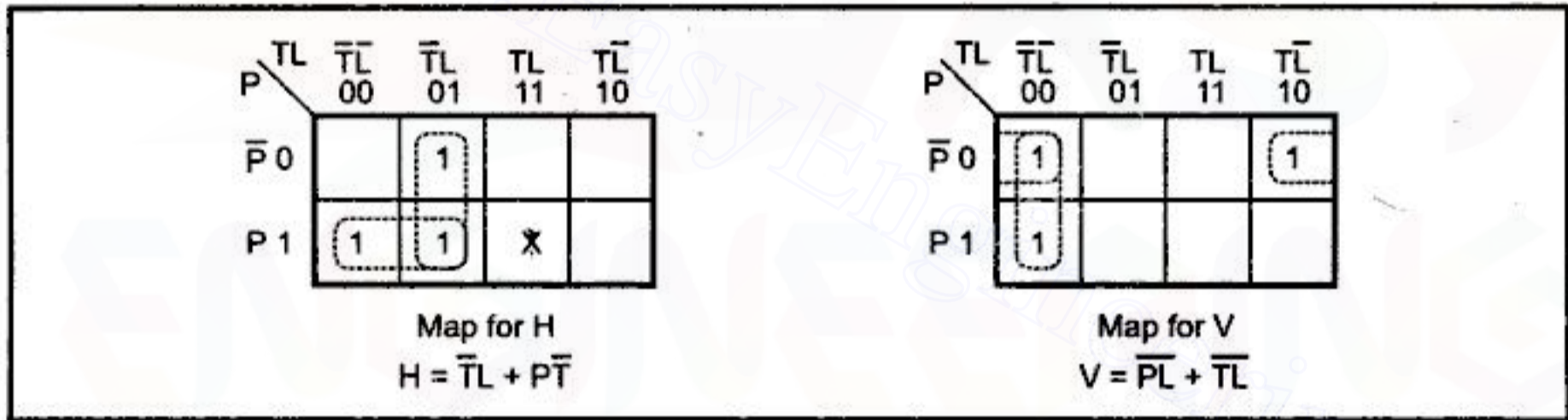
iii) $f(A, B, C, D) = \bar{A}\bar{B}D + AB\bar{C}\bar{D} + \bar{A}BD + ABC\bar{D}$
 $= \bar{A}\bar{B}D(C + \bar{C}) + AB\bar{C}\bar{D} + \bar{A}BD(C + \bar{C})$
 $+ ABC\bar{D}$
 $= \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D + AB\bar{C}\bar{D} + \bar{A}BCD$
 $+ \bar{A}B\bar{C}D + ABC\bar{D}$
 $f = \bar{A}D + AB\bar{D}$

Sol. : (a)

Inputs			Outputs	
P	T	L	H	V
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	X	X

Table 5.20

b) SOP Expressions

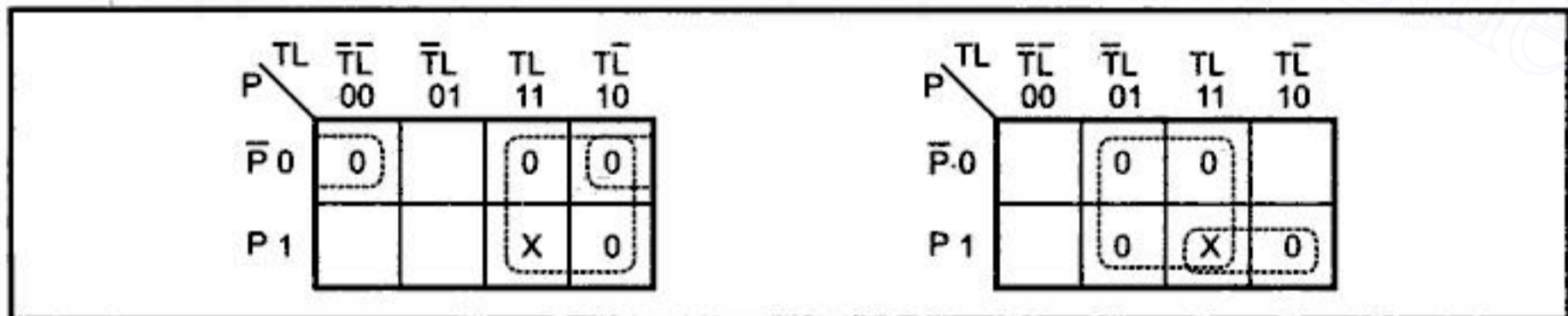


(a)

Fig. 5.65

(b)

POS Expressions



(a)

Fig. 5.66

(b)

$$\begin{aligned} \bar{H} &= \bar{P}\bar{L} + T \\ \therefore \overline{\bar{H}} &= \overline{\bar{P}\bar{L} + T} \\ \therefore H &= \overline{\bar{P}\bar{L}} \cdot \bar{T} \\ &= (\overline{\bar{P} + \bar{L}}) \cdot \bar{T} \\ &= (P + L) \cdot \bar{T} \end{aligned}$$

$$\begin{aligned} \bar{V} &= L + PT \\ \therefore \overline{\bar{V}} &= \overline{L + PT} \\ \therefore V &= \bar{L} \cdot \overline{PT} \\ &= \bar{L} \cdot (\bar{P} + \bar{T}) \end{aligned}$$

c) Implementation of logic circuit

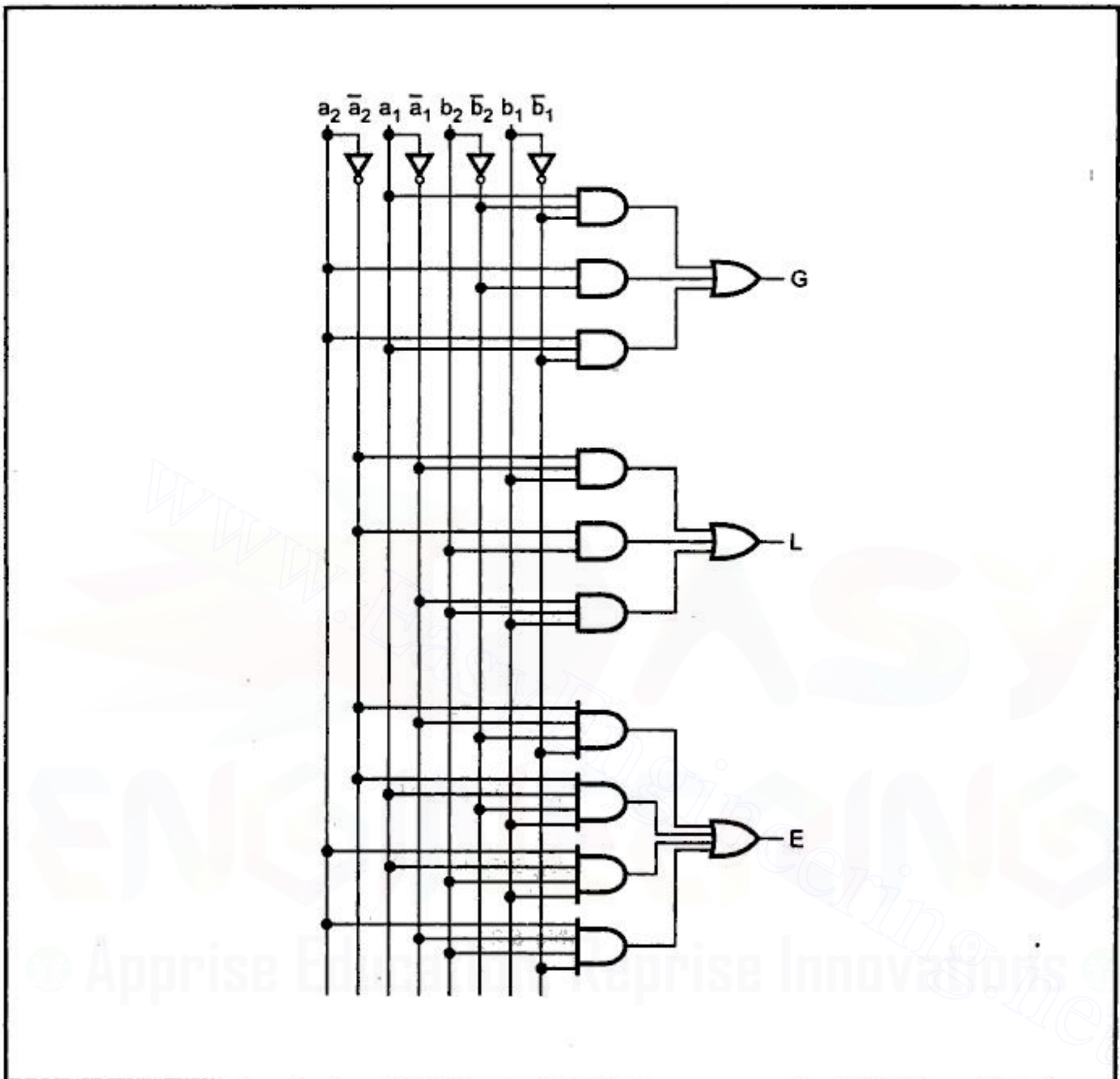


Fig. 5.70

Ex. 5.47 : The input to a combinational logic circuit is a 4-bit binary number. Design the logic circuit with minimum hardware for the following.

- i) Output $Y_1 = 1$ if the input binary number is 5 or less than 5.
- ii) Output $Y_2 = 1$ if the input binary number is 9 or more than 9.

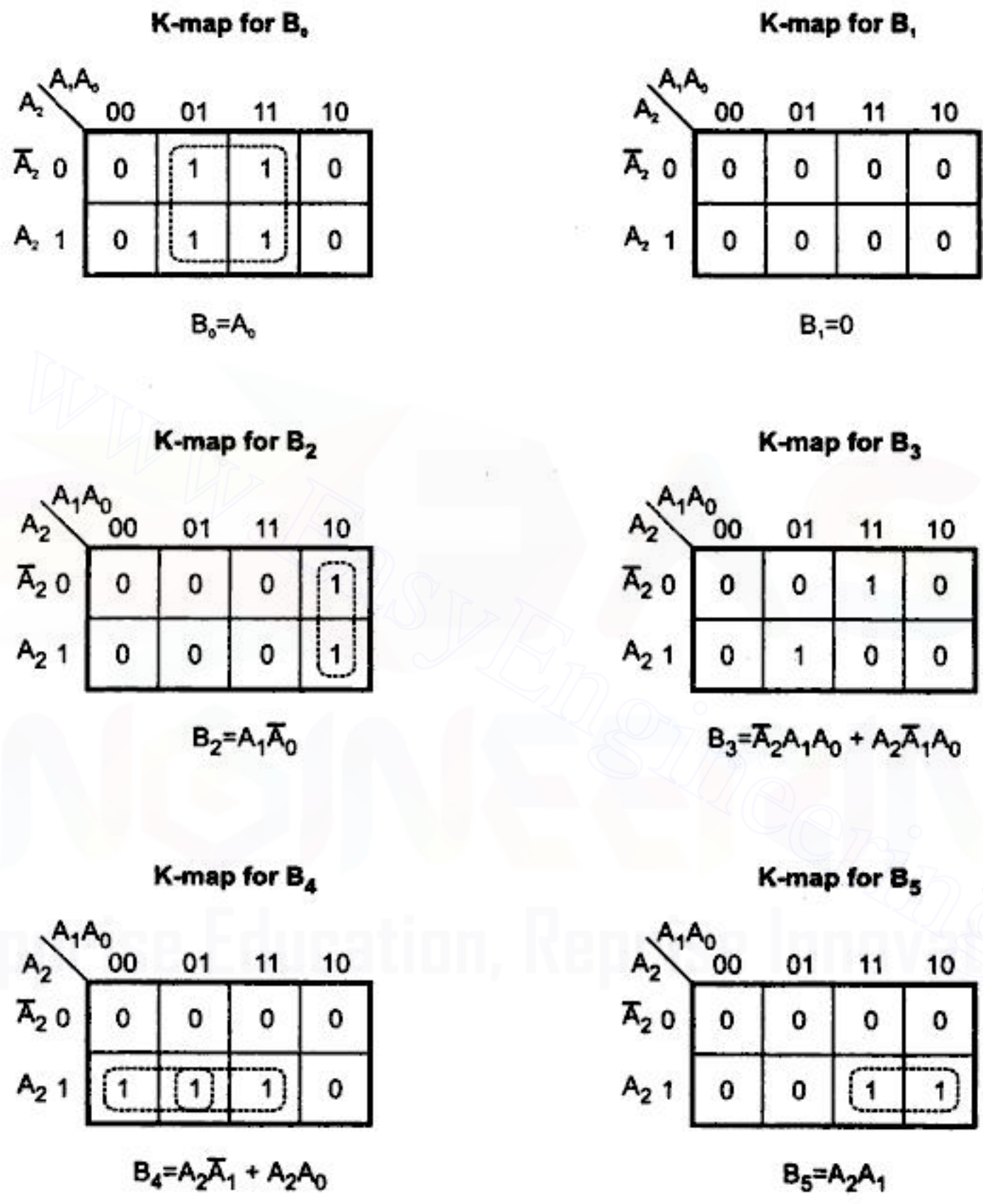


Fig. 5.76

Implementation of the circuit is as shown in the Fig. 5.77.

$$Y = DC + DA + BA + CB$$

Logic diagram

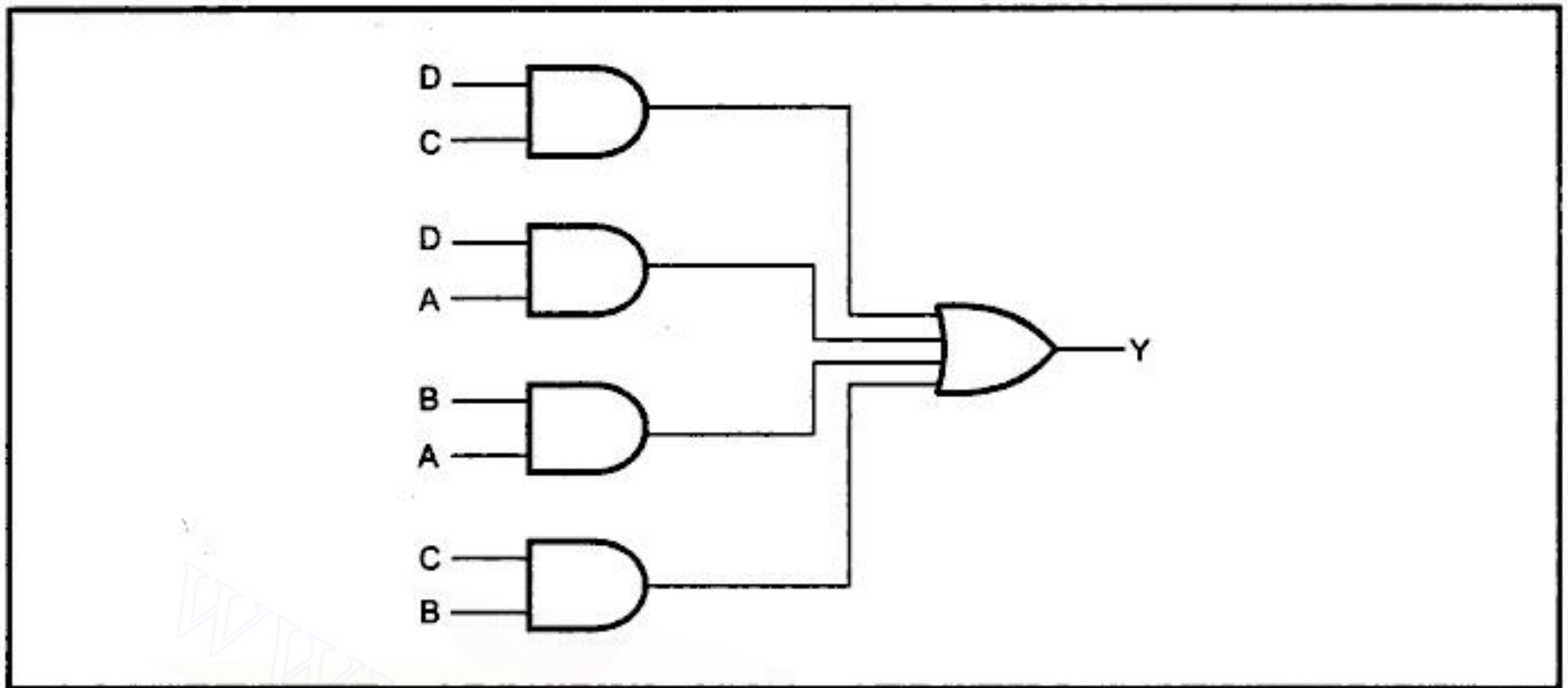


Fig. 5.80

Logic diagram using NAND gates only

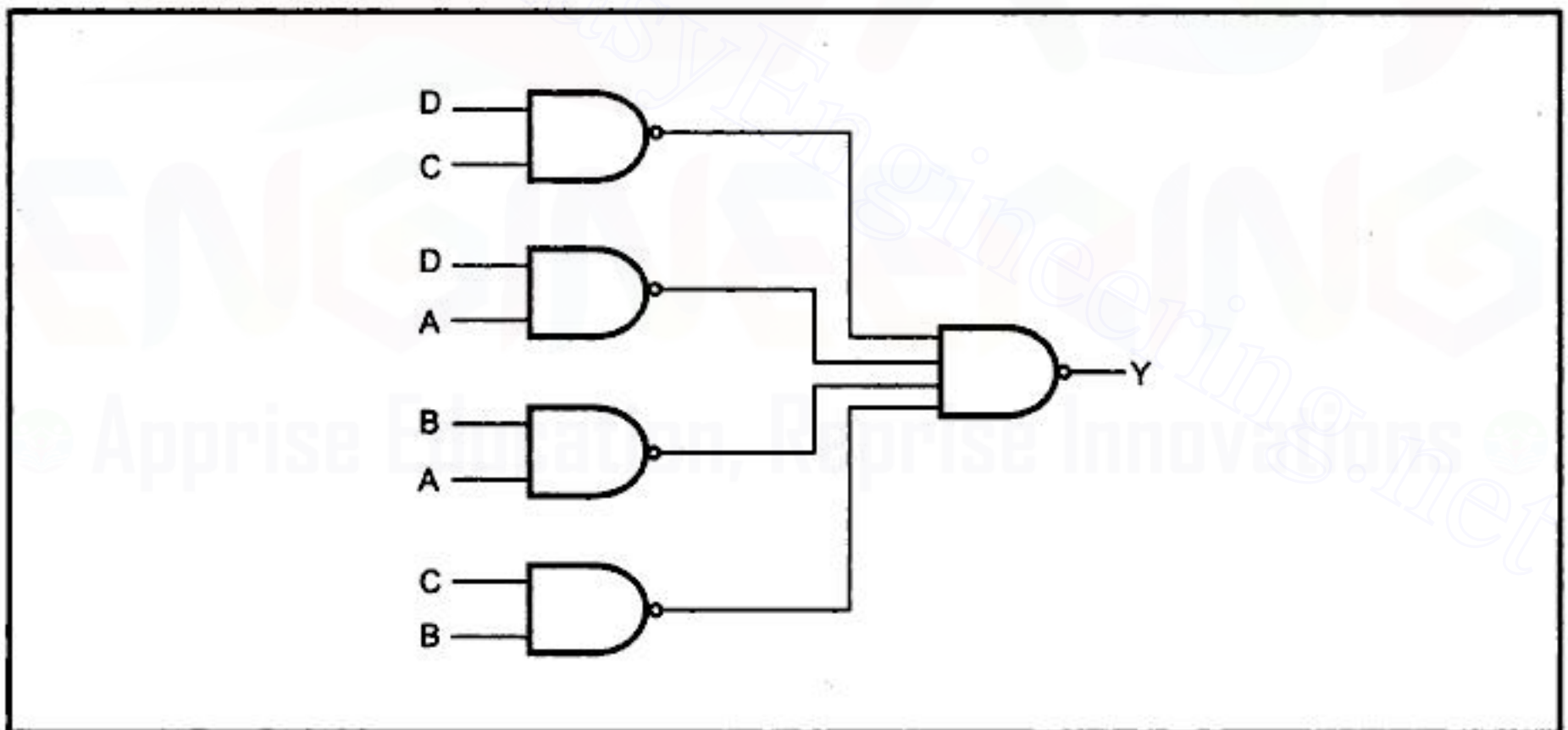


Fig. 5.81

Ex. 5.52 : Simplify the following 5 variable Boolean expression using Quine McCluskey method.

$$F = \sum m (0,1,9,15,24,29,30) + d (8,11,31)$$

Sol. : **Step 1 :** List all minterms in the binary form, as shown in the table (column (a)).

Digital Electronics **239** **Simplification of Boolean Functions**

Minterms	Binary Representation	Minterms	Binary Representation
0, 8	00-000	8, 9, 12, 13	001-0- ✓
8, 9	00100- ✓	8, 9, 24, 25	0-100- ✓
8, 12	001-00 ✓	8, 12, 24, 28	0-1-00 ✓
8, 24	0-1000 ✓	8, 12, 40, 44	-01-00 ✓
8, 40	-01000 ✓	8, 24, 40, 56	--1000 ✓
5, 7	0001-1	9, 13, 25, 29	0-1-01 ✓
5, 13	00-101	12, 13, 28, 29	0-110- ✓
5, 37	-00101	12, 44, 28, 60	--1100 ✓
9, 13	001-01, ✓	24, 25, 28, 29	011-0- ✓
9, 25	0-1001 ✓	24, 25, 56, 57	-1100- ✓
12, 13	00110- ✓	24, 56, 28, 60	-11-00 ✓
12, 28	0-1100 ✓	40, 42, 44, 46	101--0
12, 44	-01100 ✓	40, 56, 44, 60	1-1-00
24, 25	01100- ✓	25, 29, 57, 61	-11-01 ✓
24, 28	011-00 ✓	28, 29, 60, 61	-1110- ✓
24, 56	-11000 ✓	56, 57, 60, 61	111-0- ✓
40, 42	1010-0		
40, 44	101-00 ✓		
40, 56	1-1000 ✓		
7, 23	0-0111		
13, 29	0-1101 ✓		
25, 29	011-01 ✓		
25, 57	-11001 ✓		
28, 29	01110- ✓		
28, 60	-1110- ✓		
42, 46	101-10 ✓		
44, 46	1011-0 ✓		
44, 60	1-1100 ✓		
56, 57	11100- ✓		
56, 60	111-00 ✓		
23, 55	-10111		
29, 61	-11101 ✓		
57, 61	111-01 ✓		
60, 61	11110- ✓		

Table 5.30

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

Fig. 5.95

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

Logic diagram

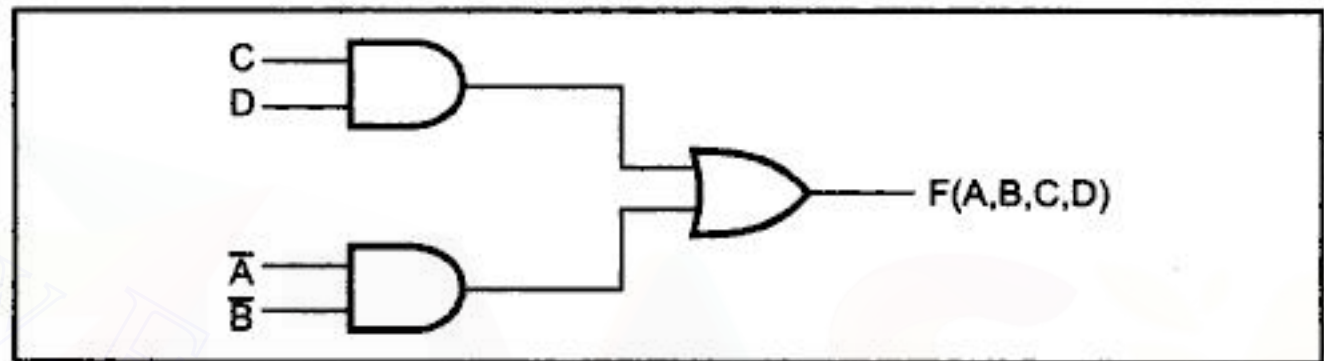


Fig. 5.96

b) ii) Product of sums :

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

$$= \sum \pi(4, 6, 8, 9, 10, 12, 13, 14) + d \sum (0, 2, 5)$$

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

Fig. 5.97

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

Logic diagram

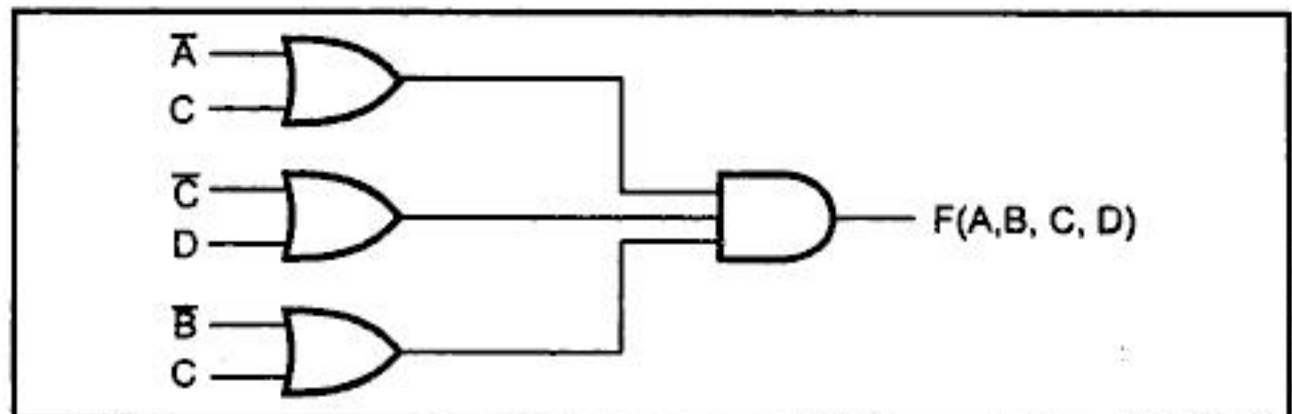


Fig. 5.98

Ex. 5.59 : Reduce the following equation using Quine McCluskey method of minimisation :

$$F(A, B, C, D) = \sum m(0, 1, 3, 4, 5, 7, 10, 13, 14, 15)$$

(May-2000)

Sol. :

Step 1: List all minterms in the binary form as shown in table 5.35 (Column (a)).

Minterm	Binary Representation	Minterms	Binary Representation
m_0	0000	m_0	0000 ✓
m_1	0001	m_1	0001 ✓
m_3	0011	m_4	0100 ✓
m_4	0100	m_3	0011 ✓
m_5	0101	m_5	0101 ✓
m_7	0111	m_{10}	1010 ✓
m_{10}	1010	m_7	0111 ✓
m_{13}	1101	m_{13}	1101 ✓
m_{14}	1111	m_{14}	1110 ✓
m_{15}	1111	m_{15}	1111 ✓
Column (a)		Column (b)	

Table 5.35

Step 2: Arrange the minterms according to number of 1s, 9s shown in the table 5.35 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.

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, 1	000- ✓	0, 1, 4, 5	0-0-
0, 4	0-00 ✓	0, 4, 1, 5	0-0-
1, 3	00-1 ✓	1, 3, 5, 7	0--1
1, 5	0-01 ✓	1, 5, 3, 7	0--1
4, 5	010- ✓	5, 7, 13, 15	-1-1

$$\text{Ans. : a) } A \bar{B} + A B ; (A + B) (A + \bar{B})$$

$$\text{b) } \bar{A} \bar{B} \bar{C} + \bar{A} B C + A \bar{B} \bar{C} + A \bar{B} C + A B C ; \\ (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (\bar{A} + \bar{B} + C)$$

Q.2 Simplify following logical expressions using Karnaugh maps

$$\text{i) } Y = A \bar{B} + A B + \bar{A} B$$

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

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

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

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

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

Q.3 Express the following function in standard SOP form

$$F_1 = A B + \bar{C} D + A \bar{B} C$$

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

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

Q.4 Simplify the following Boolean expressions using Karnaugh map.

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

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

$$\text{ii) } Y = \bar{Q} R S + R \bar{S} + \bar{P} \bar{Q} R \bar{S} + \bar{P} \bar{Q} R$$

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

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

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

$$\text{iv) } Y = (\bar{P} + Q + \bar{R}) (P + Q + R) (P + Q + \bar{R})$$

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

Q.5 Simplify the following functions

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

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

$$\text{ii) } f_2(A, B, C, D) = \sum m(0, 1, 3, 8, 9, 13, 15)$$

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

$$\text{iii) } f_3(A, B, C, D) = \sum m(0, 1, 2, 3, 11, 12, 14, 15)$$

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

$$\text{iv) } f_4(W, X, Y, Z) = \sum m(0, 3, 4, 7, 9, 12, 14)$$

$$\text{Ans. : } W X \bar{Z} + \bar{W} (Y Z + \bar{Y} \bar{Z}) + W \bar{X} \bar{Y} Z$$

Q.6 Design a logic circuit to provide an output when any two or three of four switches are closed.

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

Q.7 Use a Karnaugh map to reduce each expression to a minimum sum of products form

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

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

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

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

Q.8 Use the Quine- McCluskey method to simplify the following function

$$\text{i) } f_1(A, B, C, D, E) = \sum m(8, 9, 10, 11, 13, 15, 16, 18, 21, 24, 25, 26, 27, 30, 31)$$

$$\text{Ans. : } f_1 = \bar{A} B E + A \bar{C} \bar{E} + A B D + B \bar{C} + A \bar{B} C \bar{D} E$$

$$\text{ii) } f_2(A, B, C, D) = \sum m(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$$

$$\text{Ans. : } f_2 = A \bar{B} \bar{C} + \bar{C} D + \bar{B} D + A D$$

Q.9 Reduce by K-mapping and implement using NAND-NAND logic

$$Y = \sum m(0, 1, 3, 4, 5, 7, 10, 13, 14, 15)$$

16. Draw Karnaugh map and simplify the following Boolean functions. (Dec-95)
- a) $Y(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$ Ans. : $A + \bar{B} + C + D$
- b) And $y(A, B, C, D) = \sum m(0, 1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15)$ Ans. : $A + \bar{D} + \bar{B}$
- c) $y(A, B, C, D) = \pi M(2, 5, 6, 10) + \pi d(1, 4, 14, 15)$ Ans. : $\bar{C}\bar{D} + CD + AB + \bar{B}\bar{C}$ - SOP form
17. Express the following function in a sum of minterms and product of maxterms.
- i) $F(A, B, C, D) = D(\bar{A} + B) + \bar{B}D$ ii) $F(w, x, y, z) = \bar{y}z + wx\bar{y} + wxz + \overline{wxz}$ (May-96)
18. Using K-map find SOP and POS forms of the following and state which is less expensive :
 $F = \sum m(0, 1, 3, 5, 6, 7, 10, 14, 15)$ (Dec-96)
19. Design a combinational logic circuit that accepts a 4-bit data and show
- i) High output when it sees even number of 1's at the input.
 ii) Low output when odd number of 1's at input. (Dec-96)
20. Compare and contrast POS and SOP (Dec-96, Dec-98)
21. Prove that the maxterm with subscript j is complement of the minterm with the same subscript j :
 i.e. $1/M = M$. (Dec-96)
22. Express the following in SOP form :
 1) $f(s, y, z) = 1$ 2) $f(x, y, z) = (xy + z)(z + zy)$. (May-97)
- Ans. : i) $xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + \bar{x}\bar{y}\bar{z}$, ii) $xyz + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z$
23. Draw the K-map and simplify the following function
 $F(A, B, C, D) = \sum m(0, 2, 5, 7, 8, 10, 13, 15)$. (May-97)
24. Simplify and implement the following function using NOR gates only :
 $F = \pi M(4, 6, 10, 12, 13, 15)$. (May-97)
25. Convert the following equation in standard SOP and POS form. (May-97)
- Ans. : $f = \sum m(3, 5, 6, 7)$, $f = \pi m(0, 1, 2, 4)$
26. Design a combinational circuit whose input is 4-bit binary number and whose output is 2's complement of input number. (Dec-97)
27. Obtain the simplified expression in the sum of product for the following :
 i) $xy + \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z}$ ii) $AB + BC + BC$. (Dec-97)
- Ans. : i) $\sum m(0, 2, 6, 7)$, ii) $\sum m(2, 6, 7)$
28. Reduce the following functions using K-map technique :
- (a) $f = \sum m[0, 1, 4, 8, 9, 10] + d[2, 11]$
 (b) $f = \sum m[8, 9, 10, 11, 13, 15, 16, 18, 21, 24, 25, 26, 27, 30, 31]$
 (c) $f = \pi M(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$
 (d) $f = \pi M(2, 7, 8, 9, 10, 12)$. (May-98)
- Ans. : (a) $A\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}\bar{D}$, (b) $B\bar{C} + BE + ABD + A\bar{B}\bar{C}\bar{E} + A\bar{B}C\bar{D}E$, (c) $\bar{C}\bar{D}$,
 (d) $(B + \bar{C} + D)(\bar{A} + C + D)(C + \bar{A} + B)(A + \bar{B} + \bar{C} + \bar{D})$
29. The product of all max terms of a Boolean function of n variables is 0. Prove the statements for $n = 3$.
 (May-98)

6.2 Transistor as a Switch

Transistors are widely used in digital logic circuits and switching applications. In these applications, the voltage levels periodically alternate between a 'LOW' and a 'HIGH' voltage, such as 0V and +5V. In switching circuits, a transistor is operated at cutoff for the OFF condition, and in saturation for the ON condition. The active linear region is passed through very quickly in switching from cutoff to saturation or vice versa. In switching applications, the active region is of no interest.

In cutoff region, both the transistor junctions, viz. junction between emitter and base and the junction between base and collector are reverse biased and only the reverse current, which is very small and practically negligible, flows in the transistor.

In the saturation condition, both the junctions are in forward bias and the values of voltage drops between collector and emitter, $V_{CE(sat)}$; and between emitter and base $V_{BE(sat)}$, are small. Saturation voltages range from a few tenths of a volt to a volt, depending on the transistor type. In saturation condition, the collector current is comparatively large, and is controlled by the external resistance connected in the collector circuit. The basic transistor circuit used in switching applications is called an "inverter", the n-p-n version of which is shown in Fig. 6.1

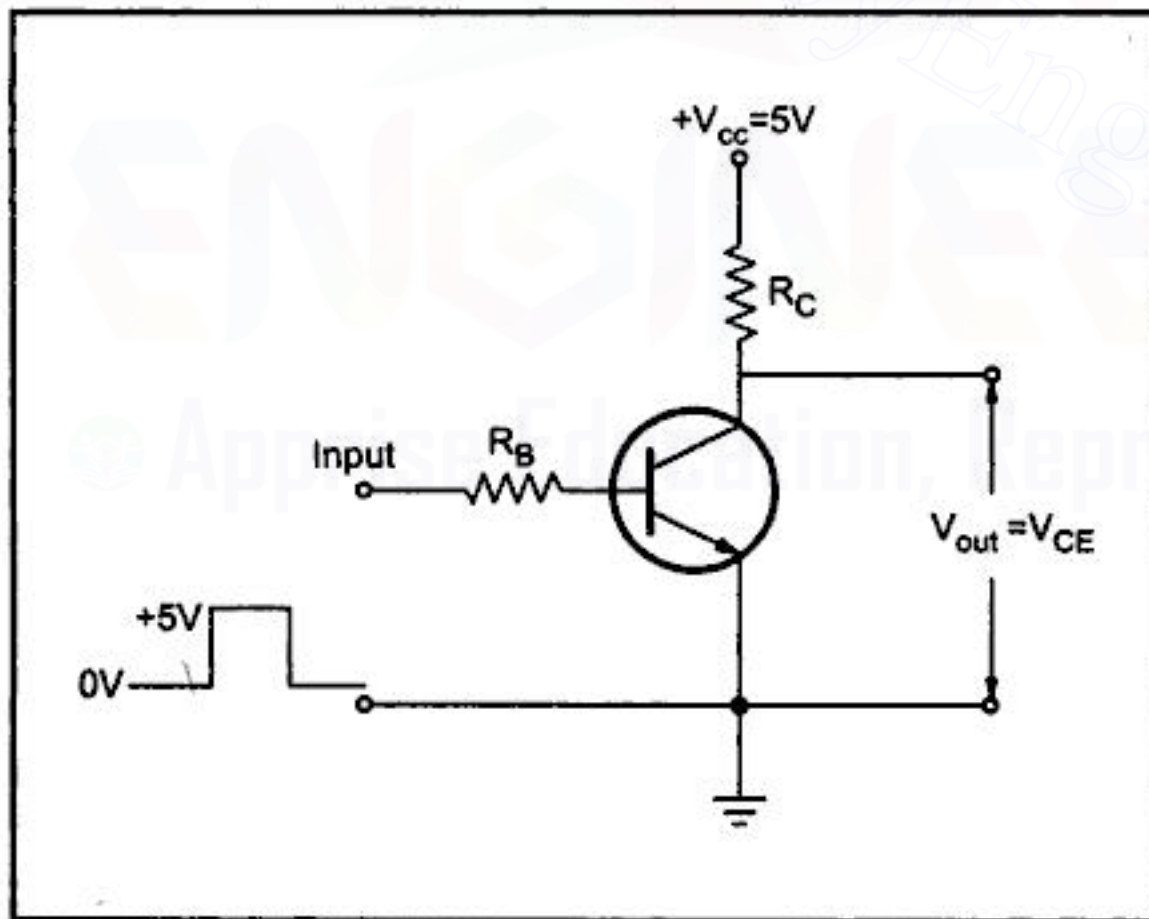


Fig. 6.1 Transistor as a switch

V_{CC} and the "HIGH" level of the input are both +5V.

When the input to the inverter is HIGH (+5V), the base-emitter junction is forward biased and current flows through R_B into the base. The values of R_B and R_C are chosen so that the amount of base current flowing is enough to saturate the transistor, that is, to drive it into the saturation region of its output characteristics. For saturation, the base current

In the Fig. 6.1, the transistor is shown to be connected in common-emitter configuration. A square-wave input is applied between the base and emitter through a base resistance R_B and the output is taken between collector and emitter. Thus the output voltage V_{out} is the voltage between the collector and emitter, i.e. V_{out} is V_{CE} .

In switching applications, the transistor operates either in cutoff region or saturation region of the output characteristics, and these regions are illustrated in Fig. 6.2. In the circuit shown,

$V_{OH(min)}$ - High-Level Output Voltage : It is the minimum voltage level at a logic circuit output in the logical 1 state under defined load conditions.

$V_{OL(max)}$ - Low-Level Output Voltage : It is the maximum voltage level at a logic circuit output in the logical 0 state under defined load conditions.

I_{IH} - High-Level Input Current : It is the current that flows into an input when a specified high-level voltage is applied to that input.

I_{IL} - Low-Level Input Current : It is the current that flows into an input when a specified low-level voltage is applied to that input.

I_{OH} - High-Level Output Current : It is the current that flows from an output in the logical 1 state under specified load conditions.

I_{OL} - Low-Level Output Current : It is the current that flows from an output in the logical 0 state under specified load conditions.

Note : The current directions shown in the Fig. 6.5 may be opposite to those shown depending on the logic family.

6.3.2 Fan Out

In a digital system, we typically find many types of digital ICs interconnected to perform various functions. In these situations, the output of a logic gate may be connected to the inputs of several other similar gates. The maximum number of inputs of several gates that can be driven by the output of a logic gate is decided by the parameter called fan-out. In general, the fan-out is defined as the maximum number of inputs of the same IC family that the gate can drive maintaining its output levels within the specified limits. For example, a logic gate with fan-out 10 can drive maximum 10 logic inputs from the same family.

6.3.3 Noise Margin

In digital circuits, the binary 0 and 1 are represented by a pair of voltage levels. Each logic family has a different standard. The Table 6.1 shows the voltages used by several families.

Family	Logic 0	Logic 1
TTL	0 V	+5 V
ECL	-1.7 V	-0.9 V
CMOS	0 V	3 - 15 V

Table 6.1

These are the ideal voltage levels. But in practice, it is difficult to get these ideal voltages. Stray electric and magnetic field can induce voltages on the connecting wires

6.3.6 Speed Power Product (Figure of Merit)

In general, for any digital IC, it is desirable to have shorter propagation delays (higher speed) and lower values of power dissipation. There is usually a trade-off between switching speed and power dissipation in the design of a logic circuit i.e. speed is gained at the expense of increased power dissipation. Therefore, a common means for measuring and comparing the overall performance of an IC family is the **speed-power product (SPP)**. It is also called **Figure of Merit**. Actually, the speed-power product is computed as the product of propagation delay and average power dissipation, so the smaller the product, the better the overall performance. Notice that the product of propagation delay in seconds and power dissipation in watts (joules per second) has the units of energy - i.e. joules (J).

$$\therefore \text{Speed-power product (J)} = \text{Propagation delay (Seconds)} \times \text{Power dissipation (Joules/seconds)}$$

Ex. 6.2 : For a certain IC family, propagation delay is 10ns with an average power dissipation of 6 mW. What is its speed power product ?

Sol. : The speed-power product is given as

$$\begin{aligned} \text{SPP} &= \text{propagation delay} \times \text{average power dissipation} \\ &= 10 \text{ ns} \times 6 \text{ mW} \\ &= 60 \text{ pico joules (pJ)} \end{aligned}$$

In this section we have seen definition of various parameter/characteristics of logic families. In the next sections we are going to study various logic families.

6.4 TTL

Transistor-transistor logic, TTL, is named for its dependence on transistors alone to perform basic logic operations. The first version, which is now known as standard TTL, was developed in 1965 and is rarely used in today's systems. Through the years, the basic design has been modified to improve its performance in several respects, and as a consequence, a number of subfamilies have evolved. In this section we are going to study the basic transistor configurations in TTL and its subfamily circuits along with their characteristics.

Fig. 6.9 shows a transistor configuration for the standard TTL circuit. In particular, this configuration is a two input NAND gate.

6.4.1 Multiple-Emitter Transistor

Looking at the Fig. 6.9 (a) it can be noticed that Q_1 is an NPN transistor having two emitters, one for each input to the gate. Although this circuit looks complex, we can simplify its analysis by using the diode equivalent of the multiple-emitter transistor Q_1 , as

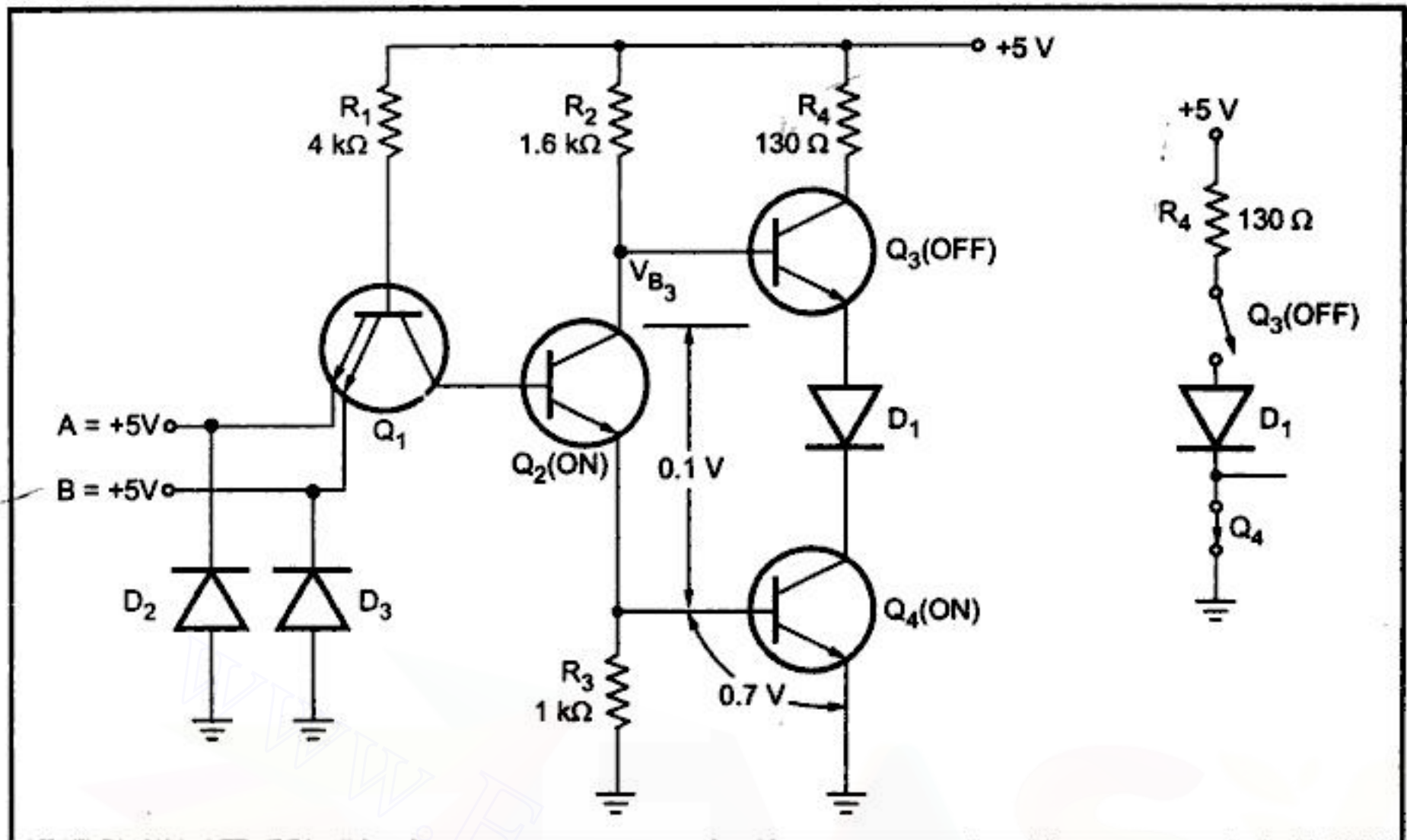


Fig. 6.11 (b) Static analysis when output is low

$$\begin{aligned} V_{B_3} &= V_{BE}(Q_4) + V_{CE(sat)}(Q_2) \approx 0.7 \text{ V} + 0.1 \text{ V} \\ &= 0.8 \text{ V} \end{aligned}$$

Now we can easily understand the purpose of diode D_1 . It does not allow base-emitter junction of Q_3 to be forward-biased and thus ensures that Q_3 remains off when Q_4 is on.

Ex. 6.3 : A two input NAND gate has $V_{CC} = +5 \text{ V}$ and $1 \text{ k}\Omega$ load connected to its output. Calculate the output voltage

(a) when both input are LOW

(b) when both input are HIGH

Sol. : a) When both inputs of NAND gate are LOW, the output is HIGH and it is given as

$$\begin{aligned} V_{OH} &= V_{CC} - V_{CE(sat)} - V_D - I_L \times (130\Omega) \\ &= 5 - 0.1 - 0.7 - I_L (130\Omega) \\ &= 4.2 \text{ V} - I_L (130\Omega) \end{aligned} \quad \dots(1)$$

where the load current is

$$I_L = \frac{V_{OH}}{R_L}$$

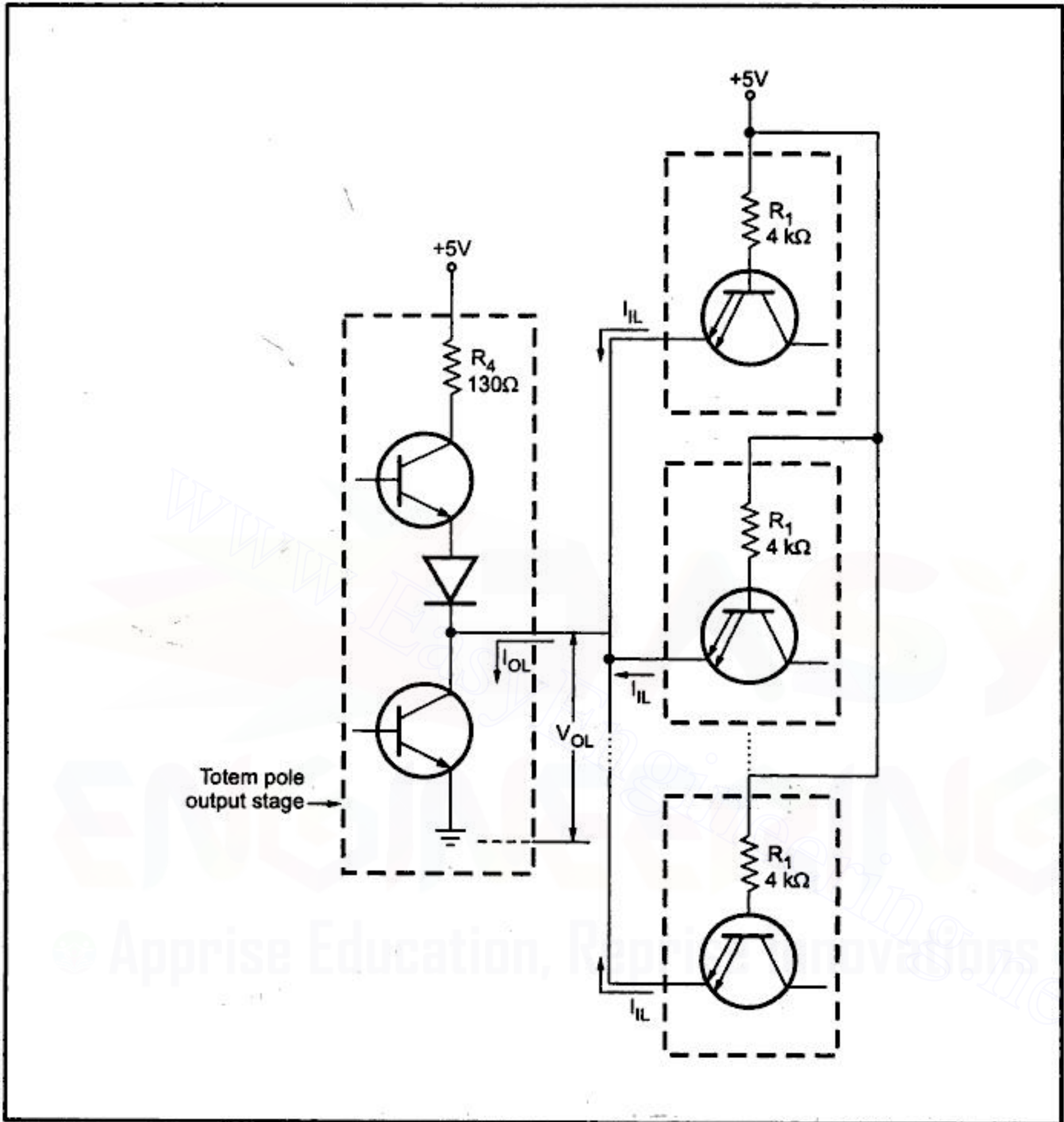
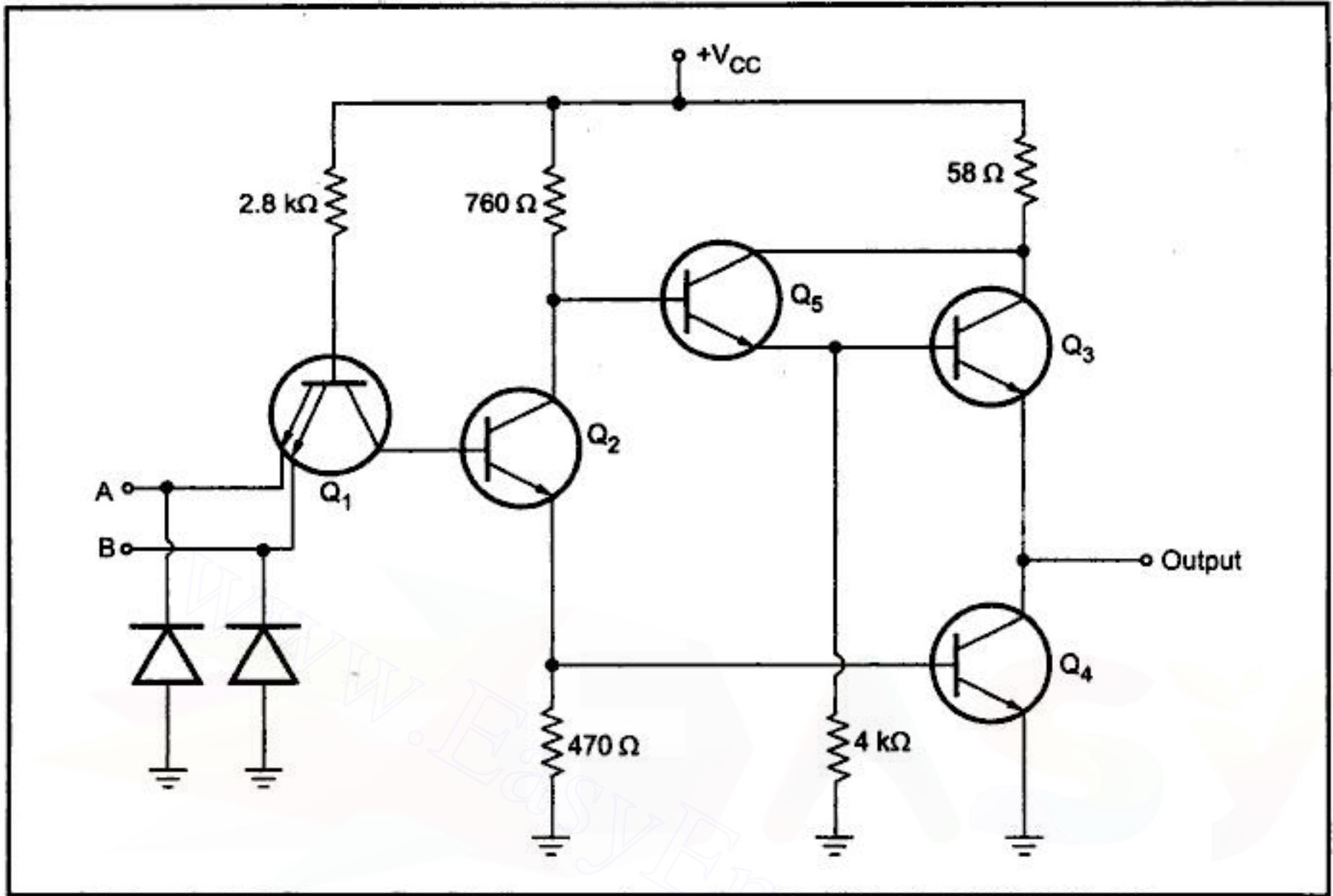


Fig. 6.13 (b) When output is low

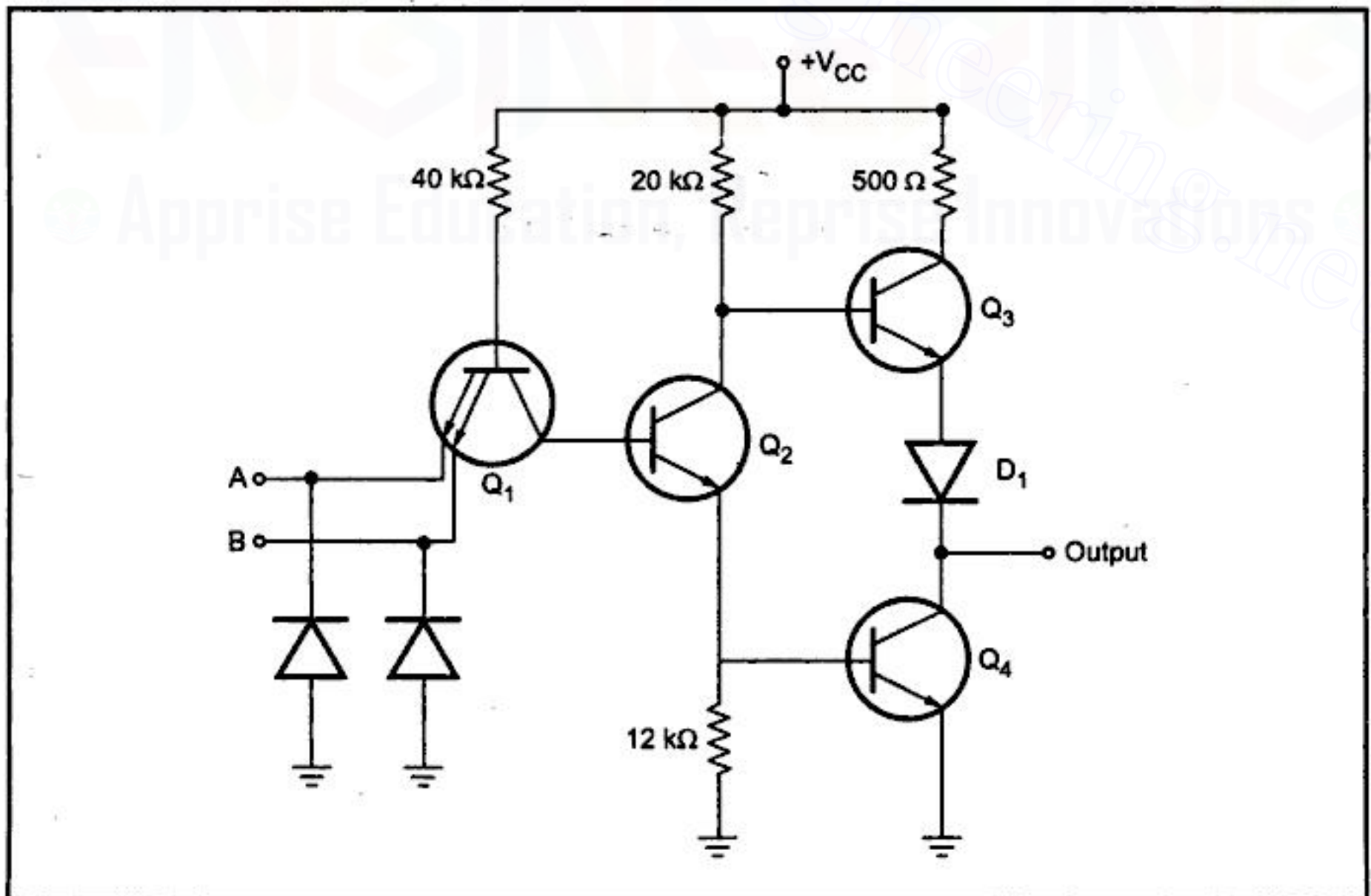
For standard TTL, $I_{OH(max)} = -400 \mu A$ and $I_{IH(max)} = 40 \mu A$.

Therefore,

$$\begin{aligned} \text{fanout} &= \frac{|-400 \mu A|}{40 \mu A} \\ &= 10 \end{aligned}$$



(a) 2-input high speed NAND gate



(b) 2-input low power NAND gate

Fig. 6.15 74H00 and 74L00 NAND gates

The 74ALS series offers an improvement over the 74LS series in both speed and power dissipation. The 74ALS series has the lowest speed-power product of all the TTL series.

6.5.5 Fast (F) TTL

This is the latest TTL series. It uses a new integrated-circuit fabrication technique to reduce inter-device capacitances. This results low propagation delays. A typical fast TTL NAND gate has an average propagation delay of 3 ns and a power consumption of 6 mW.

6.5.6 Comparison of TTL Series Characteristics

Table 6.5 shows the typical values of characteristics of all the TTL series. All of the performance ratings, except for maximum clock rate, are for a NAND gate in each series.

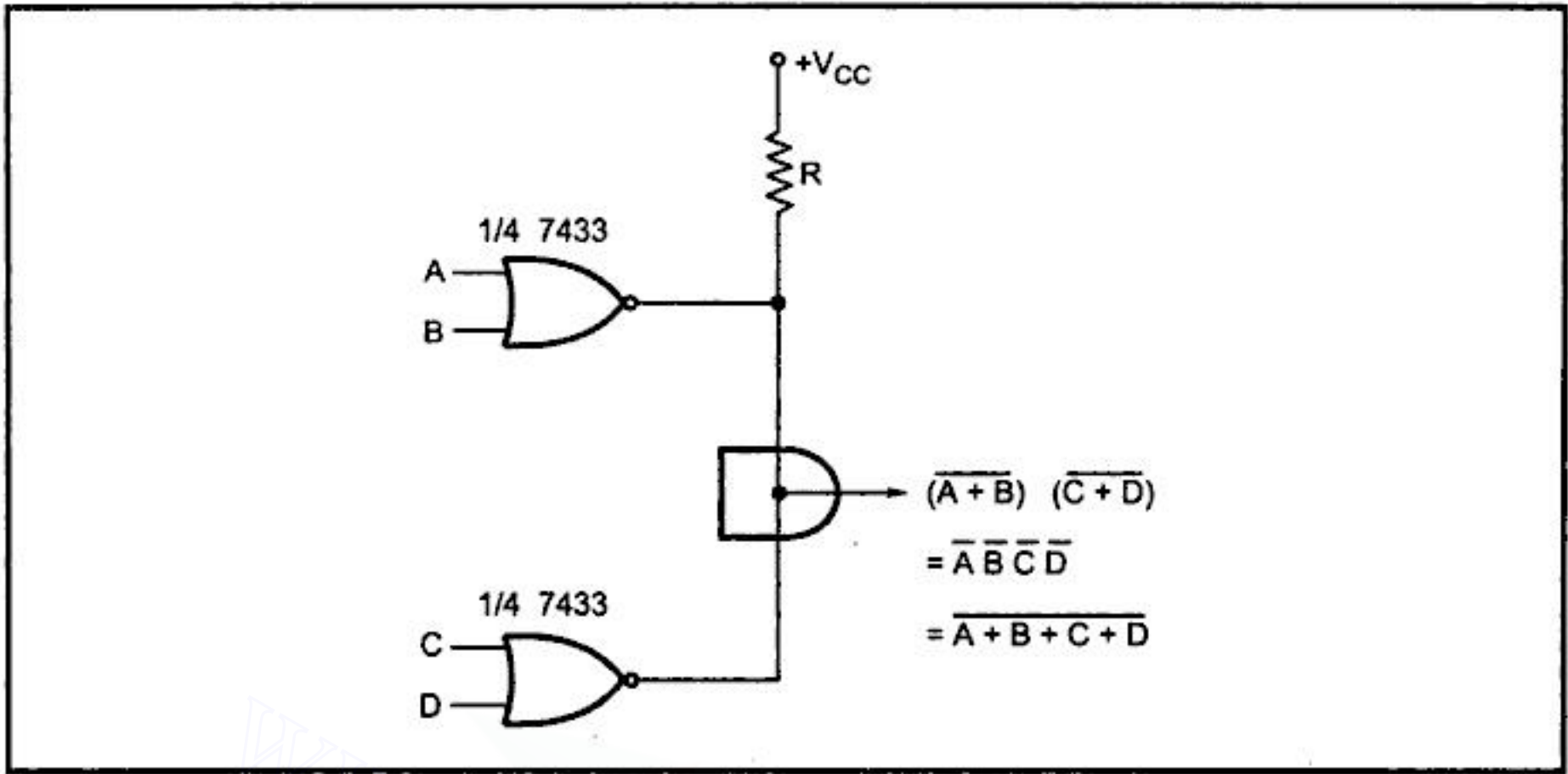
Parameter	74	74S	74LS	74AS	74ALS	74F
Propagation delay (ns)	9	3	9.5	1.7	4	3
Power dissipation (mW)	10	20	2	8	1.2	6
Speed-power product (pJ)	90	60	19	13.6	4.8	18
Max. clock rate (MHz)	35	125	45	200	70	100
Fan-out	10	10 (Low level leads)	20	40	20	33
		20 (high level leads)				
Voltage parameters:						
$V_{OH(min)}$ (V)	2.4	2.7	2.7	2.5	2.5	2.5
$V_{OL(max)}$ (V)	0.4	0.4	0.5	0.5	0.4	0.5
$V_{IH(min)}$ (V)	2.0	2.0	2.0	2.0	2.0	2.0
$V_{IL(max)}$ (V)	0.8	0.8	0.8	0.8	0.8	0.8

Table 6.5

6.6 Open Collector Outputs

Until now we have seen totem pole output. One problem with totem pole output is that two outputs cannot be tied together. See the Fig. 6.19, where the totem pole outputs of two separate gates are connected together at point X. Suppose that the output of gate A is high (Q_{3A} ON and Q_{4A} OFF) and the output of gate B is low (Q_{3B} OFF and Q_{4B} ON). In this situation transistor Q_{4B} acts as a load for Q_{3A} . Since Q_{4B} is a low resistance load, it draws high current around 55 mA. This current might not damage Q_{3A} or Q_{4B} immediately, but over a period of time can cause overheating and deterioration in performance and eventual device failure.

Some TTL devices provide another type of output called open collector output. The outputs of two different gates with open collector output can be tied together. Fig. 6.20 shows a 2-input NAND gate with an open-collector output eliminates the pull-up transistor Q_3 , D_1 and R_4 . The output is taken from the open collector terminal of transistor Q_4 .



(c) Wire-ANDed output of NOR gates

Fig. 6.23 The wired-AND connections

6.8 Comparison Between TOTEM Pole and Open Collector Output

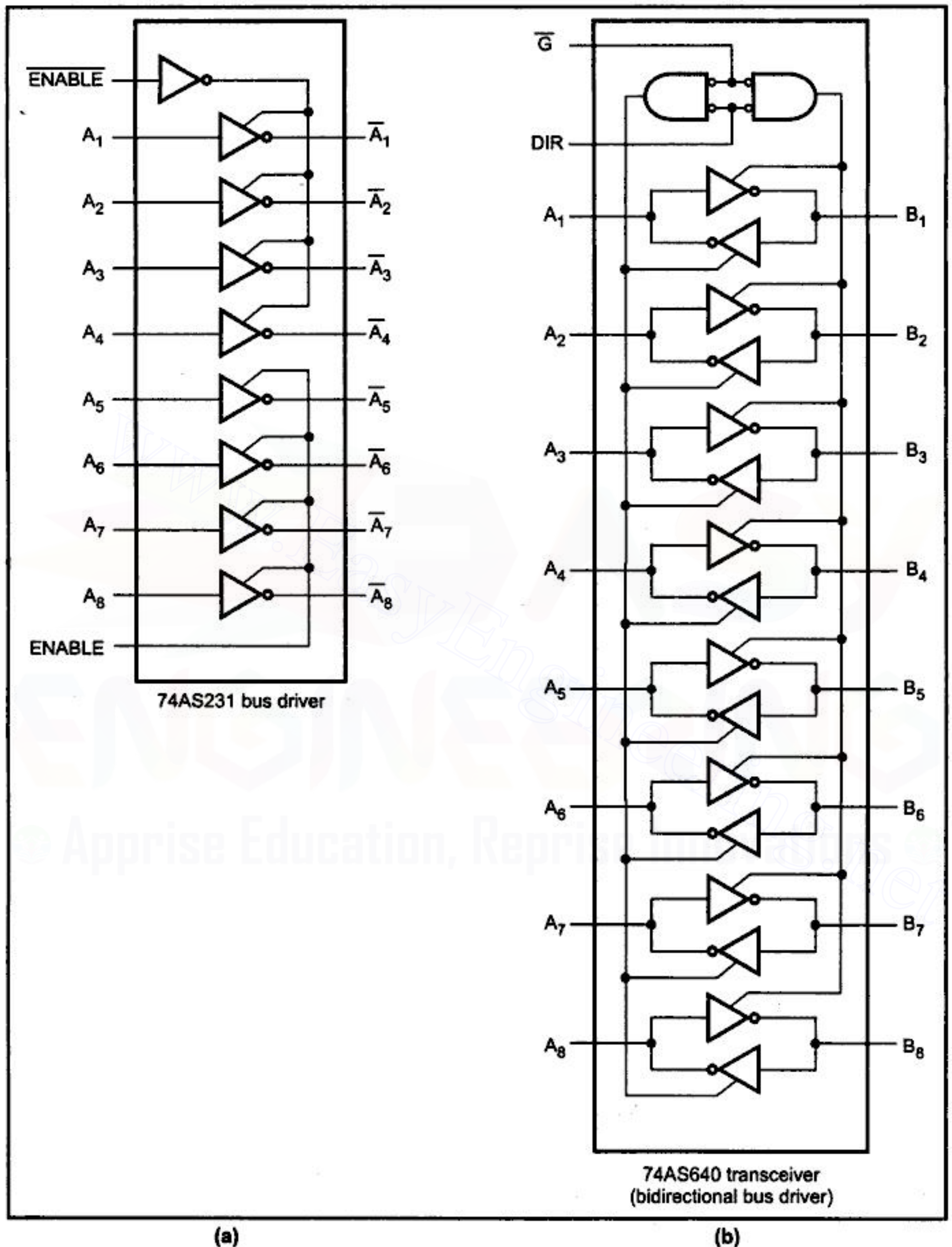
Table 6.6 summarizes the difference between totem pole and open collector outputs.

Totem Pole	Open Collector
1. Output stage consists of pullup transistor (Q_1), diode resistor and pull down transistor (Q_4)	1. Output stage consists of only pull down transistor
2. External pull-up resistor is not required	2. External pull-up resistor is required for proper operation of gate
3. Output of two gates cannot be tied together	3. Output of two gates can be tied together using wired AND technique
4. Operating speed is high	4. Operating speed is low

Table 6.6 Comparison of totem pole and open collector output

6.9 Tri-State Logic and Bus Drivers

The tristate configuration is a third type of TTL output configuration. It utilizes the high-speed operation of the totem-pole arrangement while permitting outputs to be wired-ANDed (connected together). It is called tristate TTL because it allows three possible output stages : HIGH, LOW and high-impedance. We know that transistor Q_3 is ON when output is HIGH and Q_4 is ON when output is LOW. In the high impedance state both transistors, transistors Q_3 and Q_4 in the totem pole arrangement are turned OFF. As a result, the output is open or floating, it is neither LOW nor HIGH.



(a)

(b)

Fig. 6.28 A typical tristate bus driver and transceiver

Table 6.9 summarizes the operation of NMOS NOR gate.

A	B	Q_2	Q_3	$V_o = \overline{A+B}$
0	0	OFF	OFF	1
0	1	OFF	ON	0
1	0	ON	OFF	0
1	1	ON	ON	0

Table 6.9

Characteristics of NMOS

Operating speed : Low operating speed with propagation delay time around 50ns. This is because it has high output resistance, very high input resistance and reasonably high input capacitance.

Noise Margin : Typically 1.5 V

Fan out : Typically 30

Power drain : Less, around 0.1 mW per gate.

6.12 CMOS

CMOS circuits contain both NMOS and PMOS devices to speed the switching of capacitive loads. It consumes low power and can be operated at high voltages, resulting in improved noise immunity.

6.12.1 CMOS Inverter

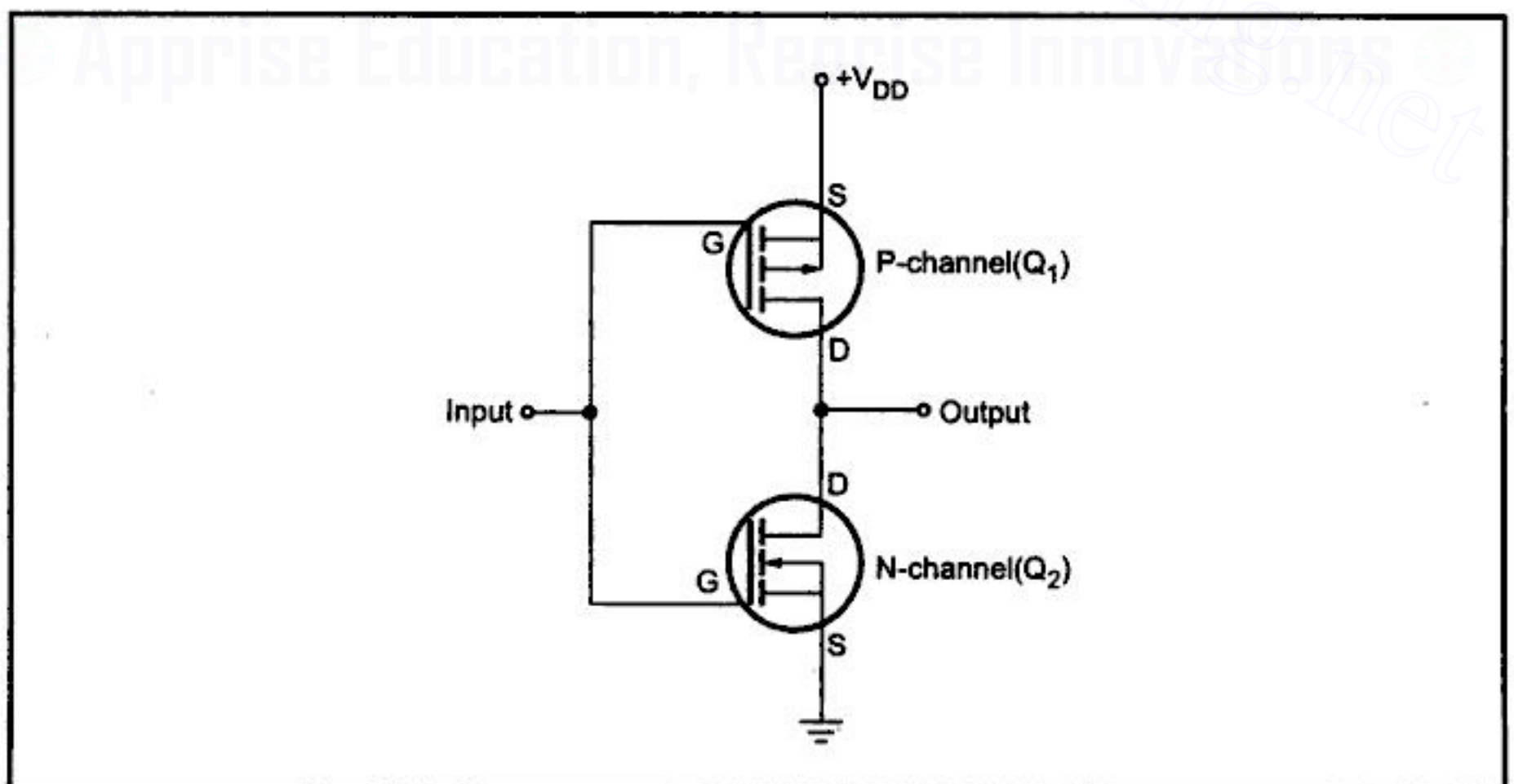


Fig. 6.32 CMOS Inverter circuit

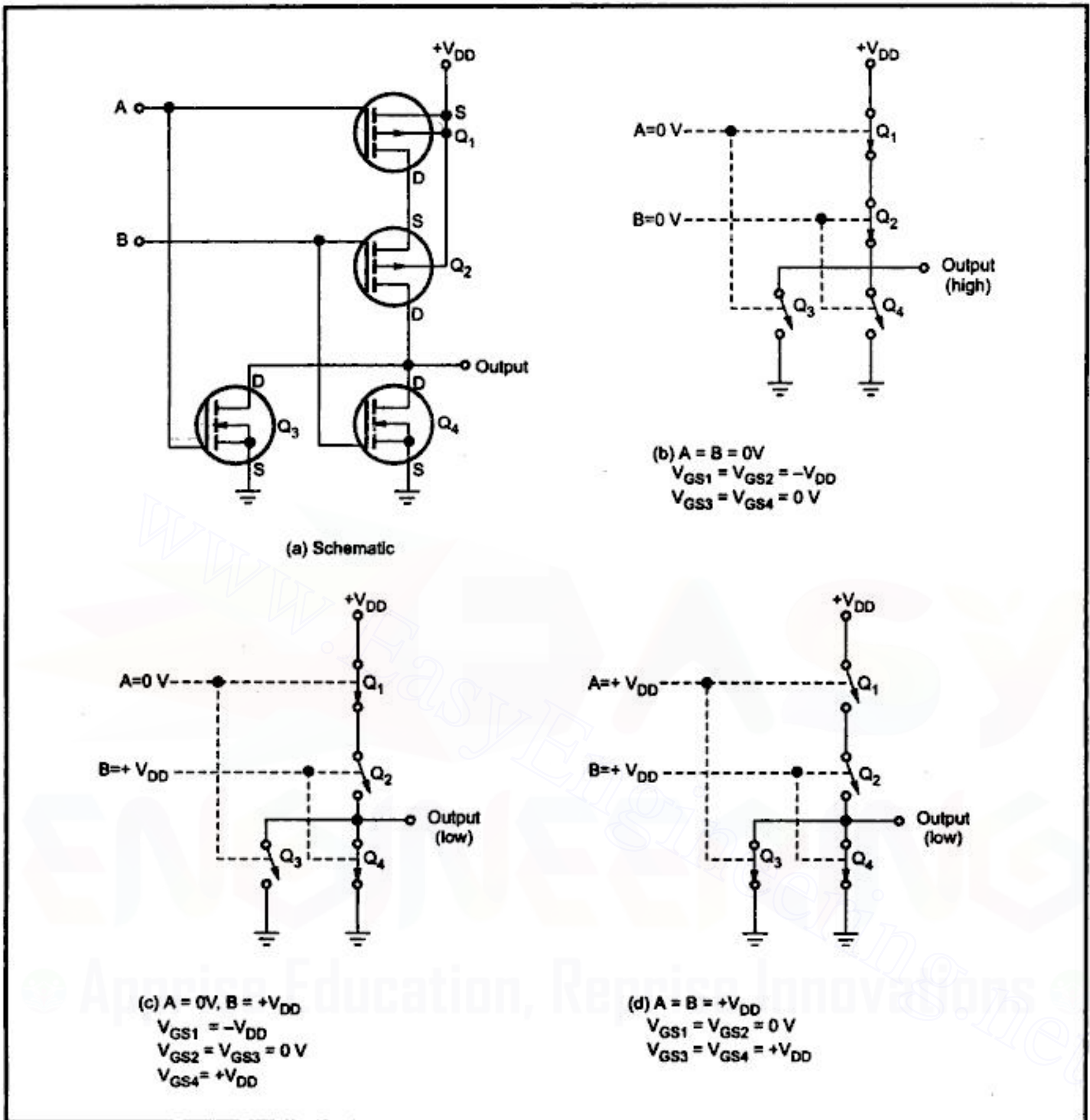


Fig. 6.35 CMOS NOR gate

6.12.4 Characteristics of CMOS

Operating Speed : Slower than TTL series. Approximately 25 to 100 ns depending on the subfamily of CMOS. It also depends on the power supply voltage.

Voltage levels and Noise Margins : The voltage levels for CMOS varies according to their subfamilies. These are listed in Table 6.13. Noise margins in table are calculated as follows.

$$V_{NH} = V_{OH(\min)} - V_{IH(\min)}$$

$$V_{NL} = V_{IL(\max)} - V_{OL(\max)}$$

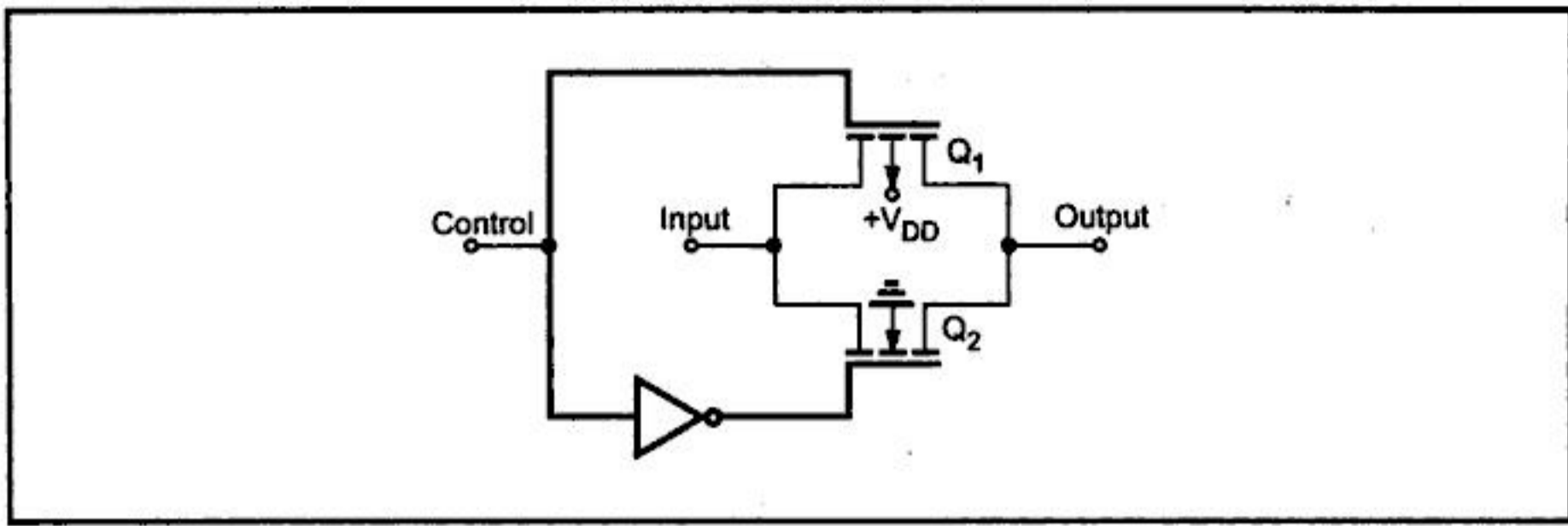


Fig. 6.38 (a) Schematic

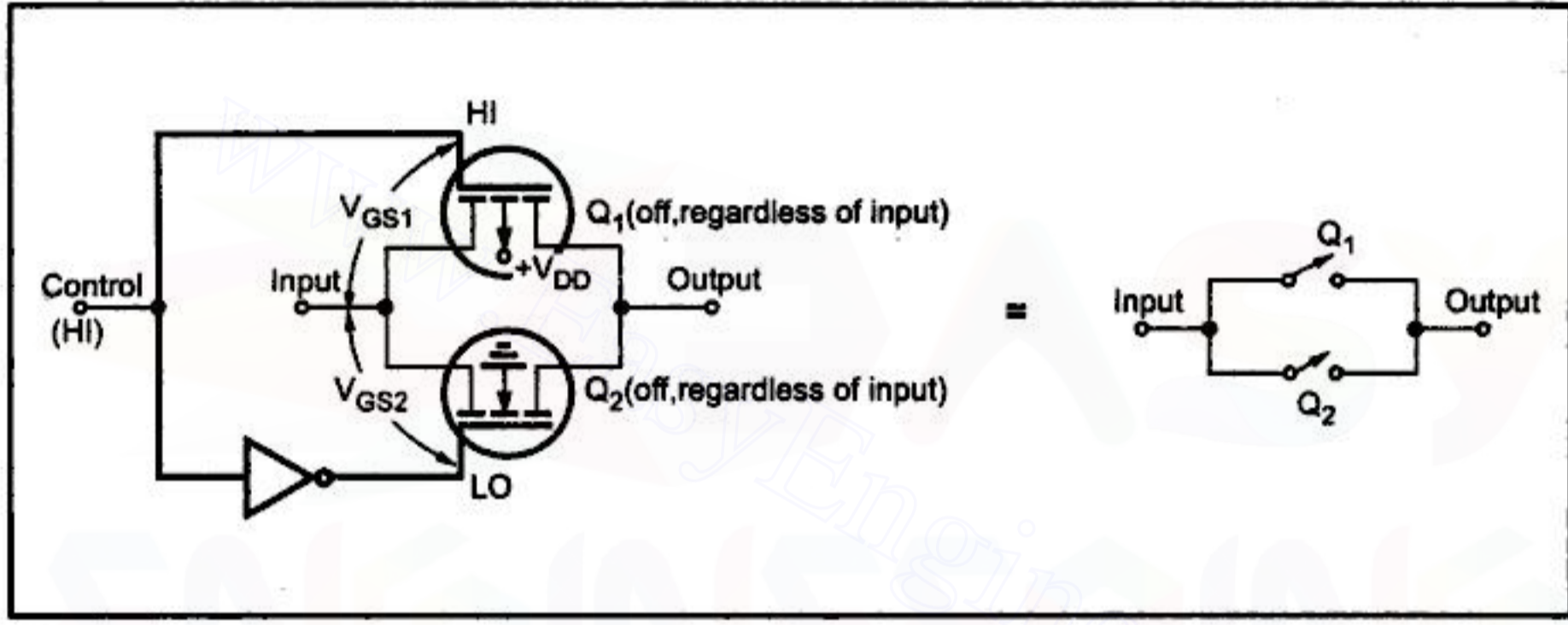


Fig. 6.38 (b) CONTROL high

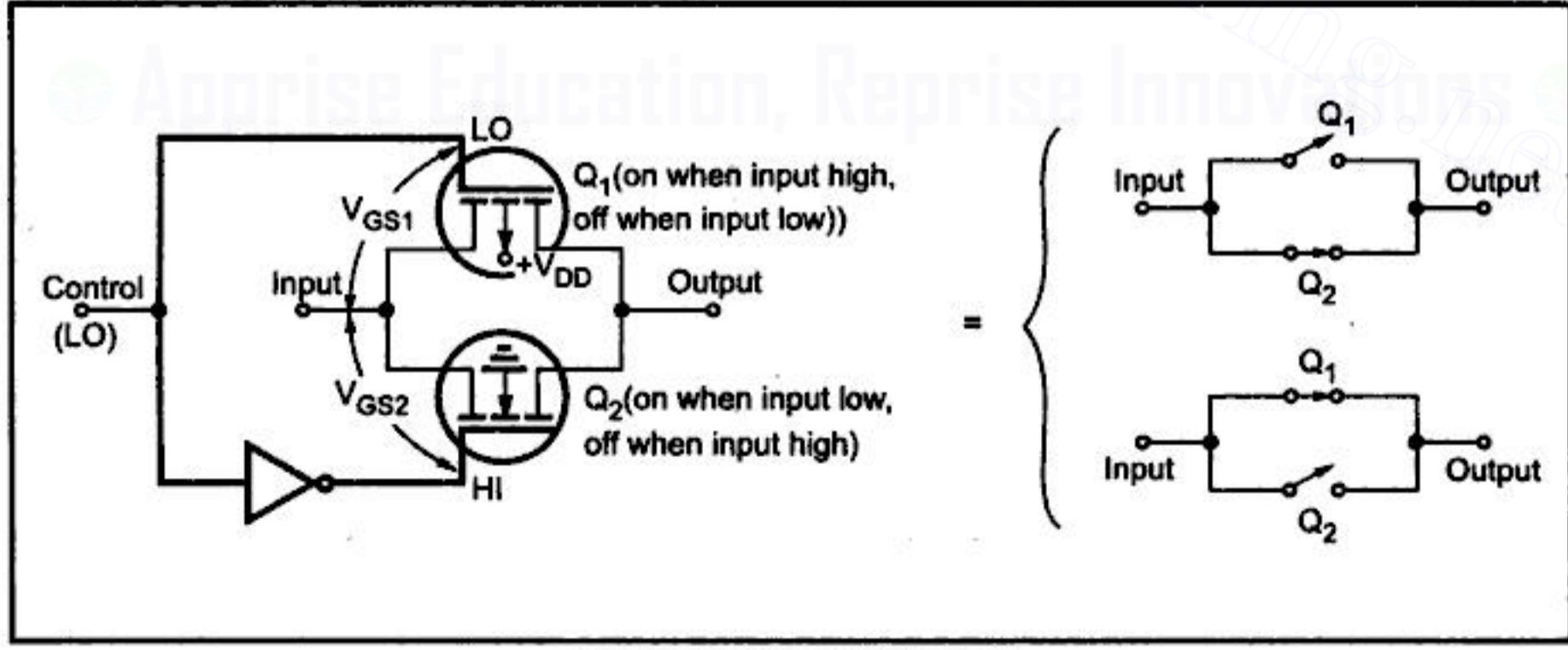


Fig. 6.38 (c) CONTROL low

Parameter		Temp Range	V _{DD} (V _{DC})	Conditions	Limits						Units	
					T _{LOW} *		+ 25°C		T _{HIGH} *			
					Min	Max	Min	Max	Min	Max		
V _{OL}	Low-Level Output Voltage	All	5	V _{IN} = V _{ss} or V _{DD} I _O < 1μA		0.05		0.05		0.05	V _{dc}	
			10		0.05		0.05		0.05			
			15		0.05		0.05		0.05			
V _{OH}	High-Level Output Voltage	All	5	V _{IN} = V _{ss} or V _{DD} I _O < 1μA	4.95		4.95		4.95		V _{dc}	
			10		9.95		9.95		9.95			
			15		14.95		14.95		14.95			
V _{IL}	Input Low Voltage# B Types	All	5	V _O = 0.5 V or 4.5 V		1.5					V _{dc}	
			10	V _O = 1.0 V or 9.0 V		3.0						
			15	V _O = 1.5 V or 13.5 V		4.0						
V _{IL}	Input Low Voltage# UB Types	All	5	V _O = 0.5 V or 4.5 V		1.0		1.0		1.0	V _{dc}	
			10	V _O = 1.0 V or 9.0 V		2.0		2.0		2.0		
			15	V _O = 1.5 V or 13.5 V		2.5		2.5		2.5		
V _{IH}	Input High Voltage# B Types	All	5	V _O = 0.5 V or 4.5 V	3.5		3.5		3.5		V _{dc}	
			10	V _O = 1.0 V or 9.0 V	7.0		7.0		7.0			
			15	V _O = 1.5 V or 13.5 V	11.0		11.0		11.0			
V _{IH}	Input High Voltage# UB Types	All	5	V _O = 0.5 V or 4.5 V	4.0		4.0		4.0		V _{dc}	
			10	V _O = 1.0 V or 9.0 V	8.0		8.0		8.0			
			15	V _O = 1.5 V or 13.5 V	12.5	12.5	12.5					
I _{OL}	Output Low (Sink Current)	Mil		V _O = 0.4 V							mA _{dc}	
			5	V _{IN} = 0 or 5V V _O = 0.5 V	0.64		0.51		0.36			
			10	V _{IN} = 0 or 10 V V _O = 1.5 V	1.6		1.3		0.9			
		15	V _{IN} = 0 or 15V	4.2		3.4		2.4				
		Com	5	V _O = 0.4 V V _{IN} = 0 or 5 V V _O = 0.5 V	0.52		0.44		0.36			mA _{dc}
			10	V _{IN} = 0 or 10V V _O = 1.5 V	1.3		1.1		0.9			
15	V _{IN} = 0 or 15V		3.6		3.0		2.4					

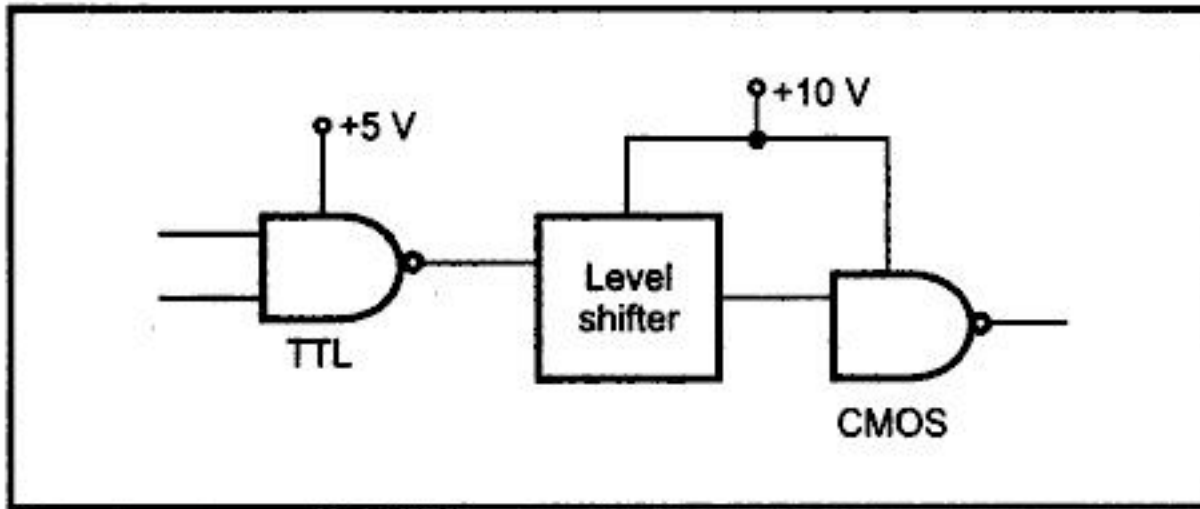


Fig. 6.43 Level shifter used as interface circuit

2. The second alternative is to use level translator circuit, such as the 40104. This is a CMOS chip that is designed to take a low-voltage input (e.g., from TTL) and translate it to high voltage output for CMOS. Fig. 6.43 shows the circuit arrangement.

6.13.2 CMOS Driving TTL

Before we consider the problem of interfacing CMOS outputs to TTL inputs, it will be helpful to review the CMOS output and TTL input characteristics for the two logic states.

For CMOS (4000B)	For TTL
$V_{OH(min)} : 4.95 \text{ V}$	$V_{IH(min)} : 2.0 \text{ V}$
$V_{OL(max)} : 0.05 \text{ V}$	$V_{IL(max)} : 0.8 \text{ V}$
$I_{OH(max)} : 0.4 \text{ mA}$	$I_{IH(max)} : 40 \mu\text{A}$
$I_{OL(max)} : 0.4 \text{ mA}$	$I_{IL(max)} : 1.6 \text{ mA}$

Table 6.18

CMOS Driving TTL in the HIGH state : Above voltage parameters show that CMOS outputs can easily supply enough voltage (V_{OH}) to satisfy the TTL input requirement in the HIGH state (V_{IH}). The parameters also show that CMOS outputs can supply more than enough current (I_{OH}) to meet the TTL input current requirements (I_{IH}). Thus no special consideration is required for CMOS driving TTL in the HIGH state.

CMOS Driving TTL in LOW state : The parameters in the Table 6.18 show that CMOS output voltage (V_{OL}) satisfies TTL input requirement in the LOW state (V_{IL}). However, the current requirements in the LOW state are not satisfied. The TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state (I_{OL}) is not sufficient to drive even one input of the TTL. In such situations some type of interface circuit is needed between the CMOS and TTL devices. Fig. 6.44 shows the possible interface circuit.

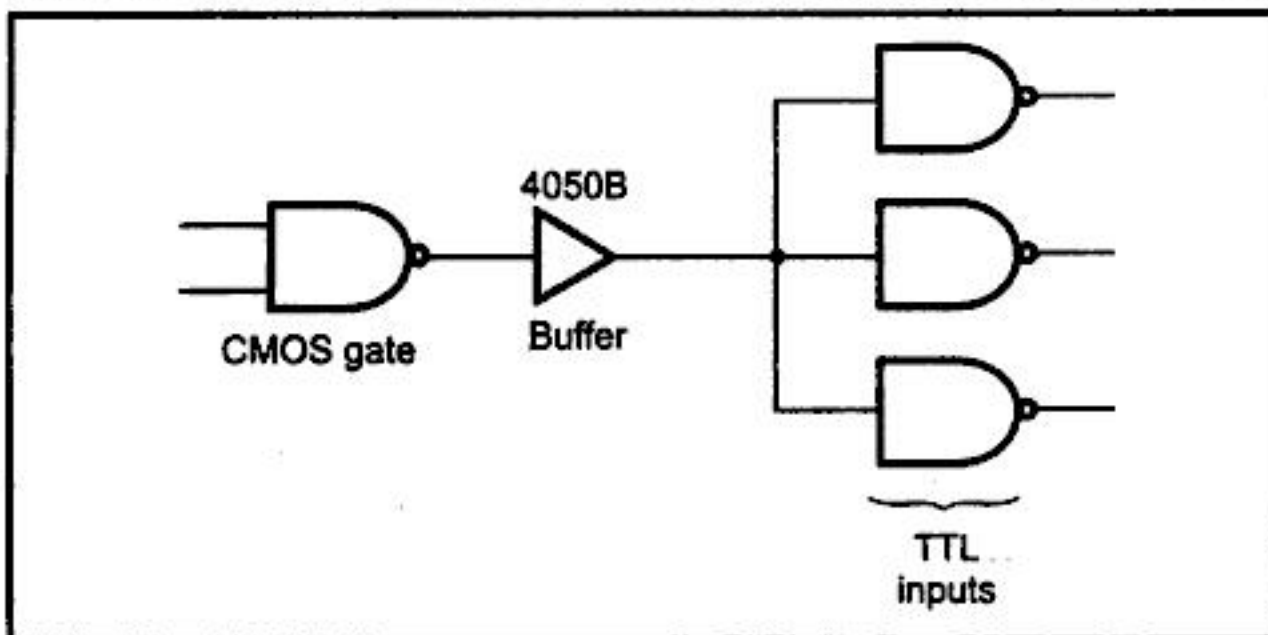


Fig. 6.44 CMOS driving TTL in LOW state using buffer

the LOW state (1.6 mA) and CMOS output current at LOW state (I_{OL}) is not sufficient to drive even one input of the TTL. In such situations some type of interface circuit is needed between the CMOS and TTL devices. Fig. 6.44 shows the possible interface circuit.

6.14.2 Dynamic MOS NOR and NAND Gates

Fig. 6.48 shows dynamic MOS NOR and NAND gates. These gates are similar to the NOR and NAND gates discussed previously. The only difference is that the load transistors are clocked by ϕ_1 and the outputs are clocked through a transmission gate by ϕ_2 , as in the dynamic inverter.

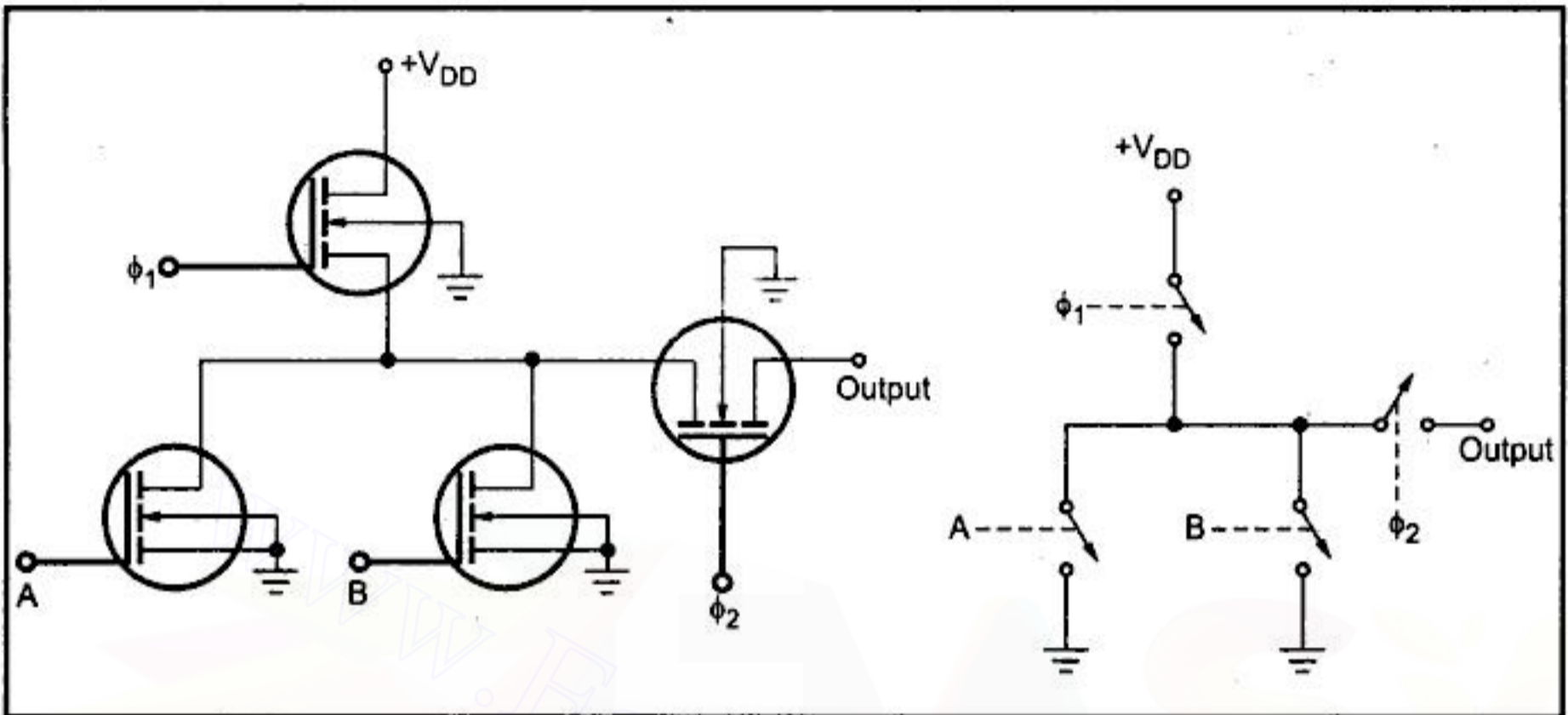


Fig. 6.48 (a) Dynamic NOR gate

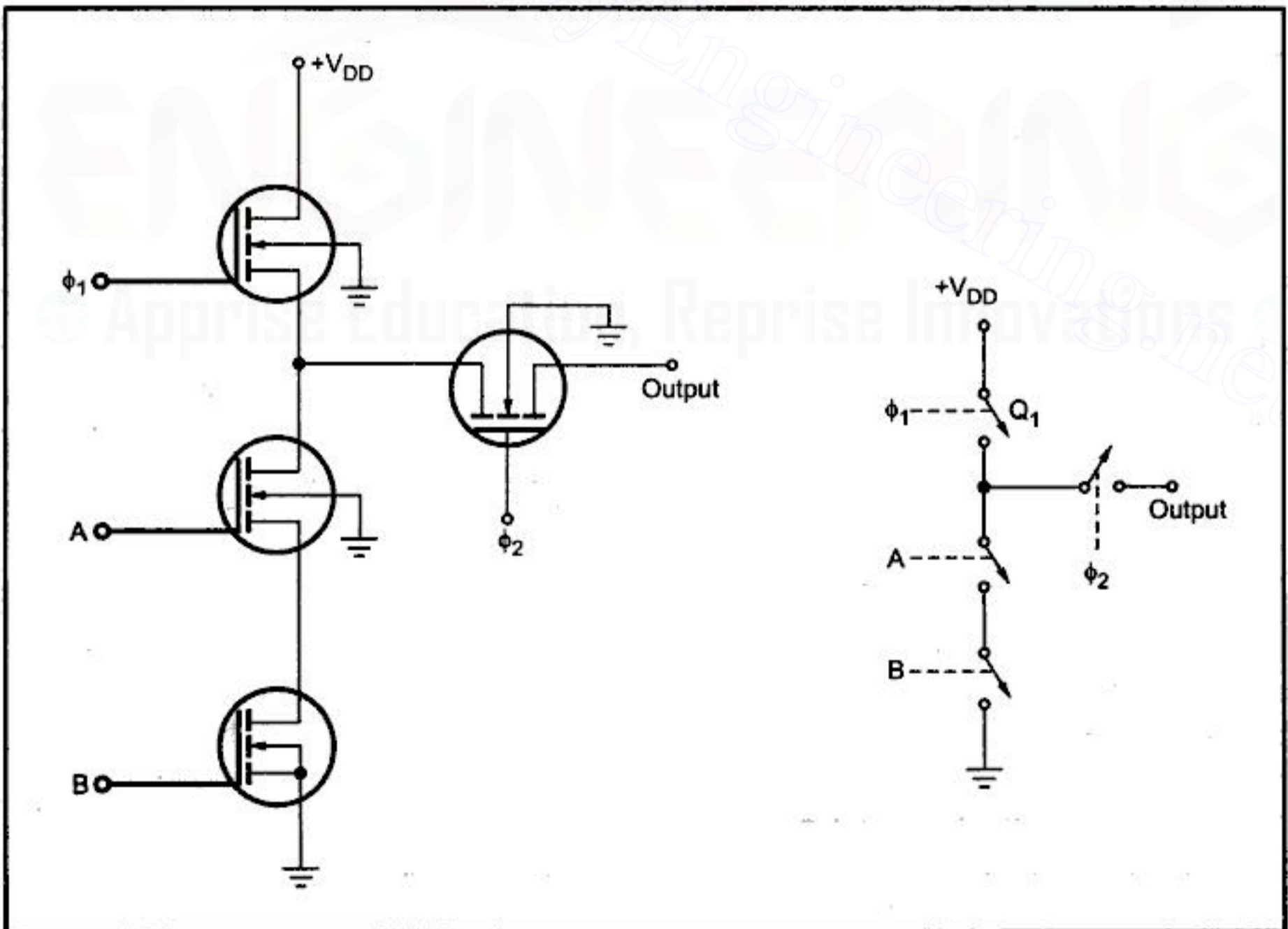


Fig. 6.48 (b) Dynamic NAND gate

Review Questions

1. What are the advantages of digital integrated circuits ?
2. Define following parameters

a) Current and voltage	b) Fan-out	c) Noise margin
d) Propagation delay	e) Power dissipation	f) Speed power product
3. Draw the circuit diagram and explain the operation of 2 input TTL NAND gate with Totem-pole output.
4. Write a note on multiple emitter transistor.
5. Describe the difference between current sinking and current sourcing.
6. State advantages and disadvantages of totem-pole output.
7. Describe the characteristics of TTL family.
8. Give the comparison of TTL series characteristics.
9. Explain the operation of low and medium power TTL NAND gate
10. Draw the circuit diagram and explain the operation of 2 input TTL NAND gate with open collector output.
11. Explain the wired-AND connection
12. Compare the totem-pole and open-collector outputs.
13. Draw and explain the circuit for tri-state TTL inverter.
14. Draw and explain the internal diagram of typical bus driver.
15. Draw and explain the basic CMOS inverter circuit.
16. Draw and explain the circuit of two input CMOS NAND gate.
17. Draw and explain the circuit of two input CMOS NOR gate.
18. Discuss the characteristics of CMOS family.
19. Give the comparison between TTL and CMOS families.
20. Explain with neat diagram interfacing of a TTL gate driving CMOS gates and vice-versa.
21. What do you mean by buffered and non buffered gates ? What is the advantage and disadvantage of buffered gates ?
22. Explain the working of transmission gate with the help of neat circuit diagram.
23. Explain open drain and high impedance outputs of CMOS.
24. What is dynamic MOS logic ? Explain its advantages and disadvantages.
25. Draw and explain the working of dynamic MOS inverter.
26. Give the characteristics of ECL family.
27. Draw the circuit diagram of a ECL NOR / OR gate. Explain the circuit operation. Show how it can be interfaced with TTL gate.
28. Discuss the advantages of
 - i) Complement output
 - ii) Connecting pull-down resistor at the input of ECL gate
29. Explain the interfacing of ECL gates with CMOS and TTL gates and vice-versa.

University Questions

1. Draw the circuit diagram of a 2-input TTL, NAND gate and explain its working.

(May-92, May-98, Dec-98)

7

Combinational Logic

7.1 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**. In combinational logic, 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. 7.1 shows the block diagram of a combinational circuit. As shown in Fig. 7.1, the combinational circuit accepts n -input binary variables and generates output variables depending on the logical combination of gates.

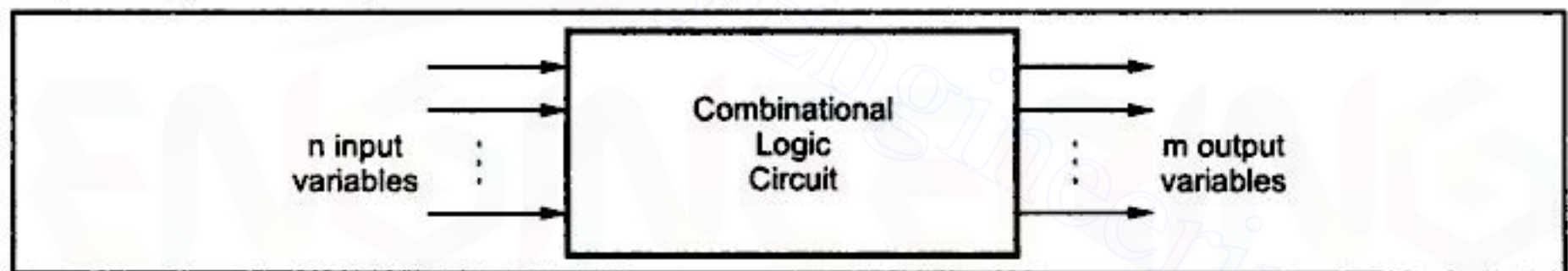


Fig. 7.1 Block diagram of a combinational circuit

7.2 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.

Logic Diagram

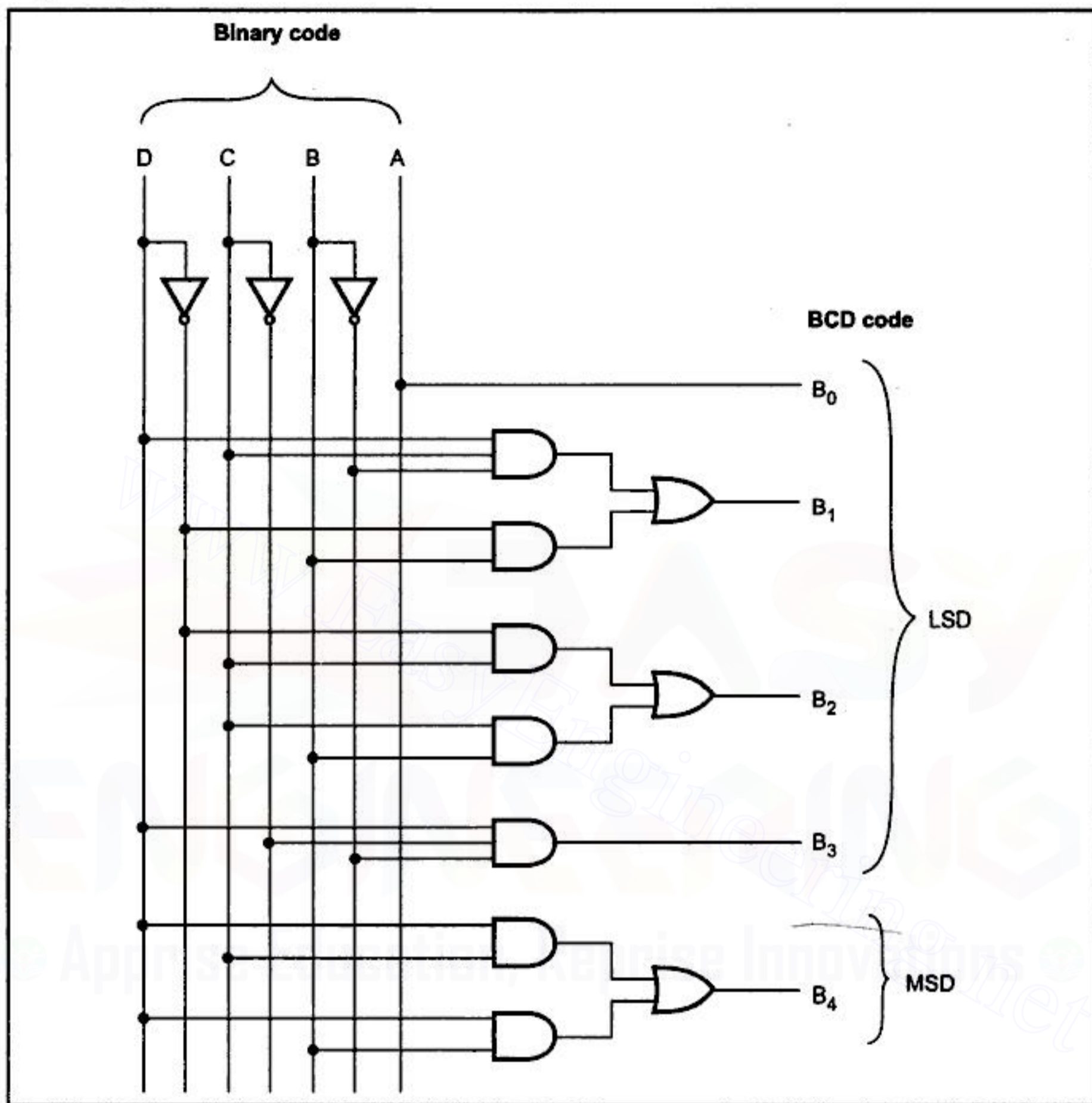


Fig. 7.5 Binary to BCD converter

7.3.2 BCD to Binary Converter

Let us see the truth table for BCD to binary converter.

B ₄	B ₃	B ₂	B ₁	B ₀	E	D	C	B	A
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0

7.3.3 BCD to Excess 3

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 7.4.

Decimal	B ₃	B ₂	B ₁	B ₀	E ₃	E ₂	E ₁	E ₀
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table 7.4

K-map simplification

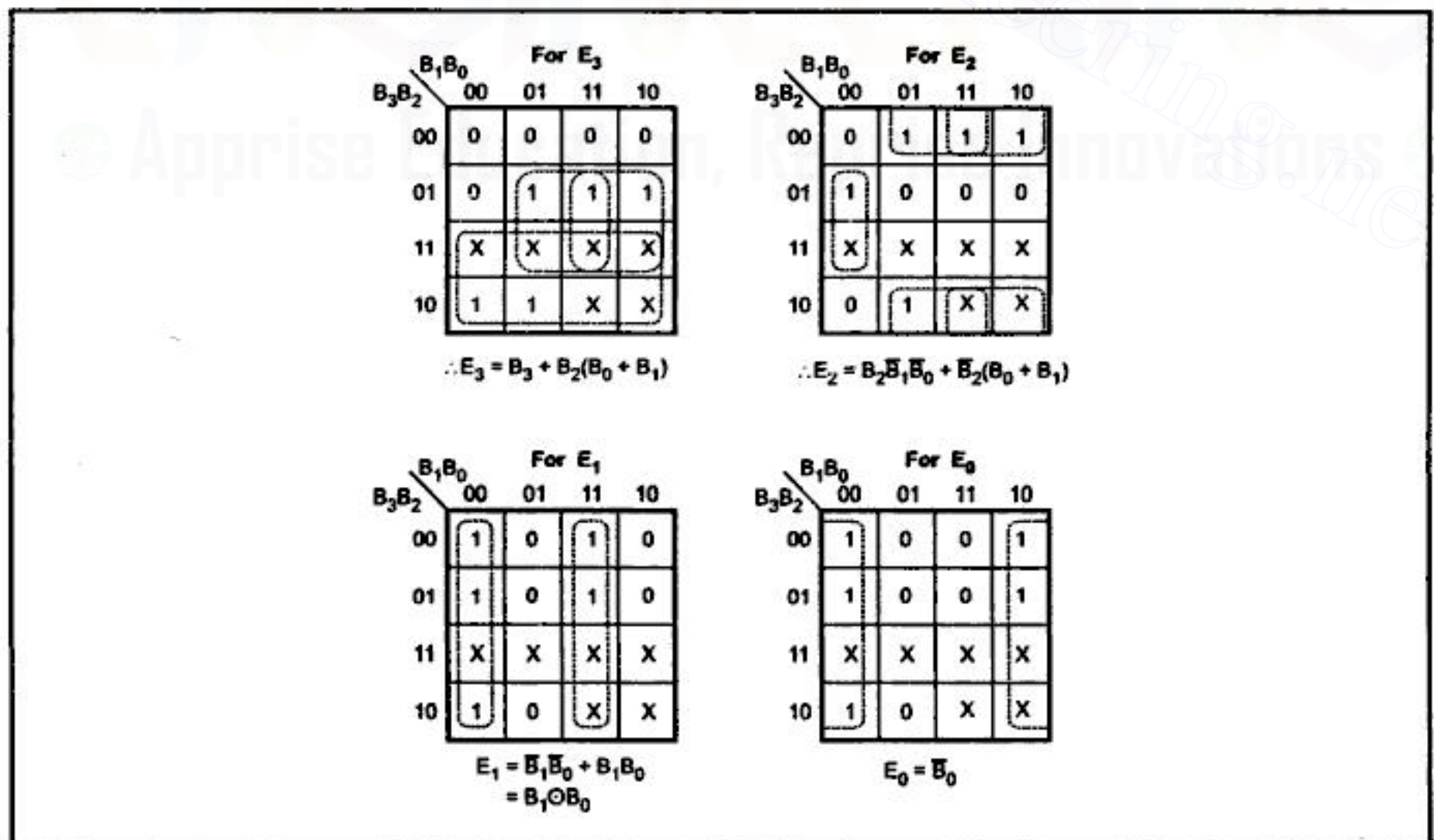


Fig. 7.8

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 7.6

K-map Simplification

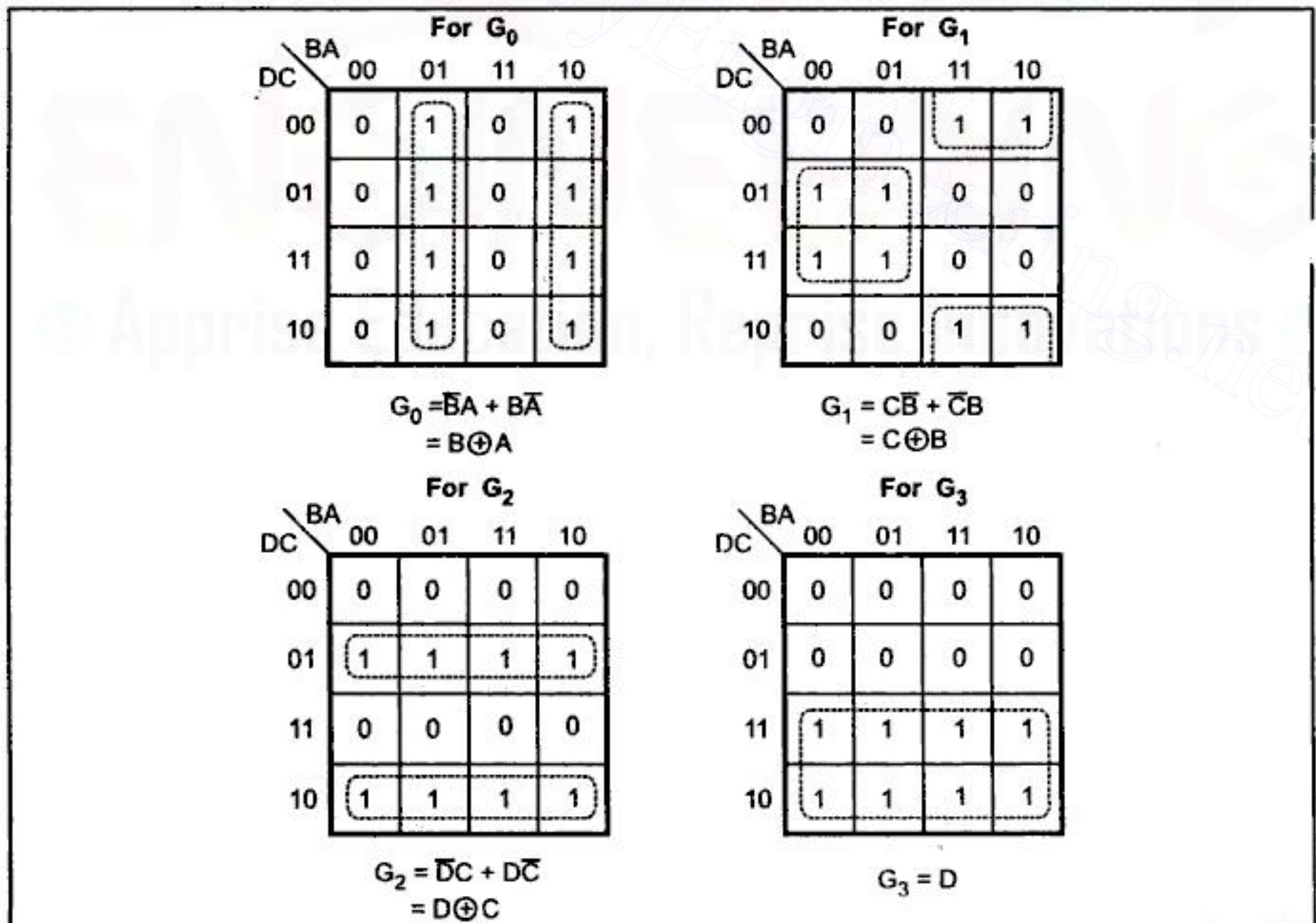


Fig. 7.12

K-map Simplification

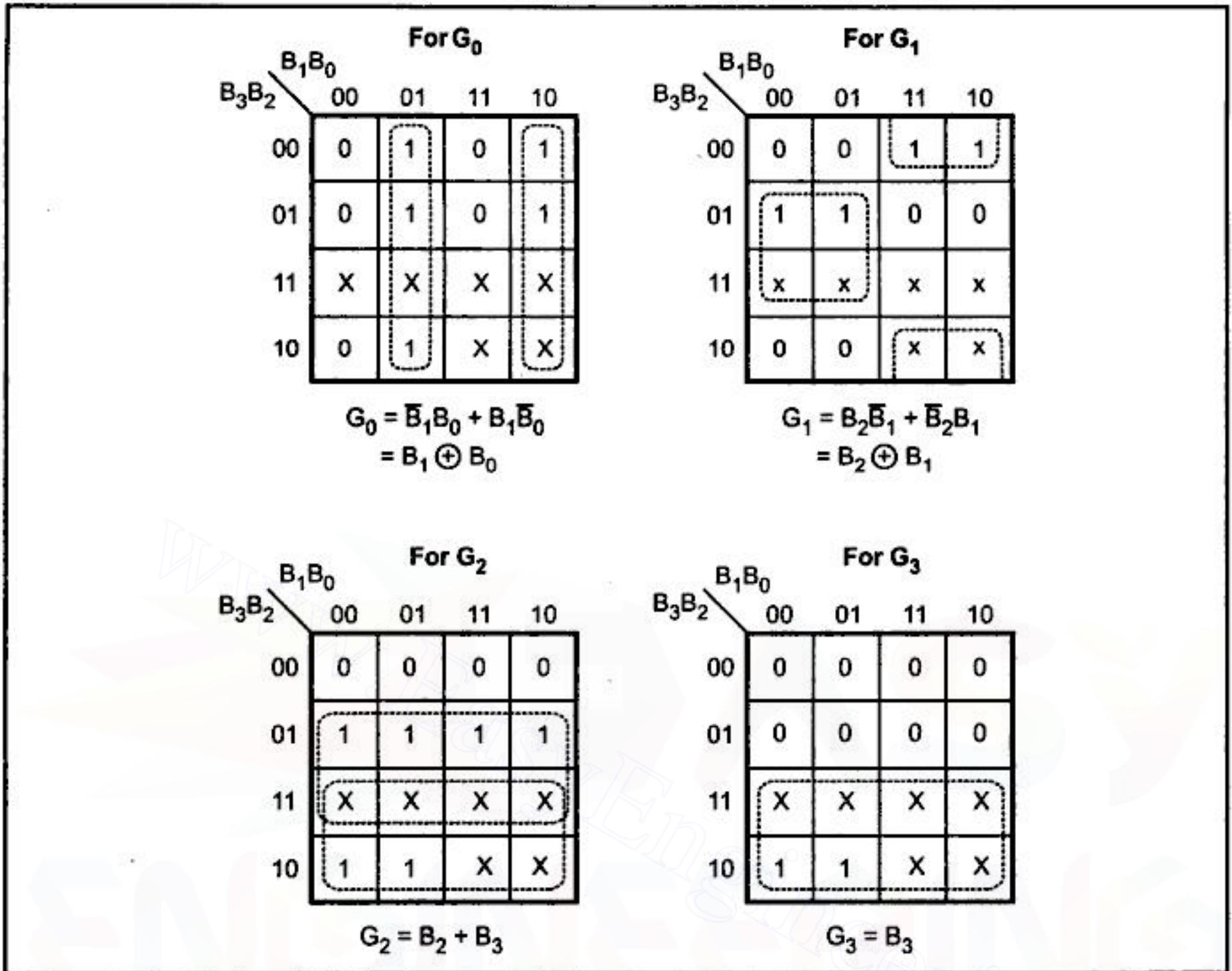


Fig. 7.17

Logic Diagram

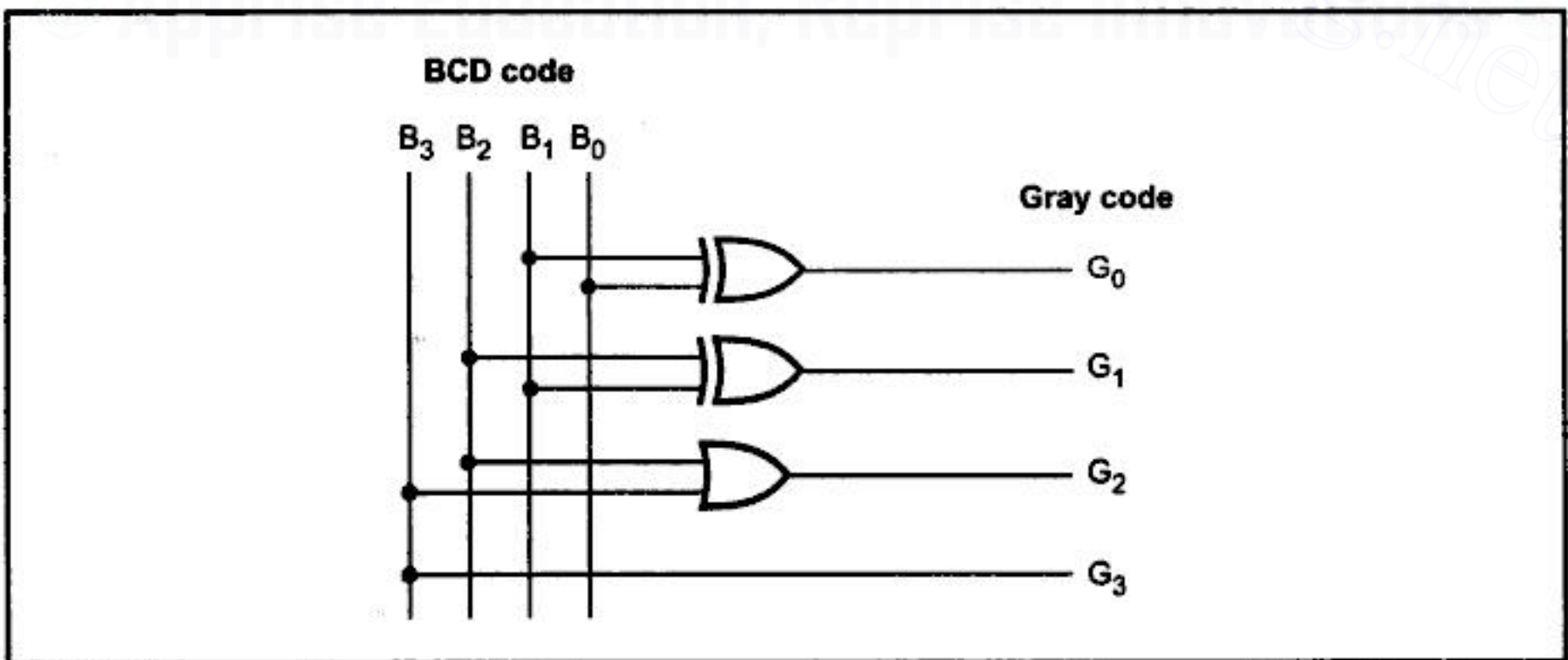


Fig. 7.18 BCD to gray code converter

$$\begin{aligned}
 &= \bar{A} \bar{B} C_{in} + A B C_{in} + A \bar{B} \bar{C}_{in} + \bar{A} B \bar{C}_{in} \\
 &= \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} \bar{C}_{in} + A B C_{in}
 \end{aligned}$$

and the carry output is

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

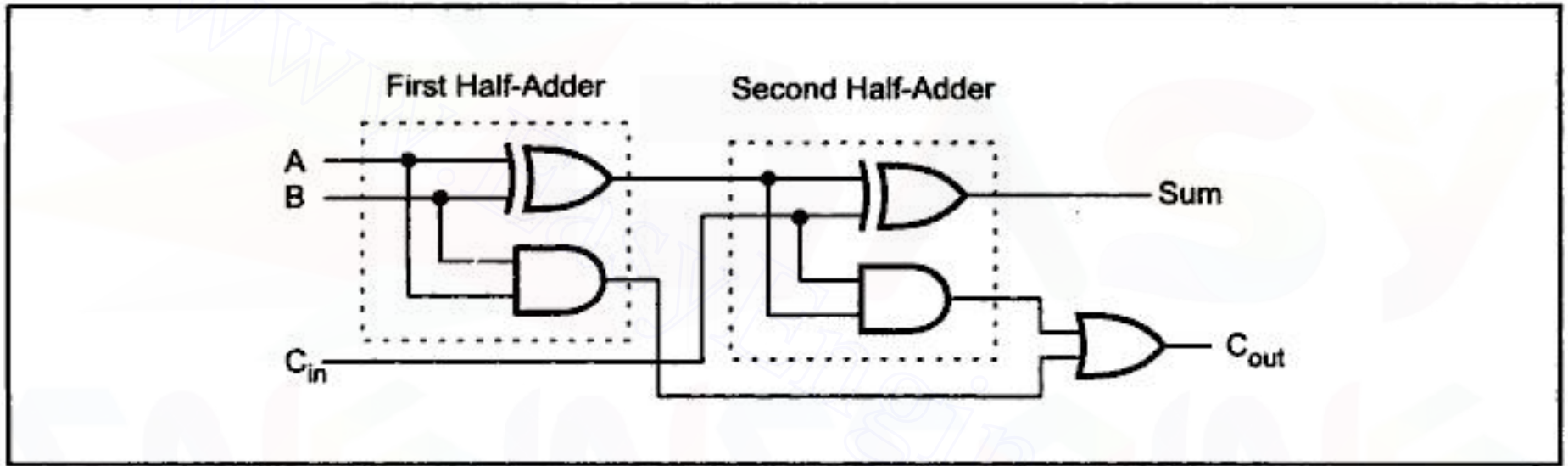


Fig. 7.26 Implementation of a full-adder with two half-adders and an OR gate

7.5 Subtractor

The subtraction consists of four possible elementary operations, namely,

- 0 - 0 = 0
- 0 - 1 = 1 with 1 borrow
- 1 - 0 = 1
- 1 - 1 = 0

In all operations, each subtrahend bit is subtracted from the minuend bit. In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed. Just as there are half and full-adders, there are half and full-subtractors.

7.5.1 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 7.11.

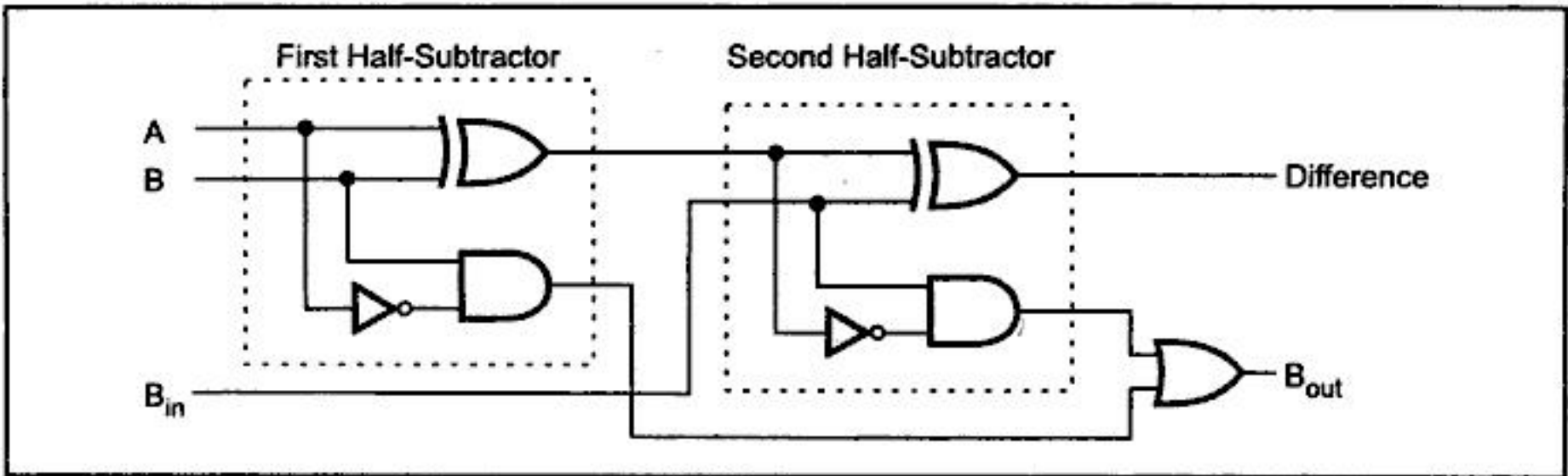


Fig. 7.32 Implementation of a full-subtractor with two half-subtractors and an OR gate

7.6 n-Bit Parallel Adder

We have seen, a single full-adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers with more than one bit, additional full-adders must be employed. A n-bit, parallel adder can be constructed using number of full adder circuits connected in parallel. Fig. 7.33 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.

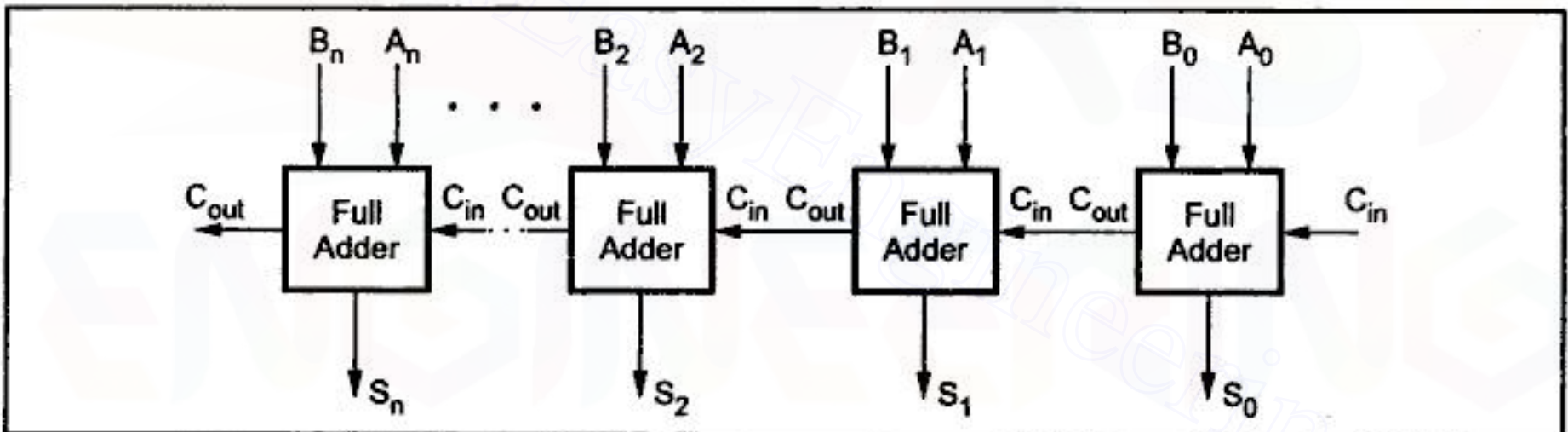


Fig. 7.33 Block diagram of n-bit parallel adder

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.

Ex. 7.2 : Design a 4-bit parallel adder using full adders .

Sol. : Fig. 7.34 shows the block diagram and Fig. 7.35 shows logic symbol for 4-bit parallel adder. Here, for least significant position, carry input of full-adder is made 0.

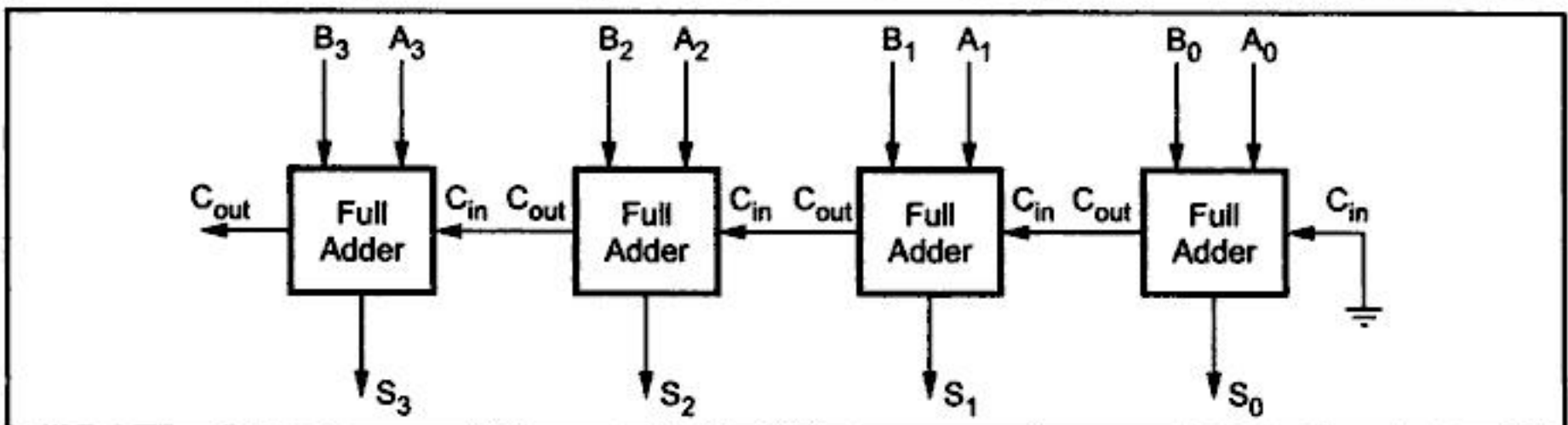


Fig. 7.34 Block diagram of 4-bit full adder

Using a look-a head carry generator we can easily construct a 4-bit parallel adder with a look-ahead carry scheme. Fig. 7.38 shows a 4-bit parallel adder with a look- ahead carry scheme. As shown in the Fig. 7.38, each sum output requires two exclusive-OR gates. The output of first exclusive-OR gate generates P_i , 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 is 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. 7.39 shows p in 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.

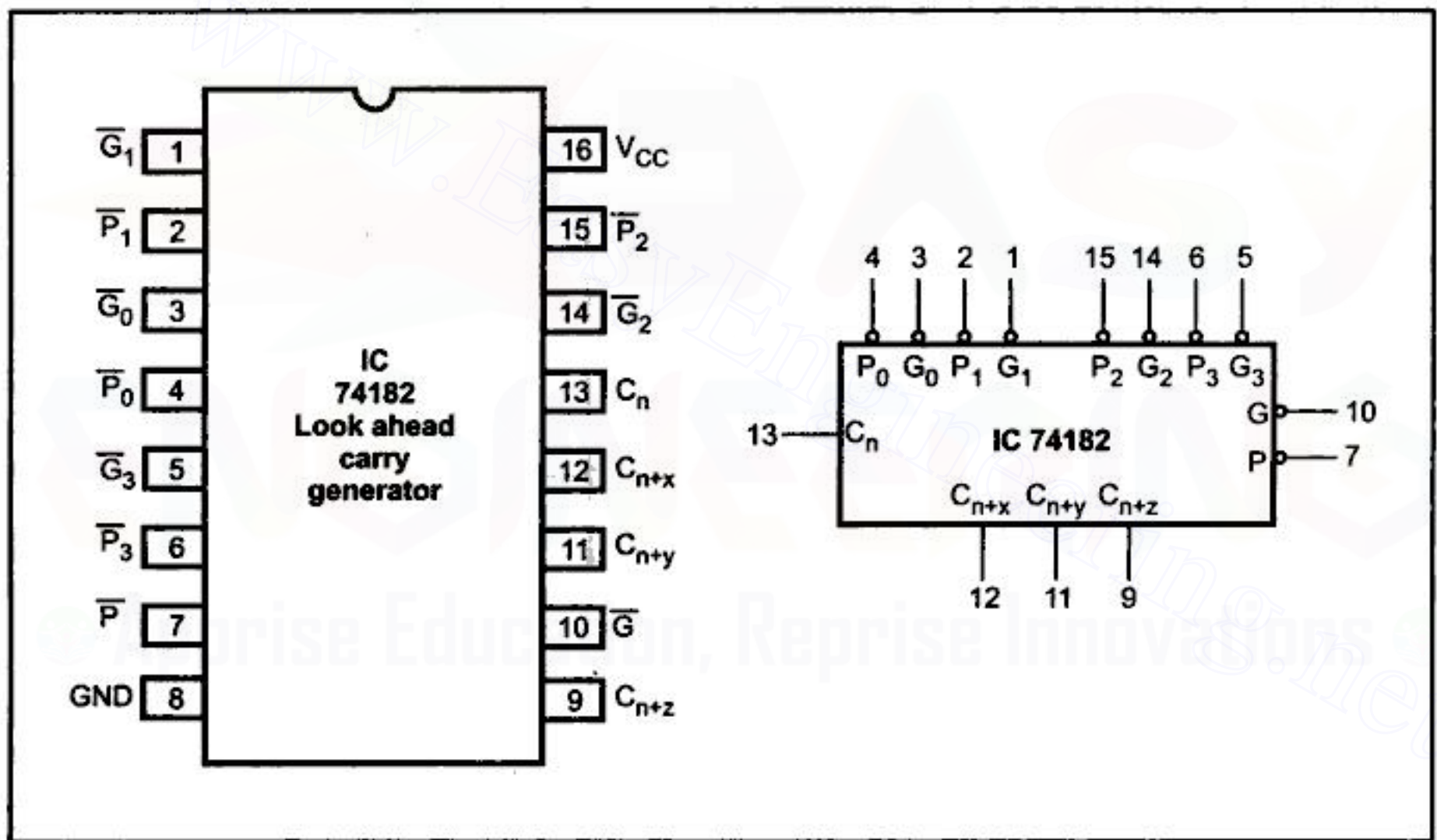


Fig. 7.39 (a) Pin diagram

Fig. 7.39 (b) Logic symbol

The logic equations provided at the outputs of 74182 are :

$$\begin{aligned}
 C_{n+x} &= G_0 + P_0 C_n \\
 C_{n+y} &= G_1 + P_1 G_0 + P_1 P_0 C_n \\
 C_{n+z} &= G_2 + P_2 G_1 + P_2 P_1 G_0 \\
 \bar{G} &= \overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0} \\
 \bar{P} &= \overline{P_3 P_2 P_1 P_0}
 \end{aligned}$$

7.10 Binary 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, as shown in Fig. 7.45. The mode input M controls the operation of the circuit. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B . When $M = 0$, we have $B \oplus 0 = B$. The full-adders receive the value of B , the input carry is 0, and the circuit performs A plus B . When $M = 1$, we have $B \oplus 1 = \bar{B}$ and $C_{in0} = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B , i.e. $A - B$.

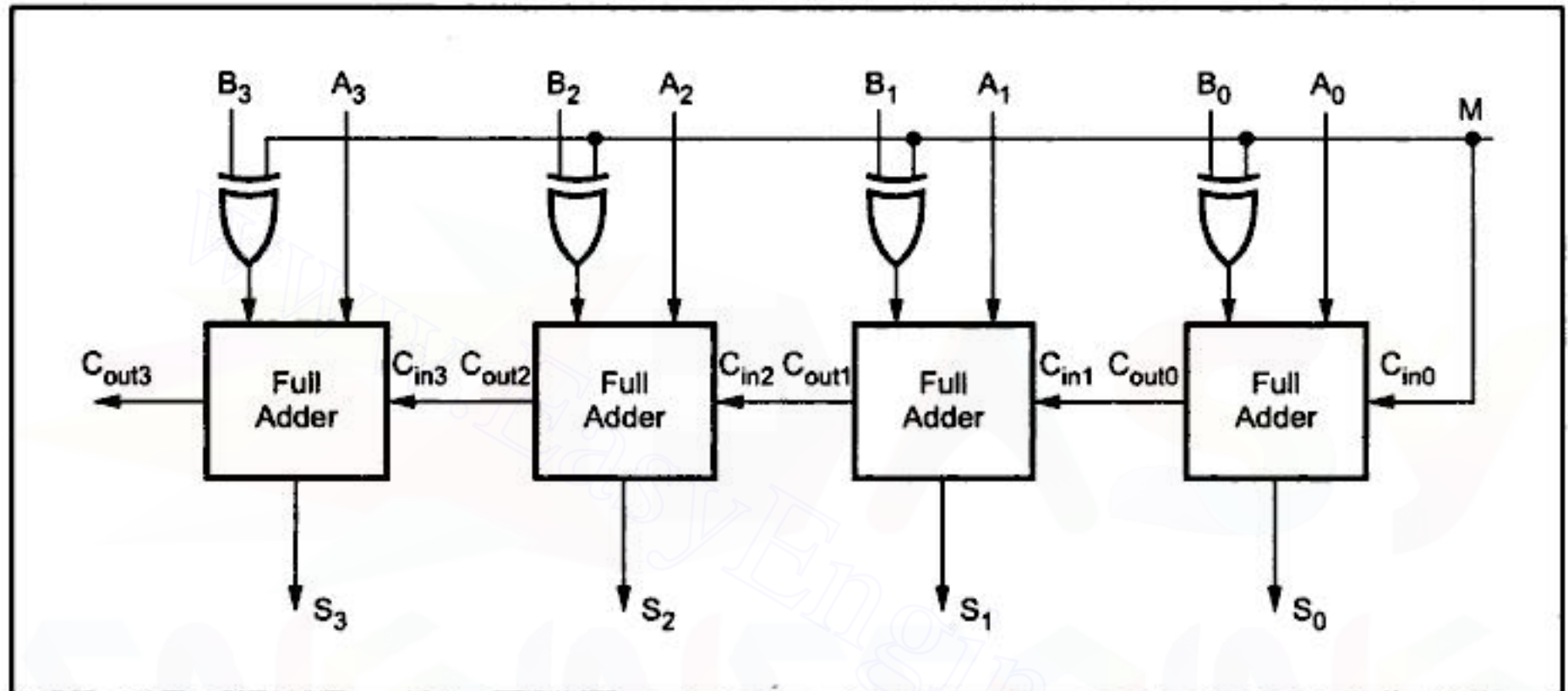


Fig. 7.44 4-bit adder-subtractor

7.11 BCD Adder

A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0000 to 1001, i.e. each BCD digit is represented as a 4-bit binary number. When we write BCD number say 526, it can be represented as

$$\begin{array}{ccc}
 5 & 2 & 6 \\
 \downarrow & \downarrow & \downarrow \\
 0101 & 0010 & 0110
 \end{array}$$

Here, we should note that BCD cannot be greater than 9.

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

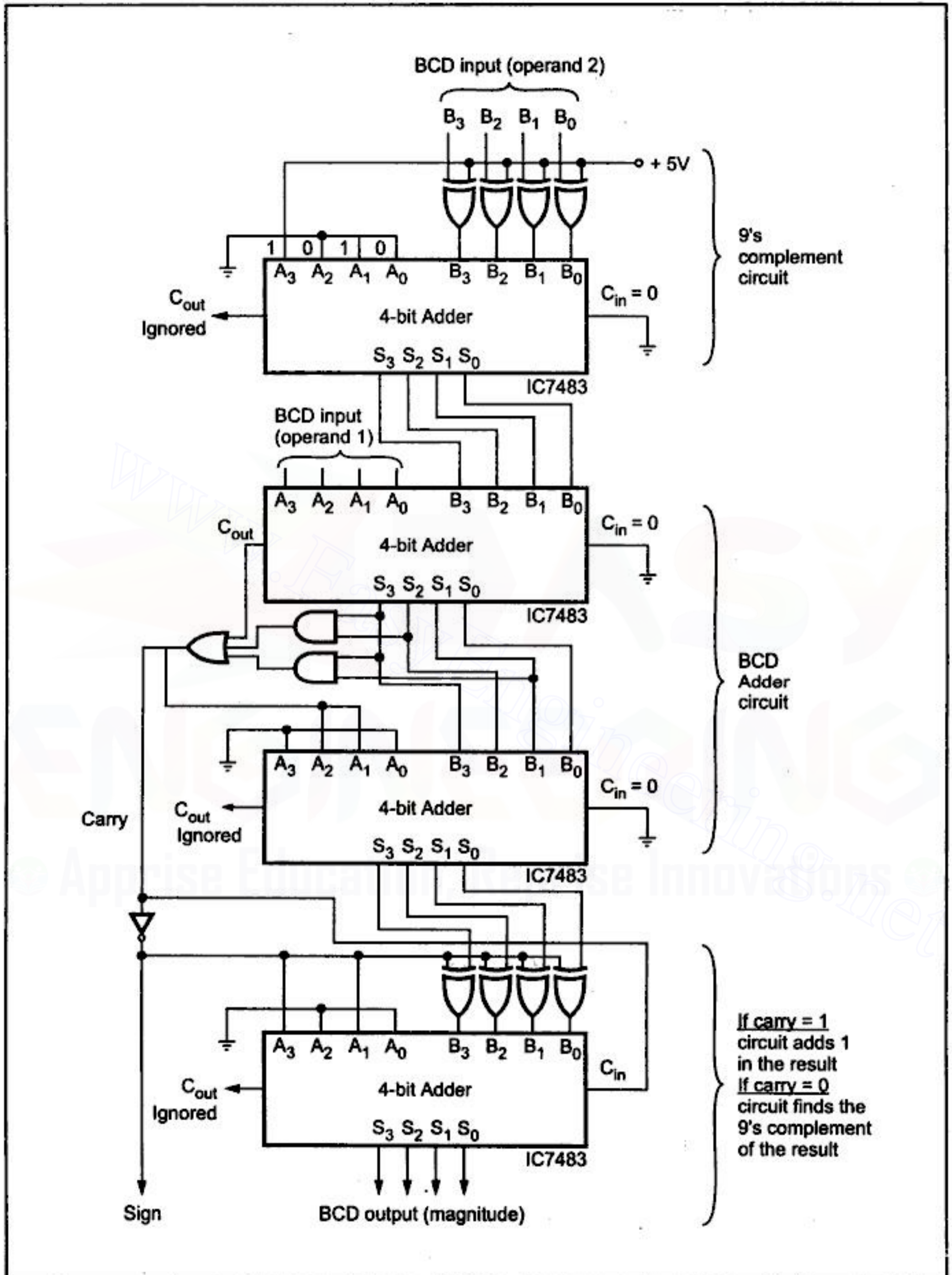


Fig. 7.49 4-bit BCD subtractor using 9's complement method

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 7.15

K-map Simplification

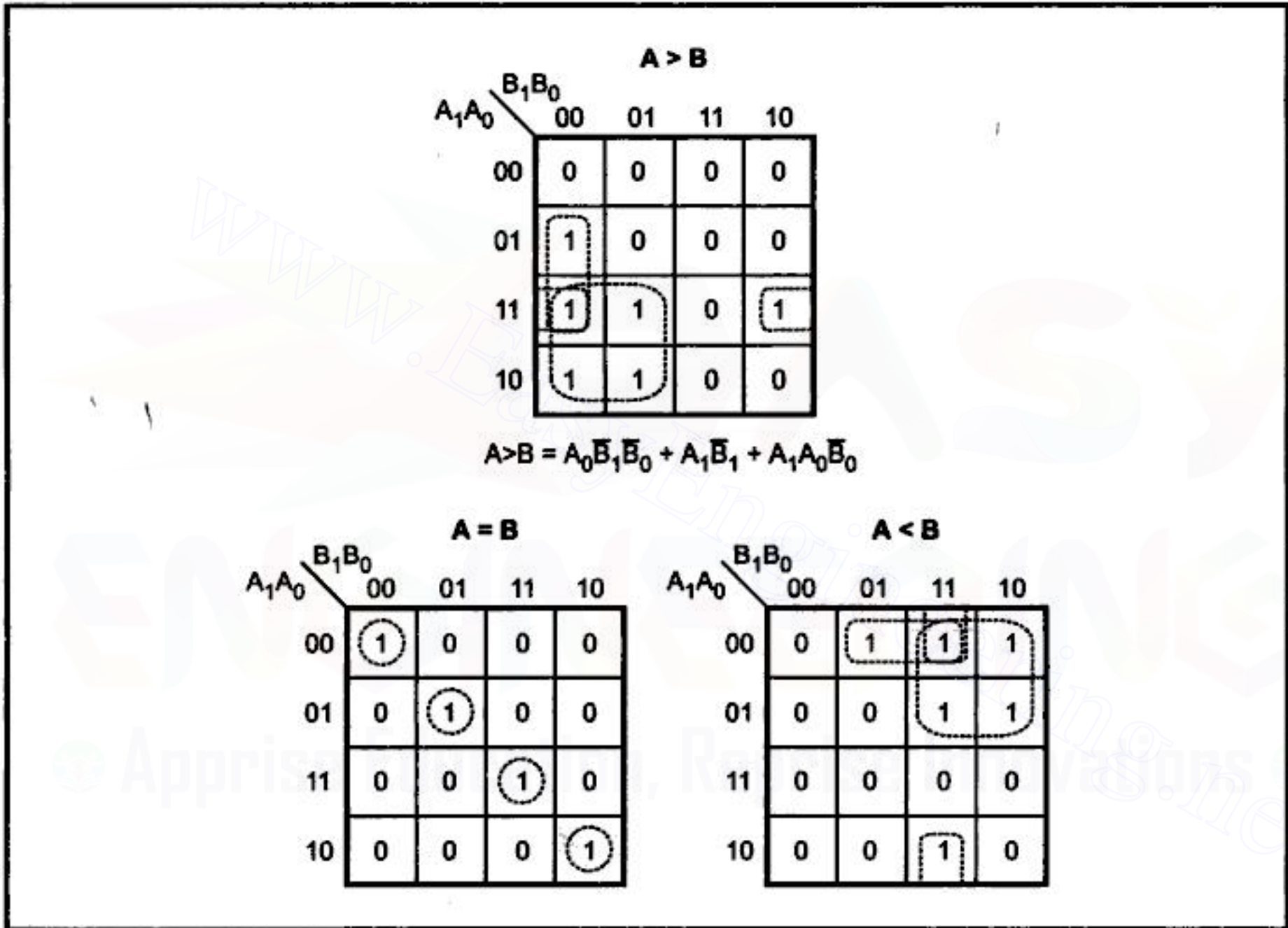


Fig. 7.54

$$\begin{aligned}
 A = B &= \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 \\
 &\quad + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0 \\
 &= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) \\
 &\quad + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0) \\
 &= (A_0 \odot B_0)(A_1 \odot B_1) \\
 A < B &= \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0 + \bar{A}_1B_1
 \end{aligned}$$

14. Draw and explain the working of 4-bit BCD subtractor using 10's complement method.
15. What do you mean by comparator ?
16. Design a 3-bit comparator using three one bit comparators and logic gates.
17. Explain the procedure for Excess-3 addition and subtraction.
18. Implement Excess-3 adder circuit using binary adders.
19. Implement Excess-3 subtractor circuit using binary subtractors.

University Questions

1. A majority function is a digital circuit whose output is 1 if and only if the majority of the inputs are 1. The output is 0 otherwise.
Write the truth table of 3-input majority function.
Show that a full adder circuit consists of a 3-input EXOR and a 3-input majority function. (May-92)
2. A 2-bit digital comparator accepts two words $A = a_2, a_1$, and $B = b_2, b_1$, and gives three outputs G, E and L.
 - i) The output G is HIGH when $A > B$
 - ii) The output E is HIGH when $A = B$
 - iii) The output L is HIGH when $A < B$
 - a) Write the truth table for this comparator.
 - b) Draw the Karnaugh maps for G, L and E outputs and write the SOP expressions for each.
 - c) Draw the logic diagram of this comparator. (May-92)
3. Explain the principle of look ahead carry.
(Dec-92, May-93, May-94, Dec-94, May-95, May-97, Dec-98)
4. Write a short note on half and full subtractor (Dec-92)
5. Design a BCD to excess-3 code converter using the logic gates. (May-93, May-95)
6. Design a 2-bit digital comparator using suitable logic gates.
(May-93, Dec-93, May-94, May-95, May-98)
7. Write a short note on
 - i) Parallel Binary Adder ii) Look ahead carry (May-93, Dec 2000)
8. Design a circuit to implement the code conversion from Binary to excess -3. (Dec-93)
9. Implement an 8 bit magnitude comparator using 4 - bit magnitude comparator IC_s (7485). (Dec-93)
10. Implement a full adder circuit with 3:8 decoder and few logic gates. (Dec-93)
11. Design and implement a full subtractor. Show how a full subtractor can be implemented using half subtractor blocks. (Dec-93)
12. Write a short note on
 - i) half and full subtractor. ii) Look ahead carry (Dec-93, May-94)
13. a) Design an excess 3 to BCD code converter using a 4 bit full adder IC.
b) Design a 5211 Gray to BCD code converter using AND-OR-NOT gates.

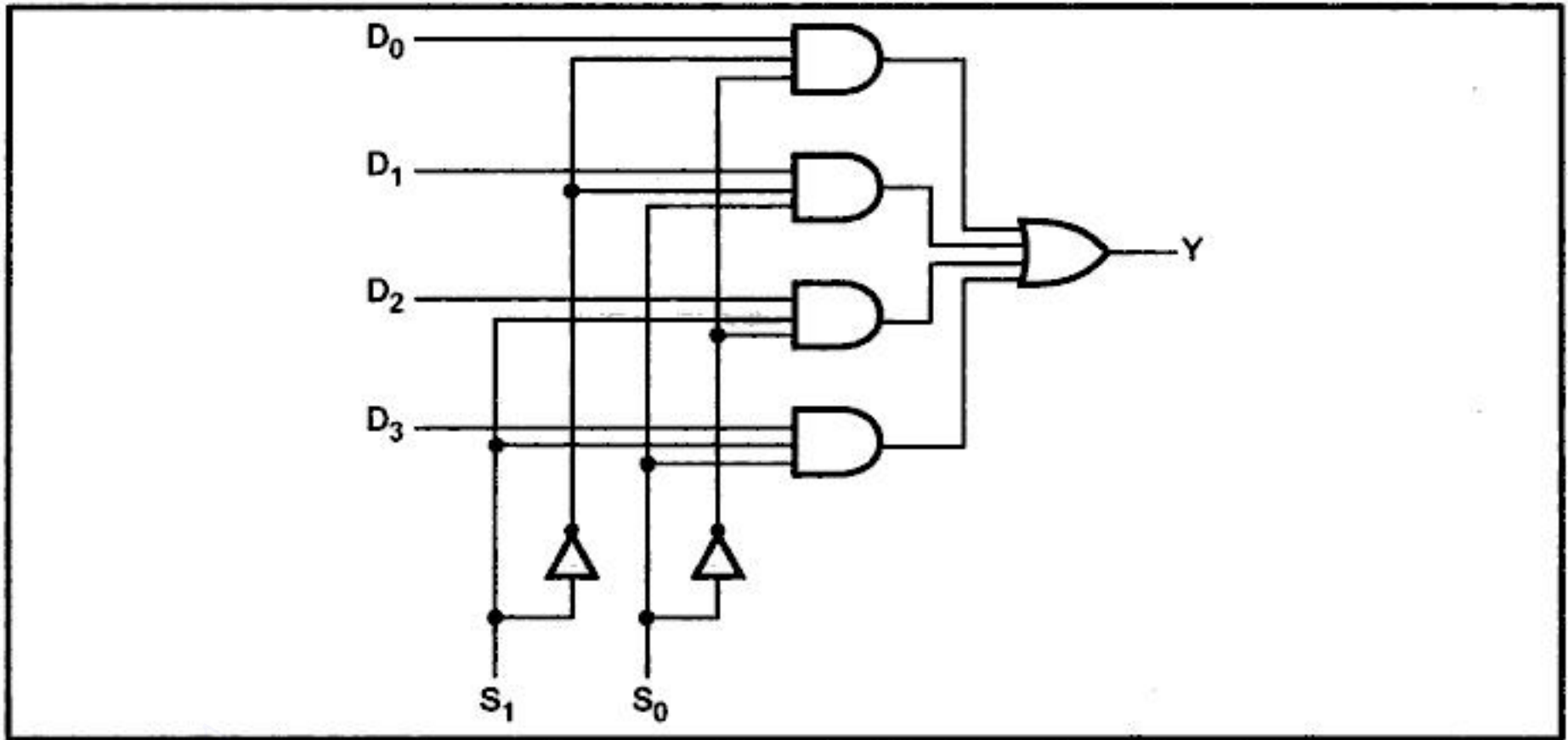


Fig. 8.1 (a) Logic diagram

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Fig. .8.1 (b) Function table

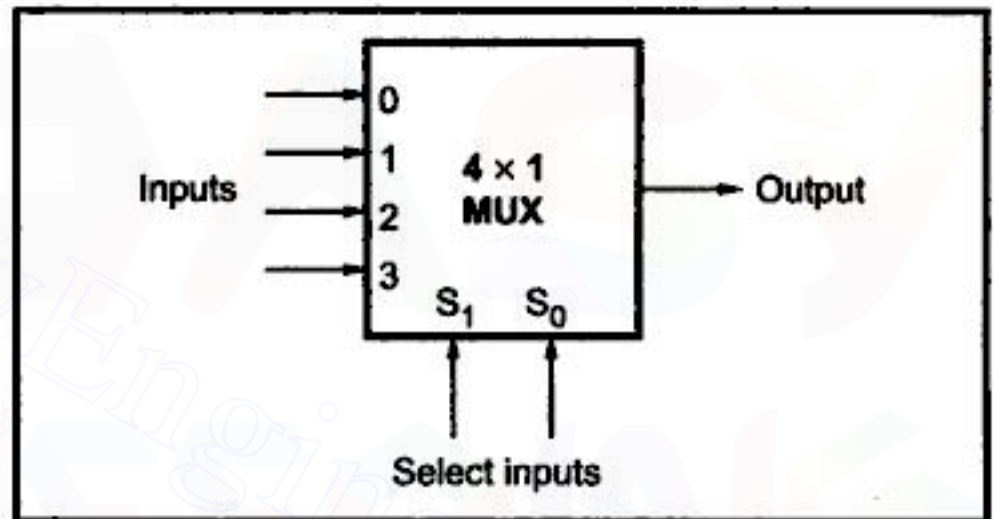


Fig. 8.1 (c) Block diagram

Fig. 8.1 4-to-1 line multiplexer

In some cases, two or more multiplexers are enclosed within one IC package, as shown in the Fig. 8.2. The Fig. 8.2 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

In the previous multiplexer implementation two least significant variables, i.e. B and C are connected to the select lines of the multiplexer. The multiplexer implementation is also possible by connecting most significant variables i.e. A and B (in above case) to the select lines of the multiplexer. The procedure for same is explained below :

Here, implementation table consists of two columns. The first column lists all the minterms where least significant variable is complemented (\bar{C} in this case), and the second column lists all the minterms with least significant variable is uncomplemented (C in this case). The minterms given in the function are circled and then each row is inspected separately as follows :

- If the two minterms in a row are not circled, 0 is applied to corresponding multiplexer input.
- If the two minterms in a row are circled, 1 is applied to corresponding multiplexer input.
- If the minterm in the column 1 is circled, least significant variable is complemented (\bar{C} in this case) and applied to the corresponding multiplexer input.
- If the minterm in the column 2 is circled, least significant variable (C in this case) is applied to the corresponding multiplexer input.

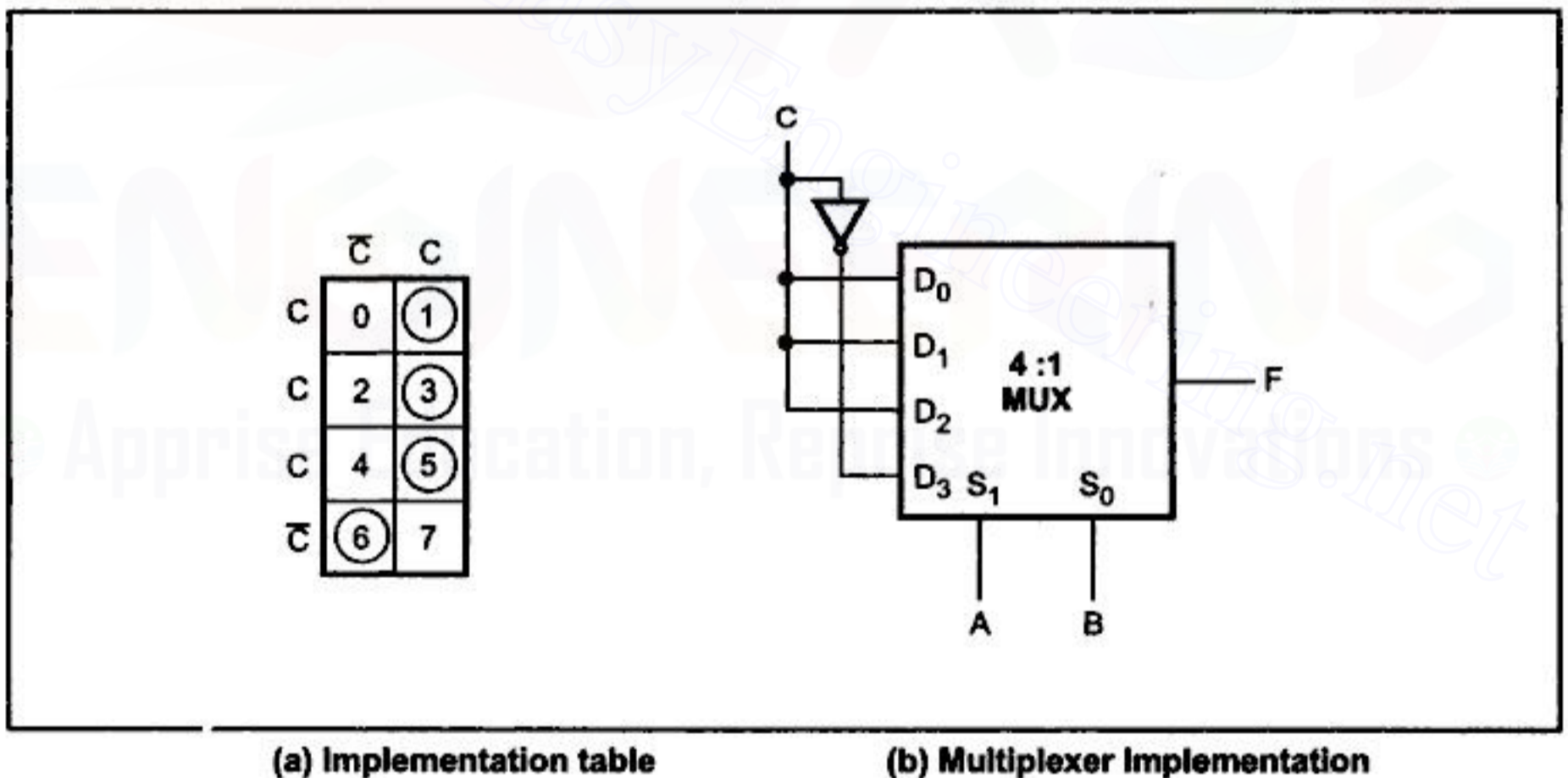
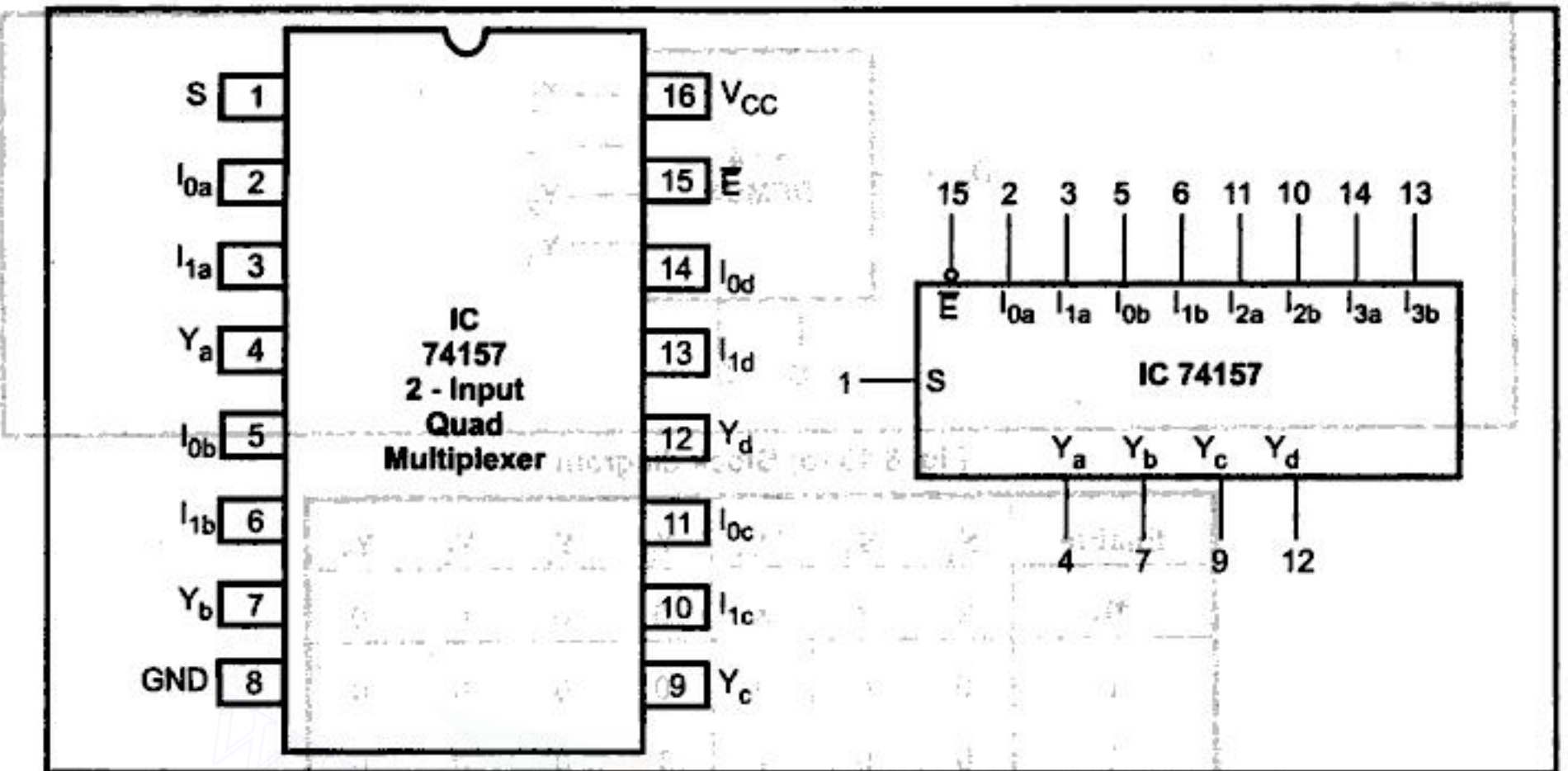


Fig. 8.6

Ex. 8.4 : Implement the following Boolean function using 8 :1 MUX.

$$F(P, Q, R, S) = \sum m(0, 1, 3, 4, 8, 9, 15)$$

Sol.: Fig. 8.7 shows the implementation of given Boolean function with 8 : 1 multiplexer.



(a) Pin diagram

Fig. 8.14

(b) Logic symbol

8.3 Demultiplexer

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. 8.15 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.

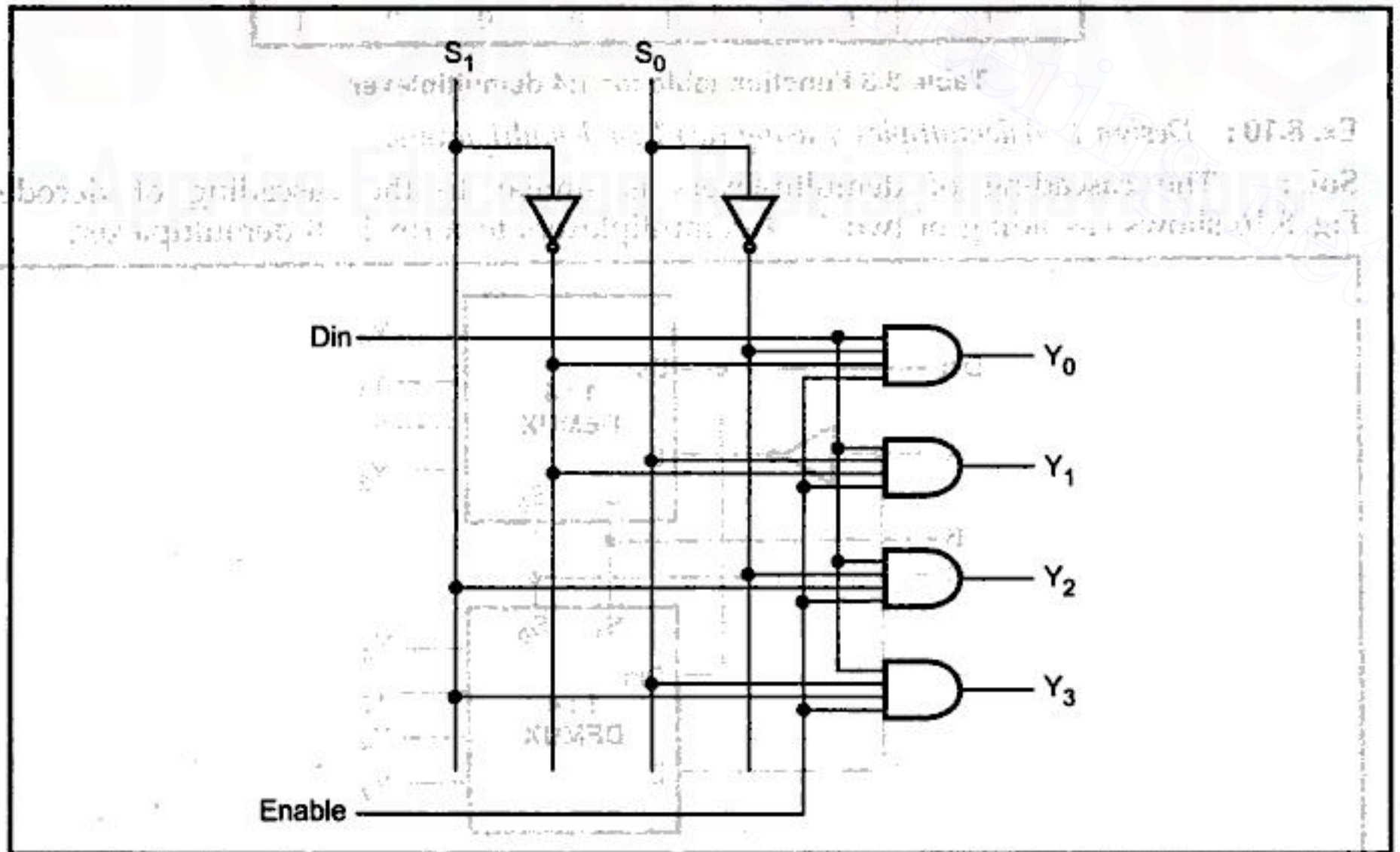


Fig. 8.15 (a) Logic diagram

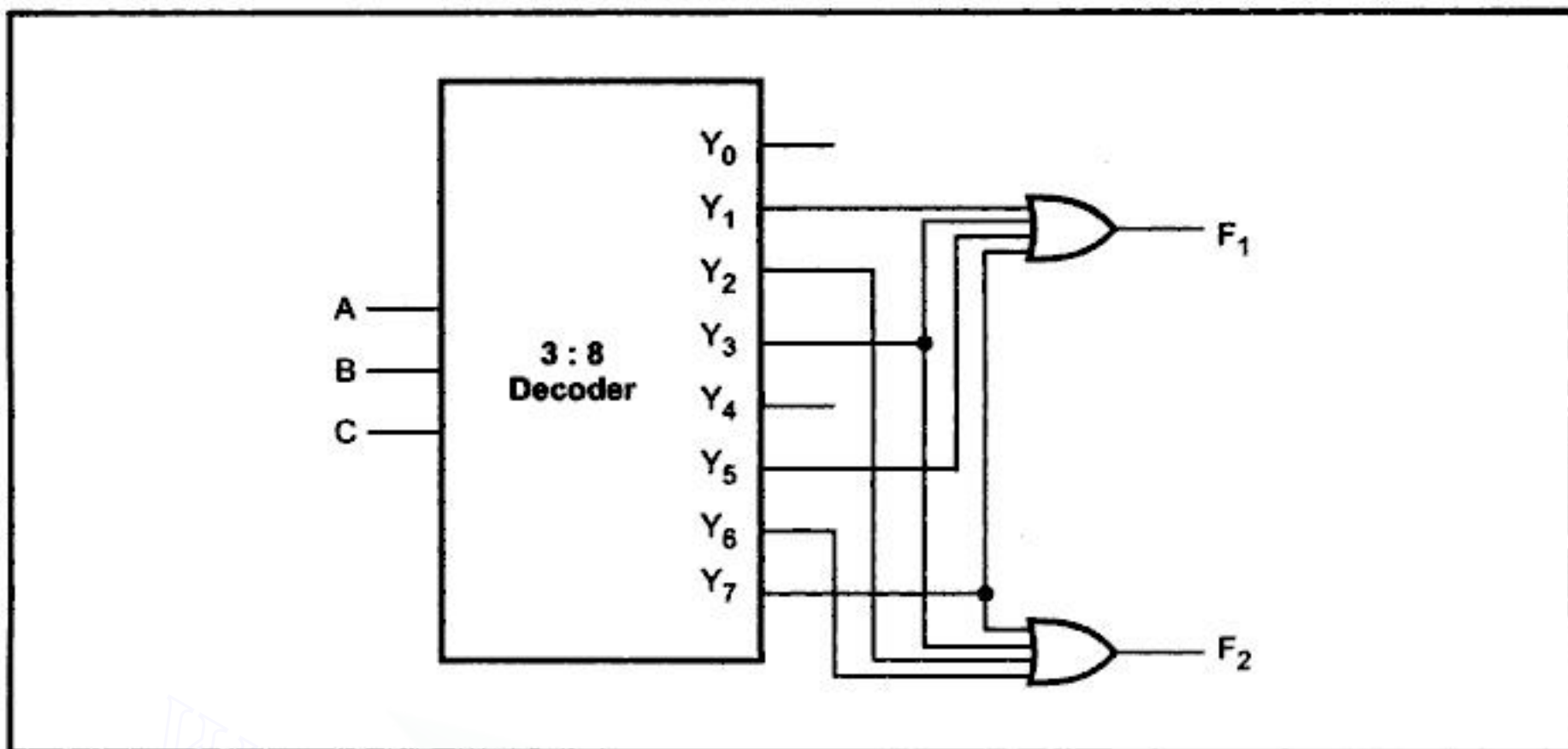


Fig. 8.20 Boolean function implementation using decoder and gates

8.4.2 Decoder IC 74138

The IC 74138 accepts three binary inputs (A_0, A_1, A_2) and when enabled provides eight individual active low outputs ($O_0 - O_7$). The device has three enable inputs : two active low (\bar{E}_1, \bar{E}_2) and one active high (E_3). Fig. 8.21 and 8.22 show pin diagram and function table.

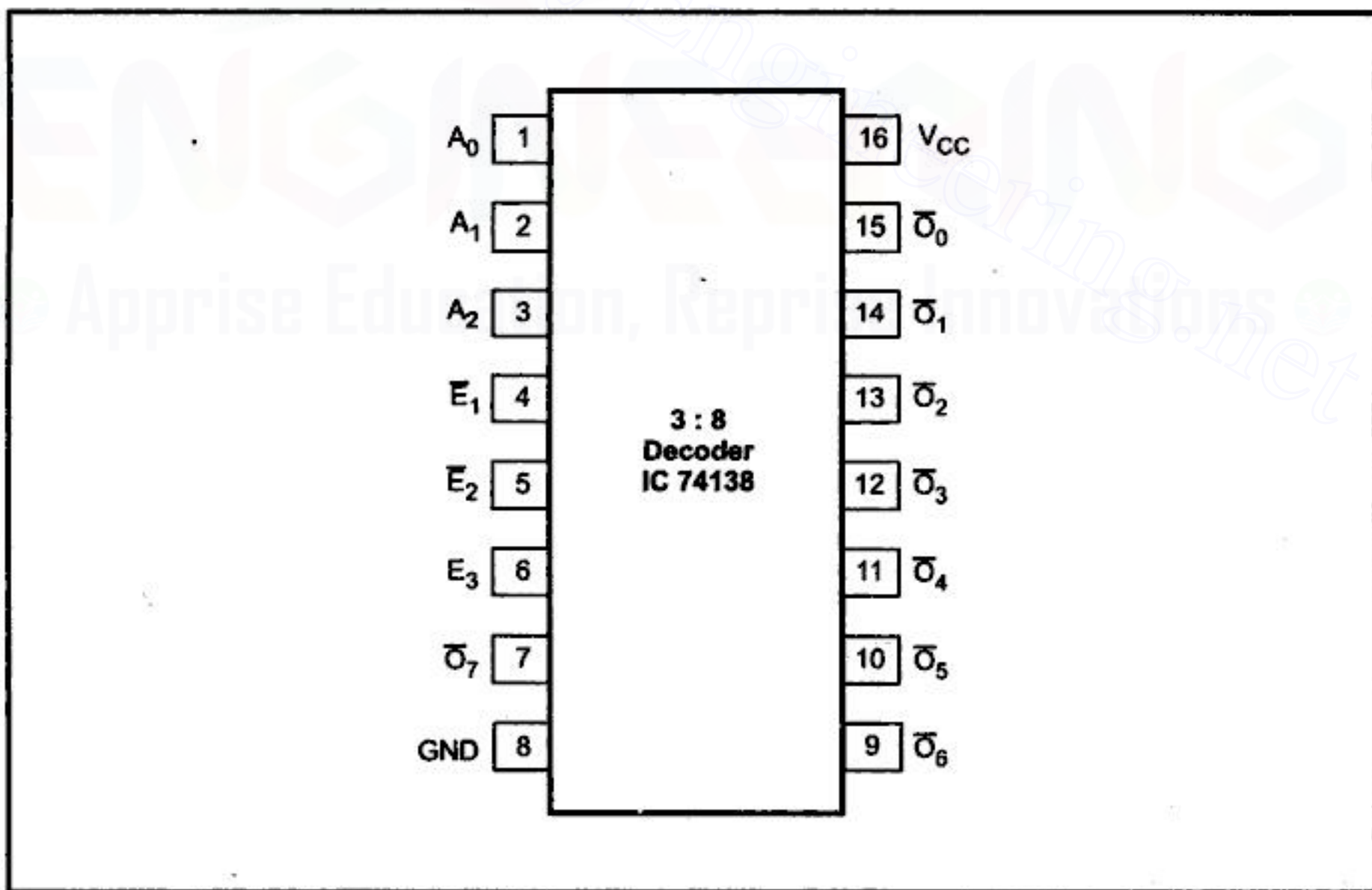


Fig. 8.21 Pin diagram

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Table 8.9 Truth table for BCD-to-common-cathode 7-segment decoder/driver

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

Table 8.10 Truth table for BCD-to-common-anode 7-segment decoder/driver

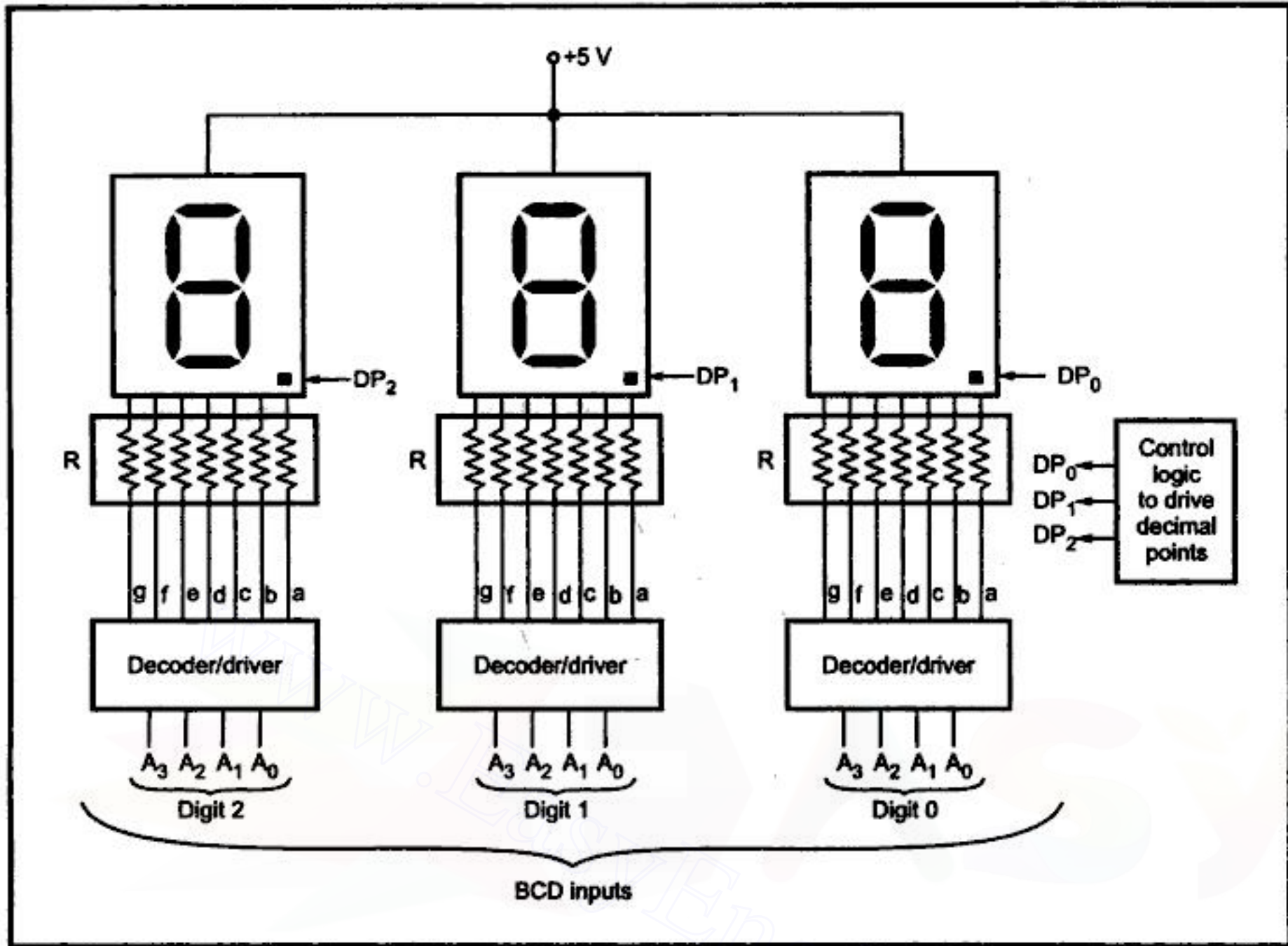


Fig. 8.36 Cascaded multi-digit non multiplexed display system

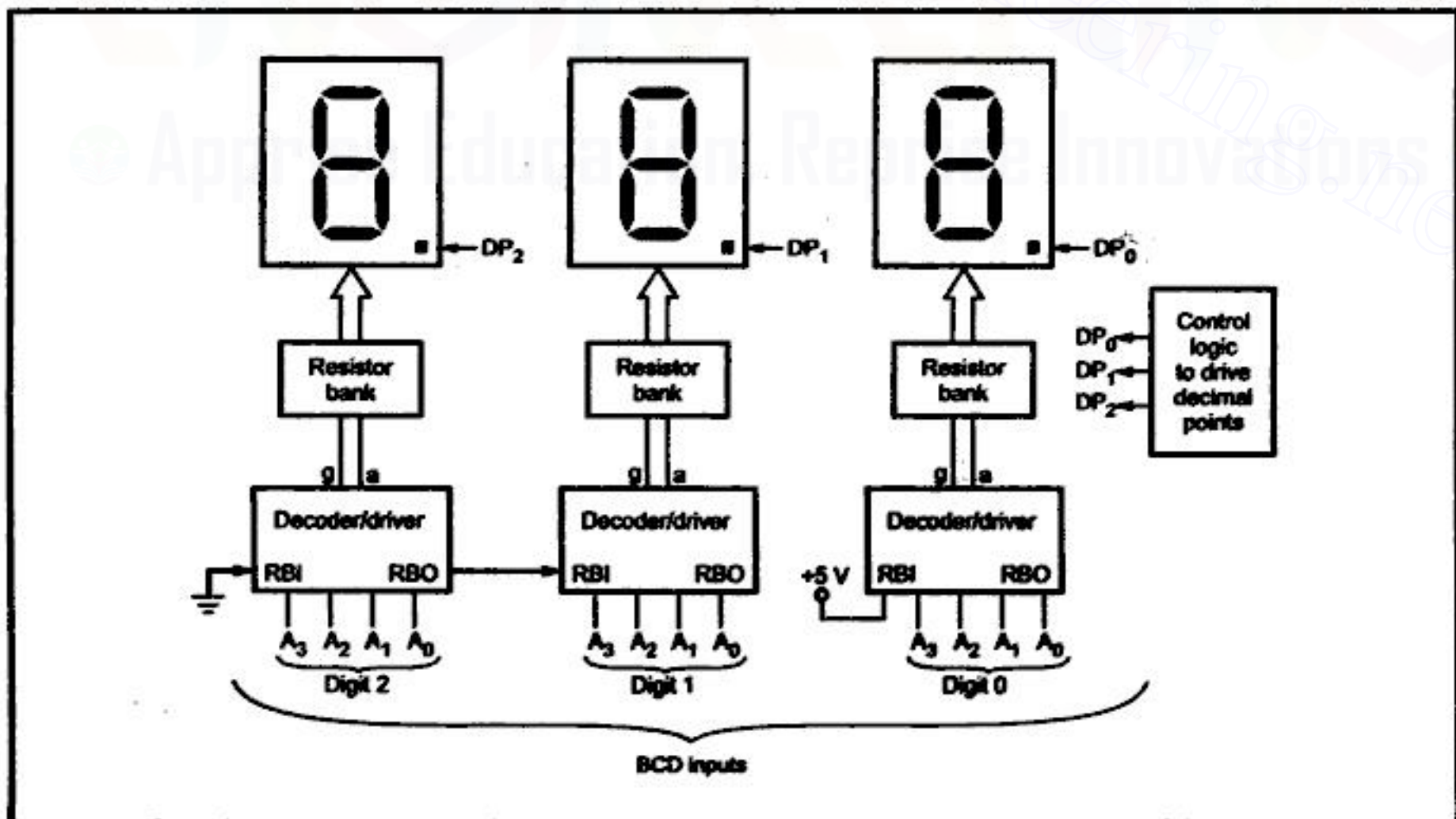


Fig. 8.37 Circuit for blanking leading zeroes

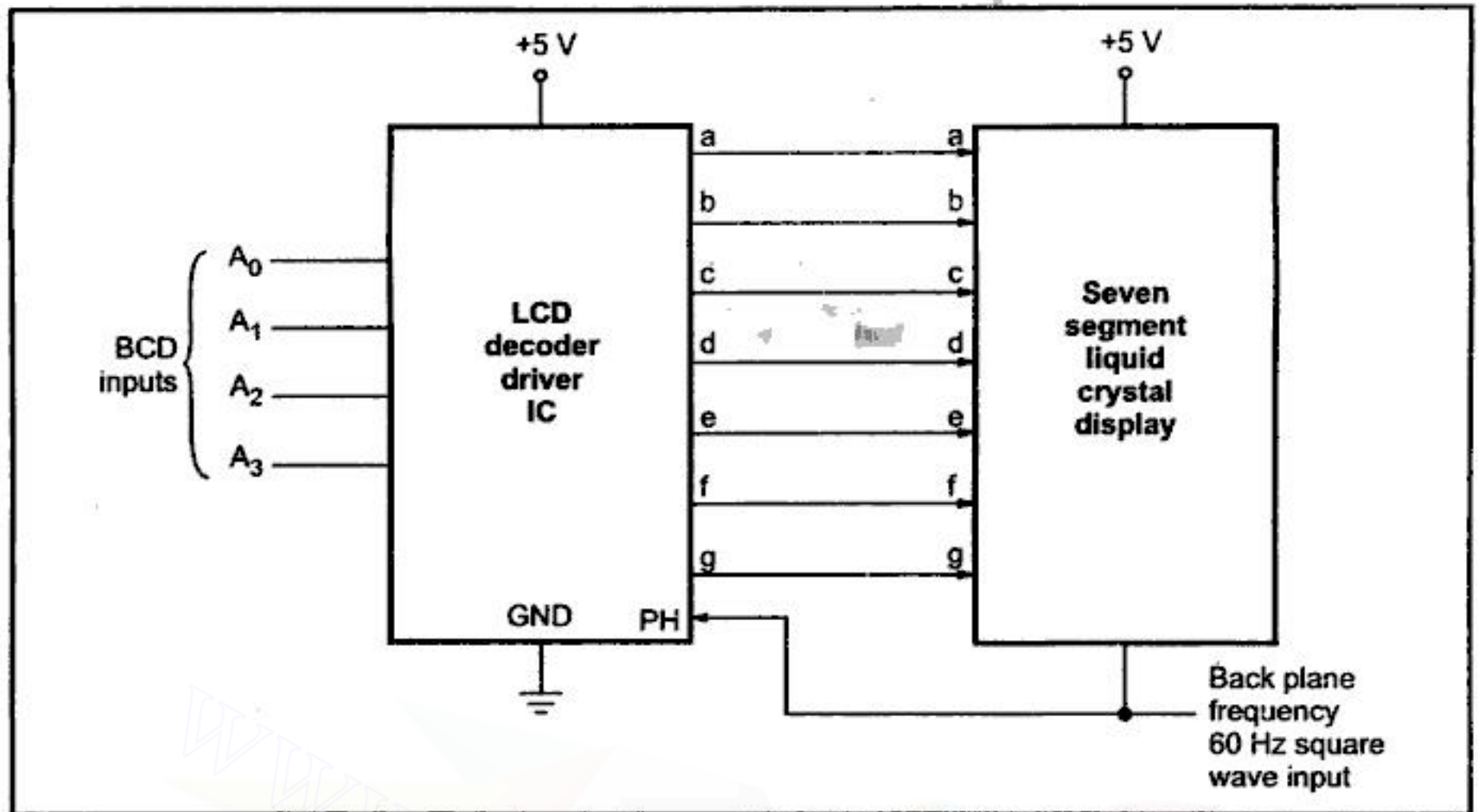


Fig. 8.41 Basic connections for driving LCD seven segment display

IC 4543B

The 4543B BCD-to-7-segment latch/decoder/driver is designed for use with liquid crystal displays and is constructed with complementary MOS (CMOS) enhancement mode devices. Fig. 8.42 shows the pin diagram for IC 4543 B.

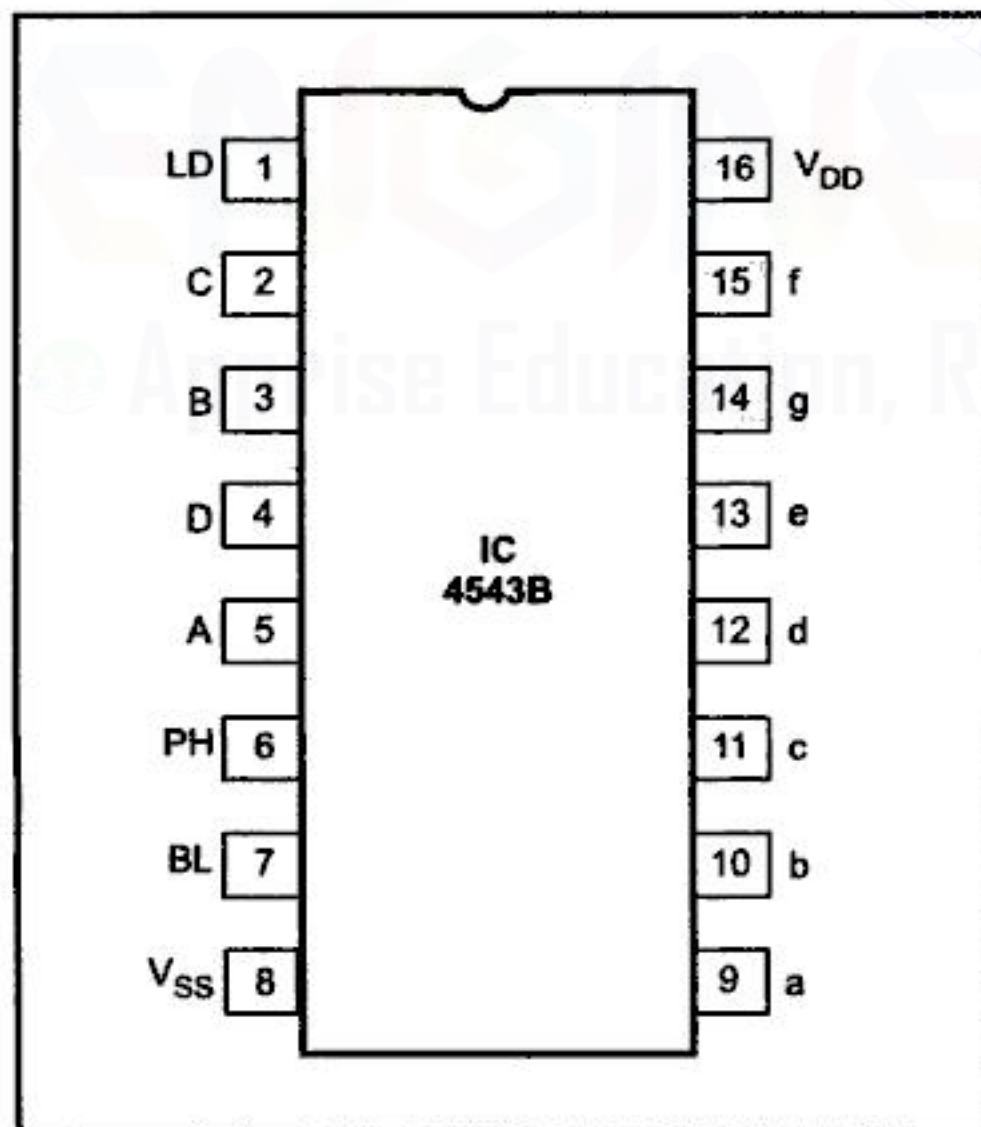


Fig. 8.42 Pin diagram for IC 4543 B

Pins A, B, C and D represent BCD inputs with A as a least significant bit (LSB) and D as a most significant bit (MSB). Pins a through g are the seven segment outputs. The 4543 B has three control terminals : LD (Latch Disable), PH (Phase), and BL (Blank). In normal use the LD terminal is held high and BL terminal is tied low. The state of the PH terminal depends on the type of display that is being driven. For driving LCD displays, a square wave (about 60 Hz swinging fully between the GND and V_{CC} values) must be applied to the phase terminal. For driving common cathode LED displays, phase must be grounded and for driving common anode LED displays, phase must be tied to logic high.

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

Logic diagram

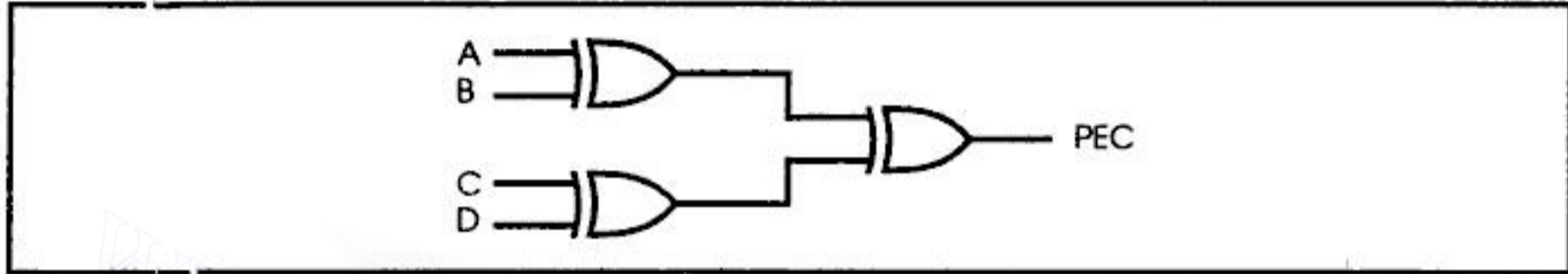


Fig. 8.47

Let us study the standard IC for parity generator and checker.

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. 8.48 shows pin diagram and logic diagram for 74180.

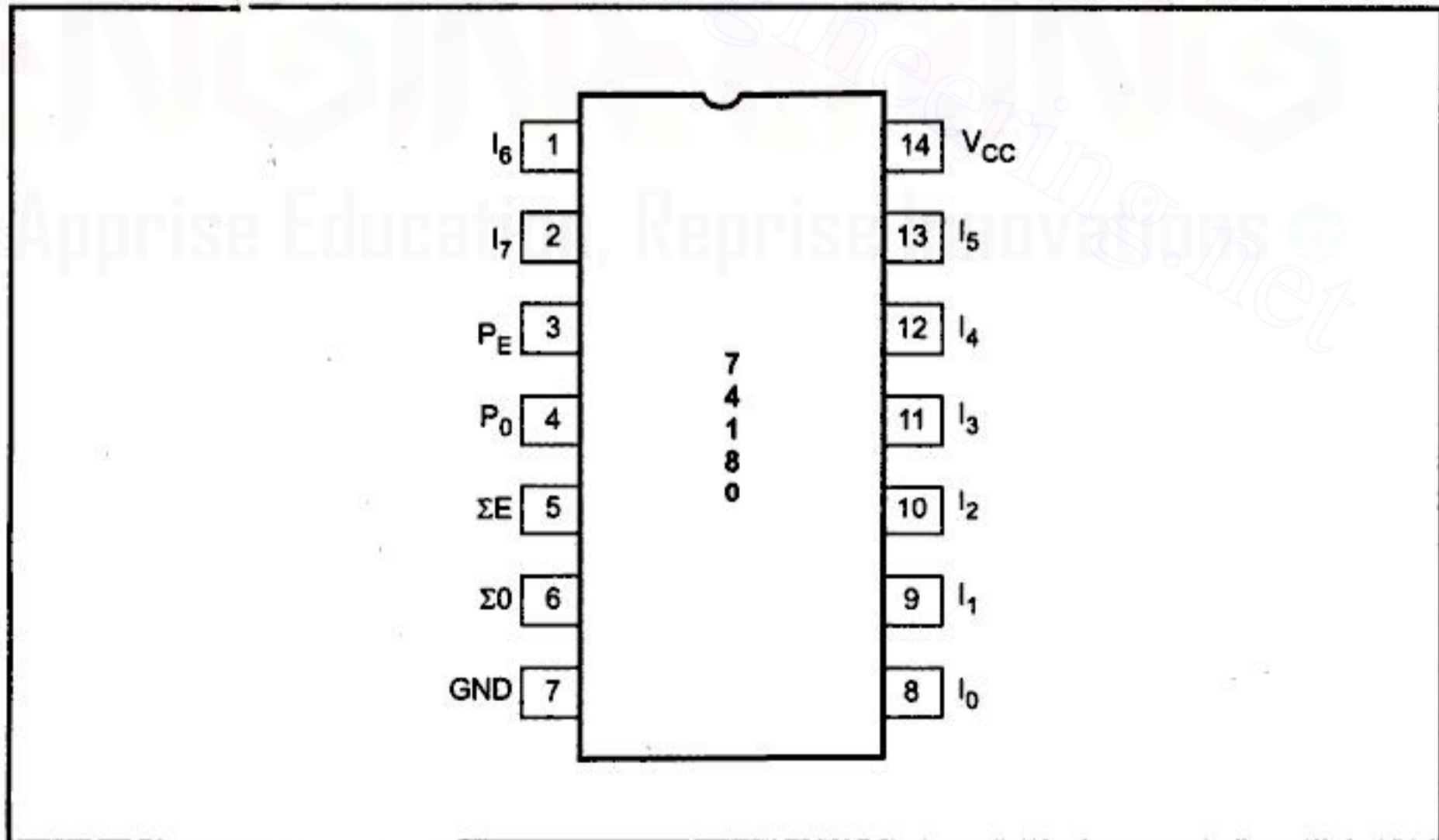


Fig. 8.48 (a) Pin diagram

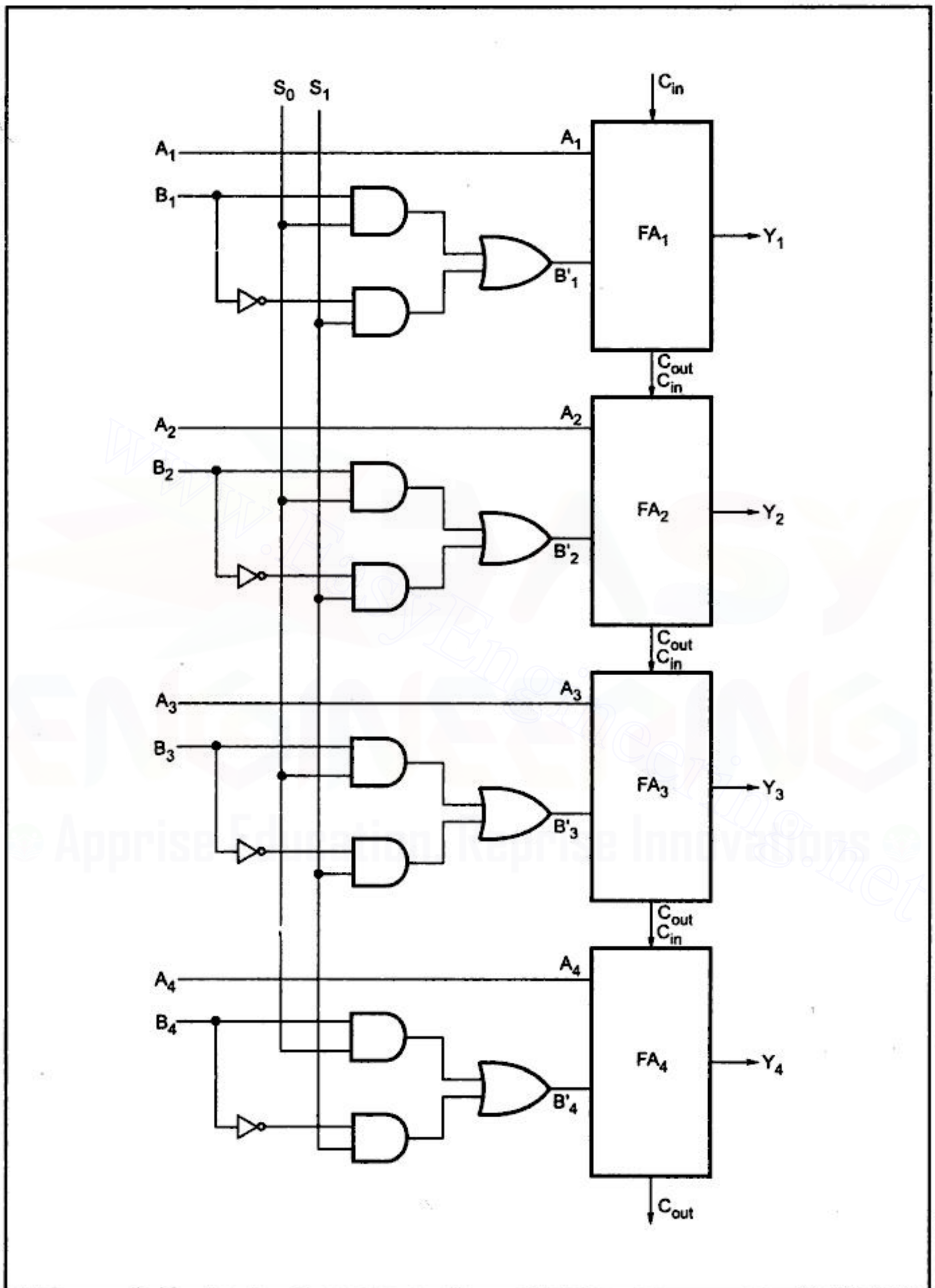


Fig. 8.52 Logic diagram of arithmetic circuit

8.9 Arithmetic Logic Unit (IC 74181)

The arithmetic and logic unit performs all the necessary arithmetic and logical operations. It requires one or two operands upon which it operates and produces a result. It is basically a multifunction combinational logic circuit. It provides select input to select the particular operation. In this section we study the very popular ALU IC, IC 74LS181.

IC74LS181

It is a 4-bit Arithmetic Logic Unit (ALU). Its features are as given below :

Features

- Provides 16 arithmetic operations : add, subtract, compare, double, plus twelve other arithmetic operations.
- Provides all 16 logic operations of two variables : exclusive-OR, compare, AND, NAND, OR, NOR, plus ten other logic operations.
- Full lookahead for high speed arithmetic operation on long words.

Fig. 8.57 and Fig. 8.58 show the block diagram and connection diagram for IC74LS181.

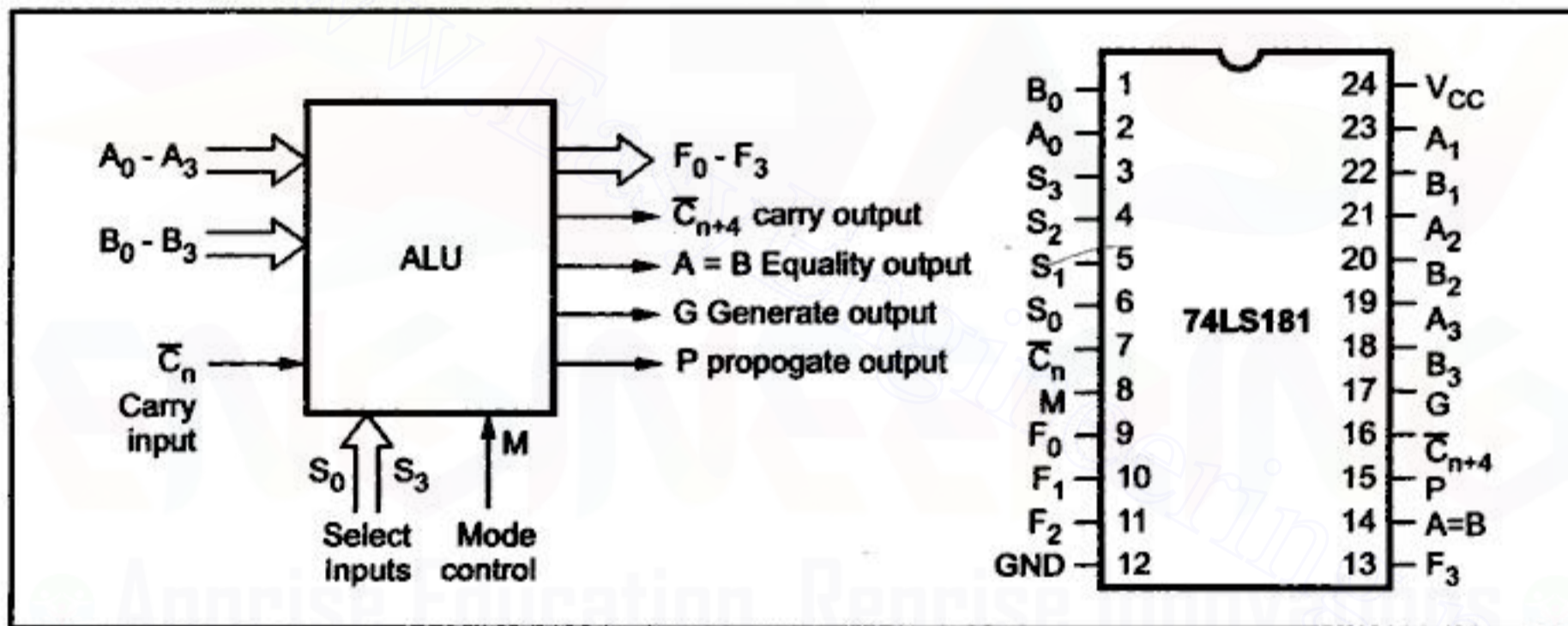


Fig. 8.57 Block diagram

Fig. 8.58 Connection diagram

As shown in the Fig. 8.57, 74LS181 has two four bit operands ($A_0 - A_3$ and $B_0 - B_3$). Its mode select input selects one of the two modes : arithmetic or logic, and four function select inputs select a particular function from the selected mode. Table 8.19 gives the pin description for IC 74LS181 and Table 8.20 gives the function table for IC74LS181.

Pin Names	Description
$A_0 - A_3$	Operand Inputs
$B_0 - B_3$	Operand Inputs
$S_0 - S_3$	Function Select Inputs
M	Mode Control Input

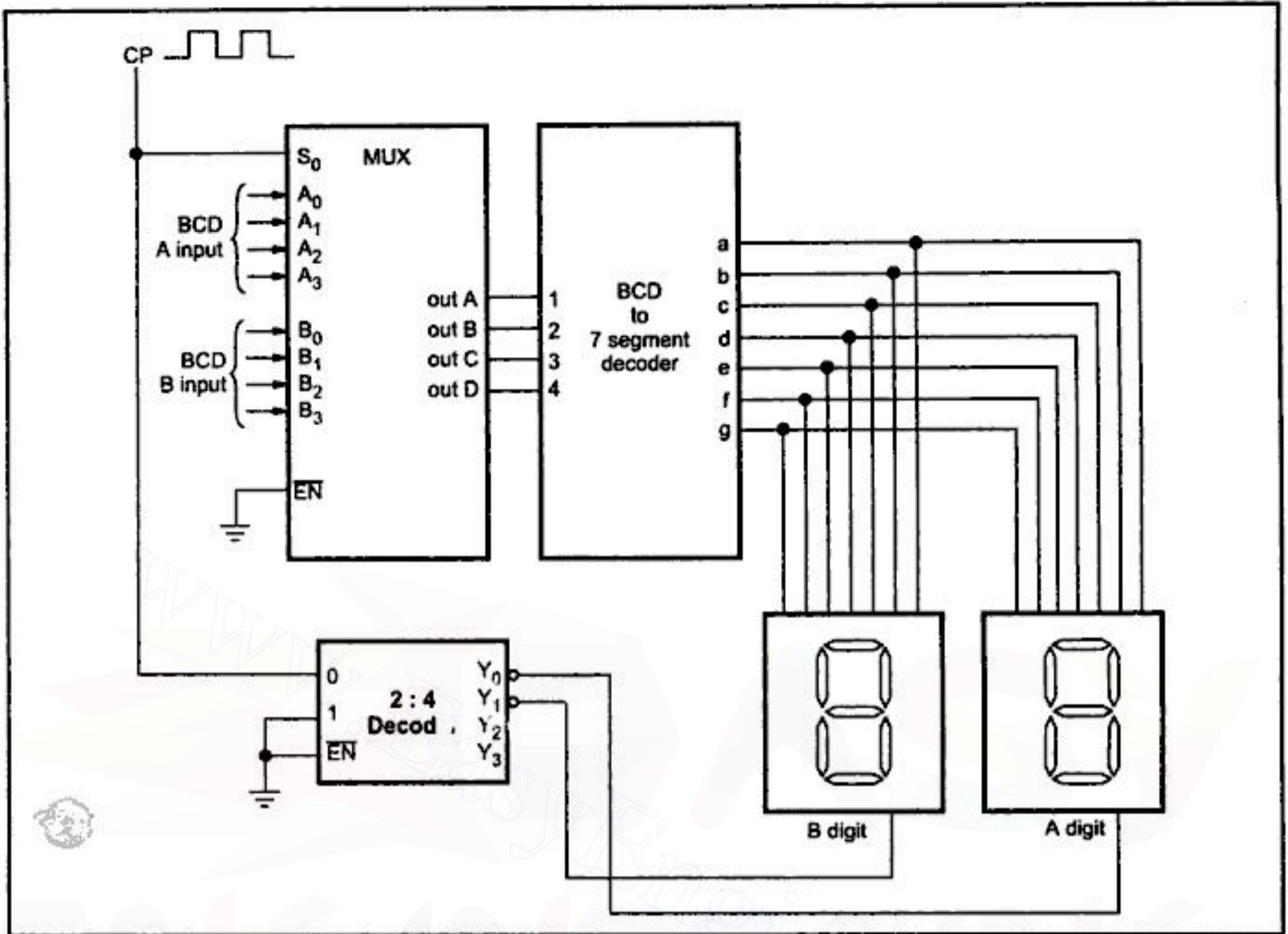


Fig. 8.62

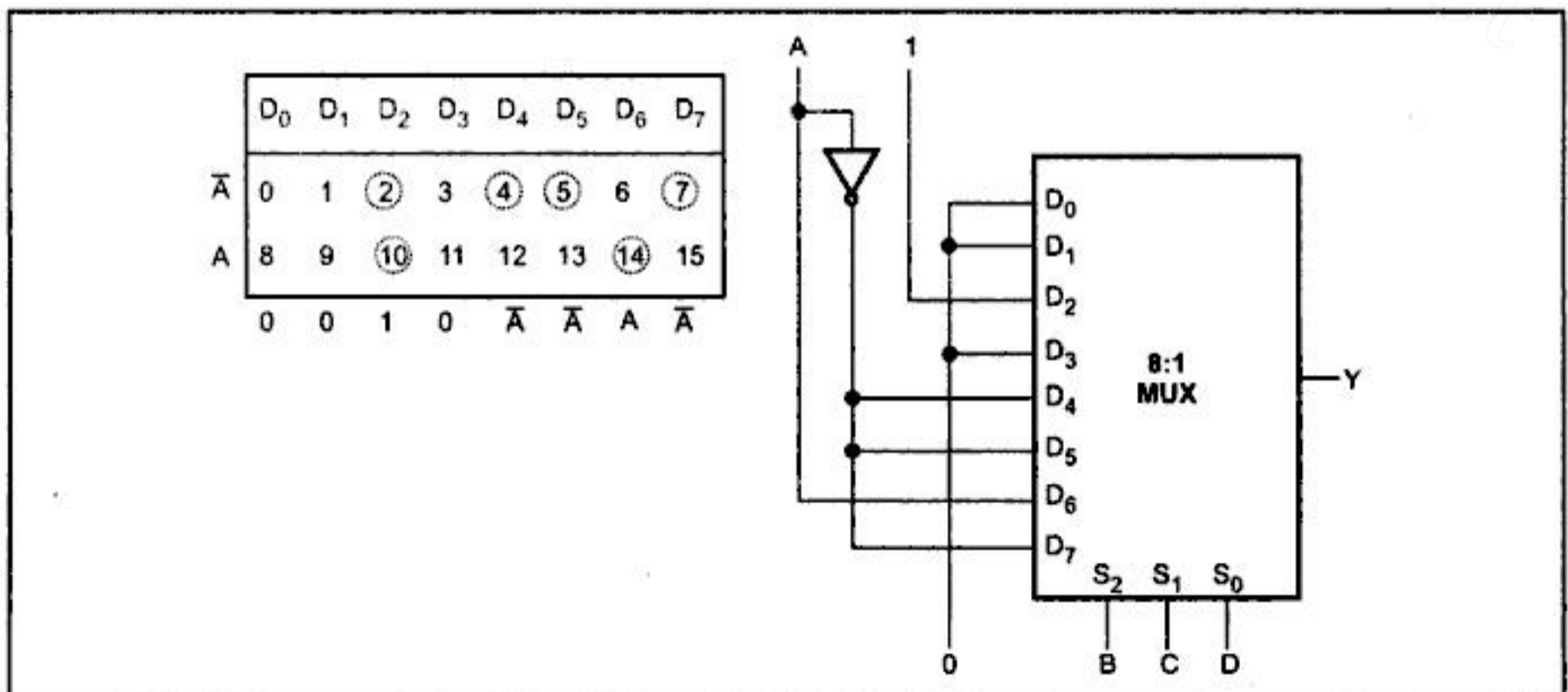
Ex. 8.22 : Implement the following using 8 : 1 MUX

i) $f(A, B, C, D) = \sum m(2, 4, 5, 7, 10, 14)$

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

(May-99)

Sol.: i)



(a) Implementation table

(b) Multiplexer implementation

Fig. 8.63

9

Flip-Flops, Registers and Counters

9.1 Introduction

So far we have studied the analysis and design of combinational digital circuits. It constitutes only a part of digital systems. The other major aspect of digital system is analysis and design of sequential circuits.

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement cannot be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback from the output back to the input.

Fig. 9.1 shows the block diagram of sequential circuit. As shown in the Fig. 9.1, memory elements are connected to the combinational circuit as a feedback path.

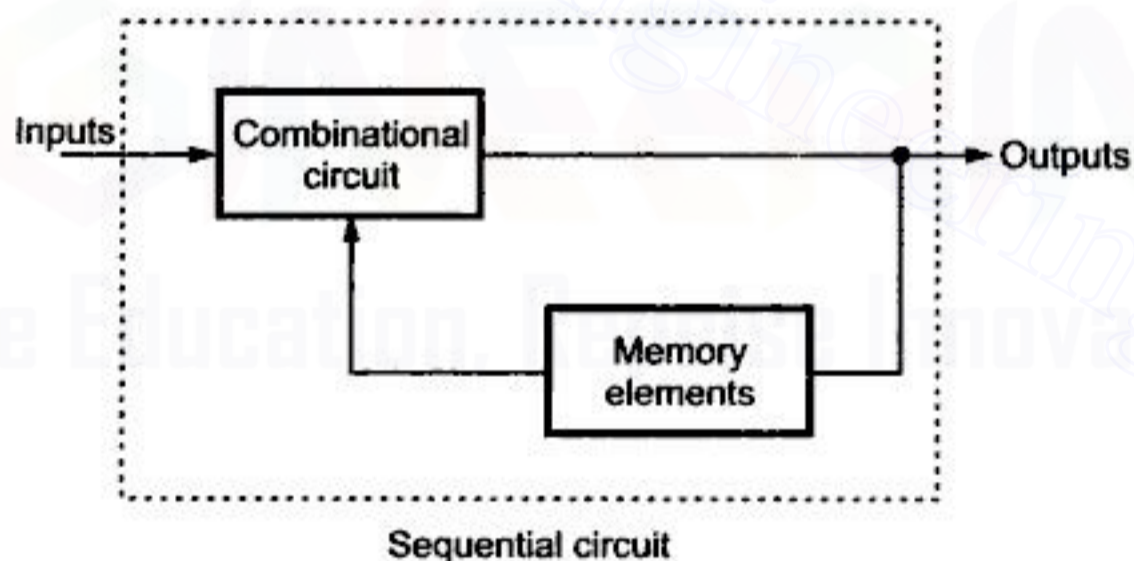


Fig. 9.1 Block diagram of sequential circuit

The information stored in the memory elements at any given time defines the present state of the sequential circuit. The present state and the external inputs determine the outputs and the next state of the sequential circuit. Thus we can specify the sequential circuit by a time sequence of external inputs, internal states (present states and next states), and outputs.

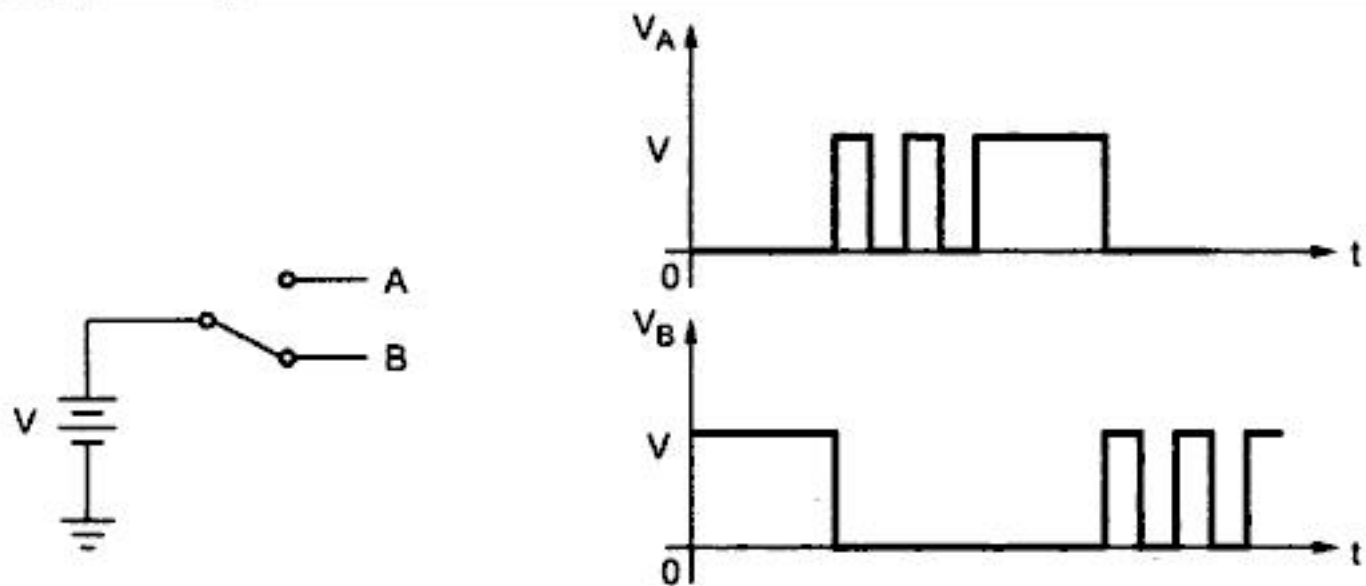


Fig. 9.6 Effect of keydebounce

output does not change, preventing debouncing of key output. In other words, we can say that the output does not change during transition period, eliminating key debounce.

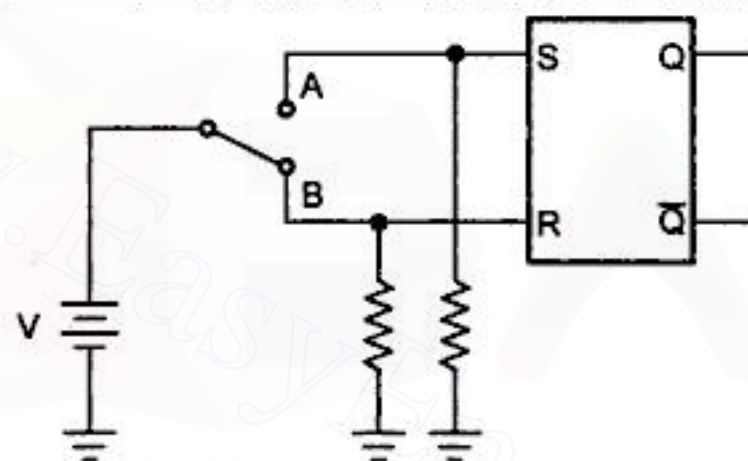


Fig. 9.7 (a) Switch debouncer

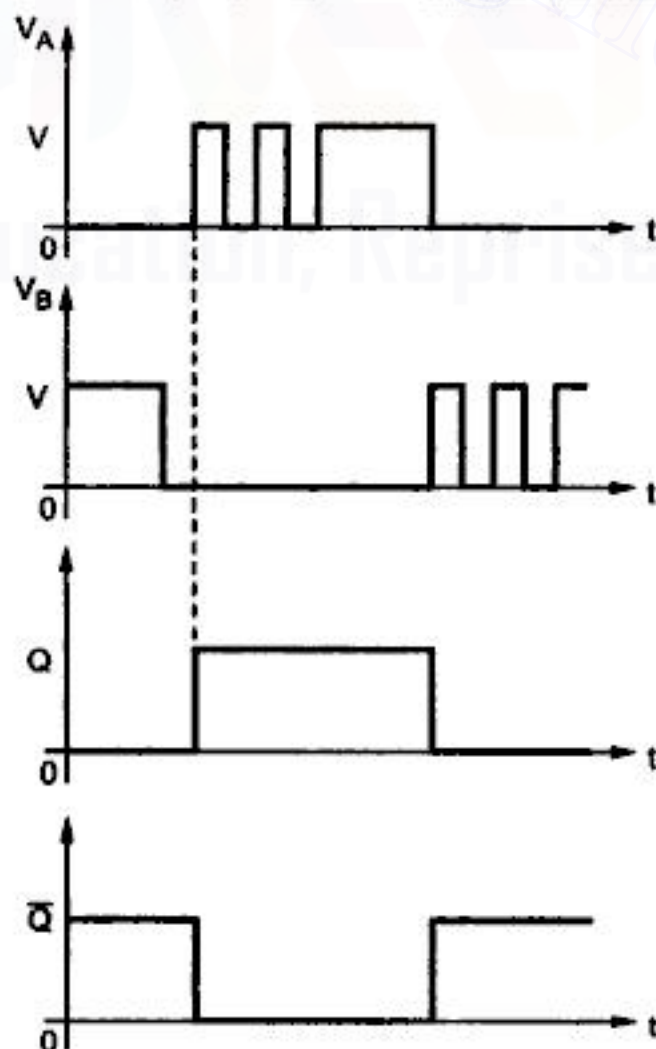


Fig. 9.7 (b) Waveforms of switch debouncer

pulse and is sensitive to its inputs only at this transition of the clock. Fig. 9.19 shows the input and output waveforms for positive edge triggered JK flip-flop. As shown in the Fig. 9.19, race around condition is avoided with edge triggering the flip-flop.

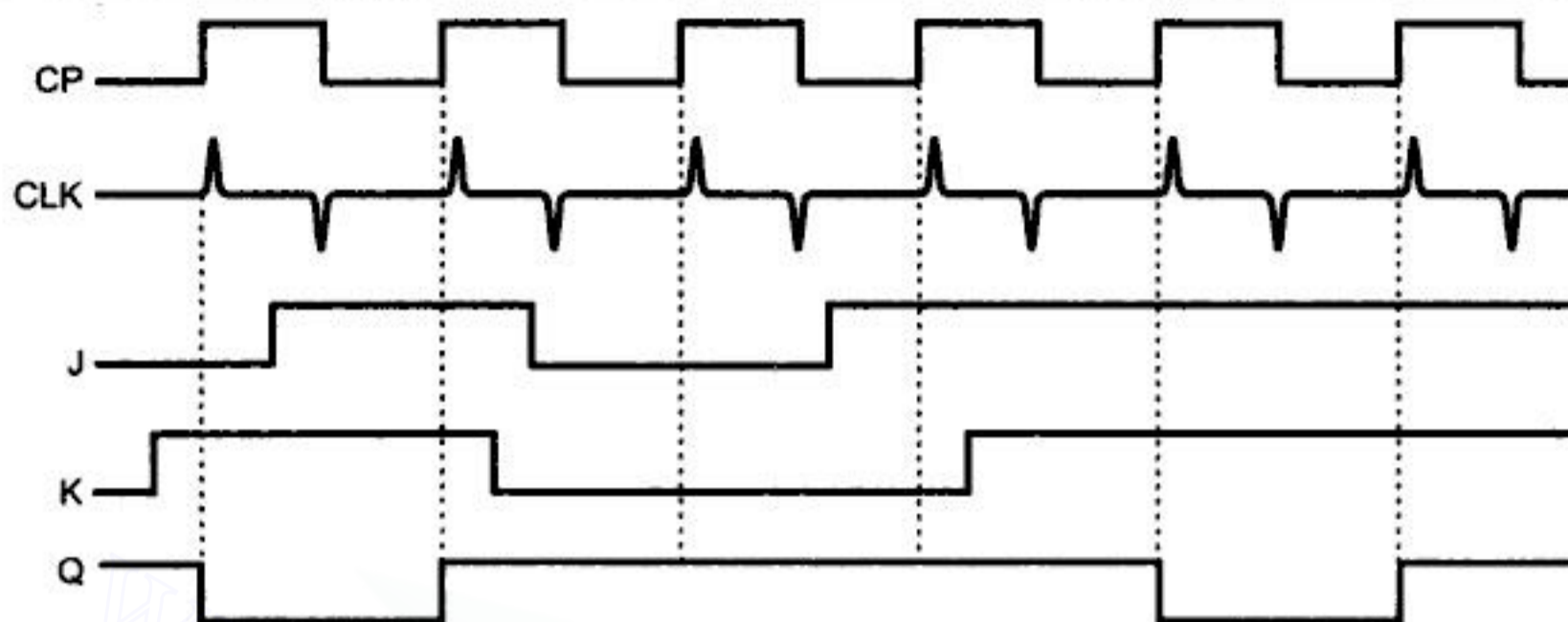


Fig. 9.19 Input and output waveforms for positive edge triggered JK flip-flop

Looking at the truth table for JK flip-flop and simplifying Q_{n+1} function by K-map we get the characteristic equation for JK flip-flop as $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$. This is illustrated in Fig. 9.20.

		JK			
		00	01	11	10
Q _n	0	0	0	1	1
	1	1	0	0	1

$$\therefore Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

Fig. 9.20 Characteristic equation for JK flip-flop

9.3.7 Clocked SR Flip-Flop

Fig. 9.21 shows the clocked SR flip-flop. The circuit is similar to SR latch except enable signal is replaced by clock pulse (CP). On the positive edge of the clock pulse. The circuit responds to the S and R inputs.

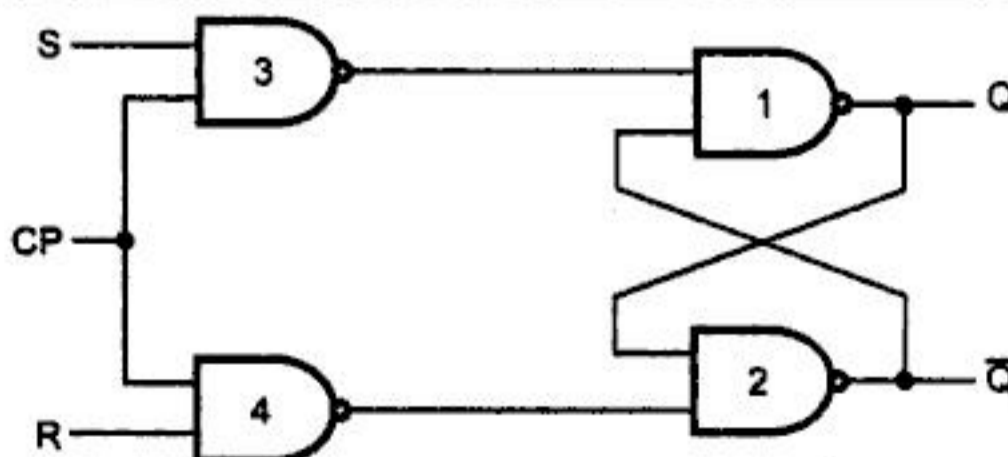


Fig. 9.21 Clocked SR flip-flop

The Fig. 9.22 shows the logic symbol and truth table of clocked SR flip-flop, and Fig. 9.23 shows input and output waveforms.

9.3.12 Asynchronous or Direct Inputs

For the flip-flops discussed so far, the RS, SR, D, JK, and T, the inputs are called synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.

Flip-flops available in IC packages sometimes provide special inputs for setting (preset) or clearing (clear) the flip-flop, asynchronously. These inputs are called asynchronous or direct inputs. These inputs are connected directly into the latch portion of the flip-flop so that they override the effect of the synchronous inputs J, K and the clock.

Refer Fig. 9.32. As shown in the Fig. 9.32, an active (high) level on the preset input immediately sets the flip-flop. (Recall that a logic 1 at any input of a NOR gate forces its output to a logic 0) and an active (high) level on the clear input immediately resets the flip-flop, without the need for a clock pulse.

A logic symbol for a JK flip-flop with preset and clear inputs is shown in the Fig. 9.31. These inputs are active high, therefore, they must both be kept low for synchronous operation.

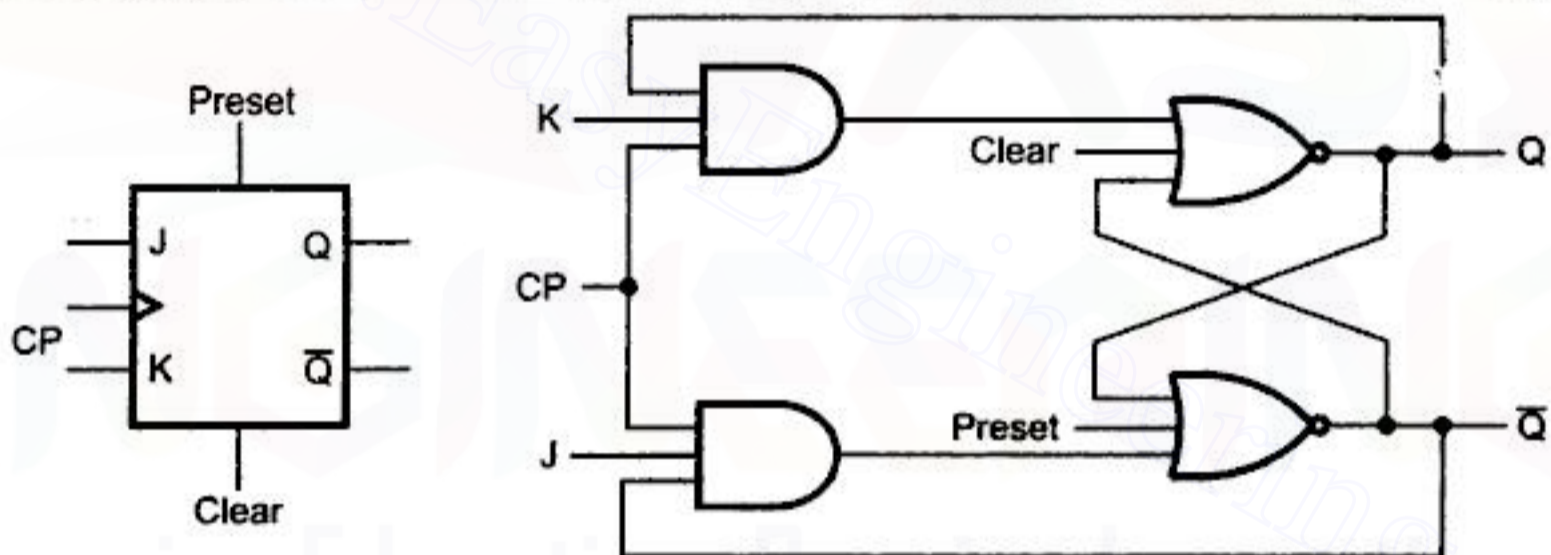


Fig. 9.31 (a) Logic symbol

(b) JK flip-flop with active high preset and clear

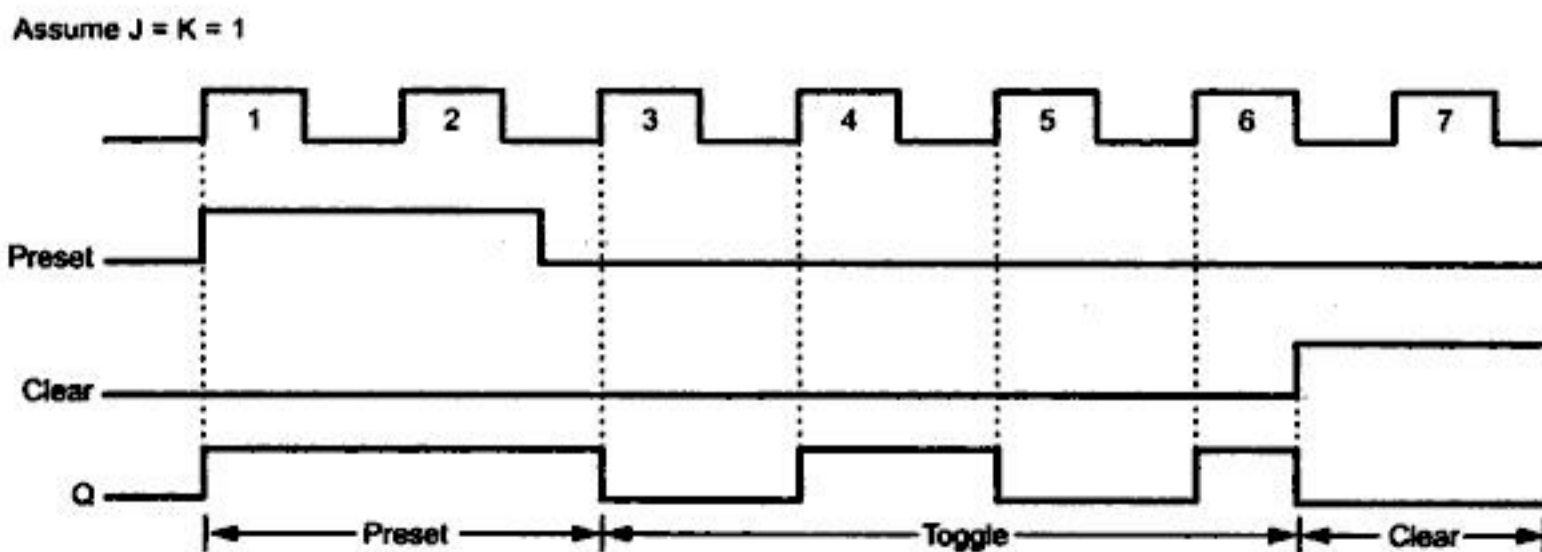


Fig. 9.32 Input and output waveforms showing operation of active high preset and clear inputs

9.4.4 JK Flip-Flop to T Flip-Flop

The excitation table for above conversion is as shown in Table 9.6 (a).

Input	Present state	Next state	Flip-flop inputs	
T	Q_n	Q_{n+1}	J_A	K_A
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

Table 9.6 (a)

K-map Simplification

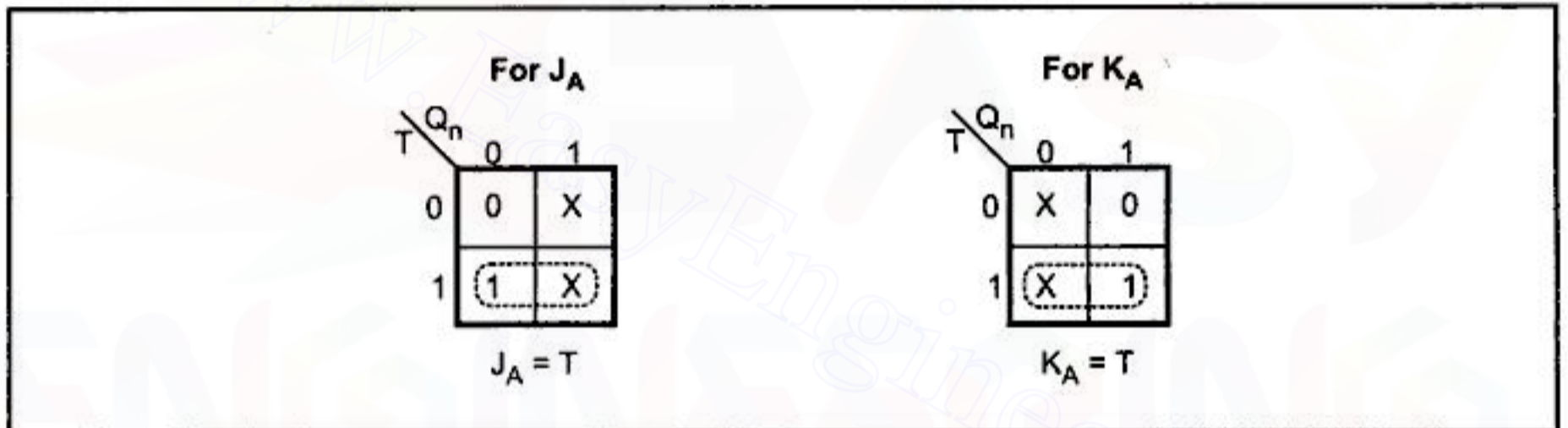


Fig. 9.41

Logic Diagram

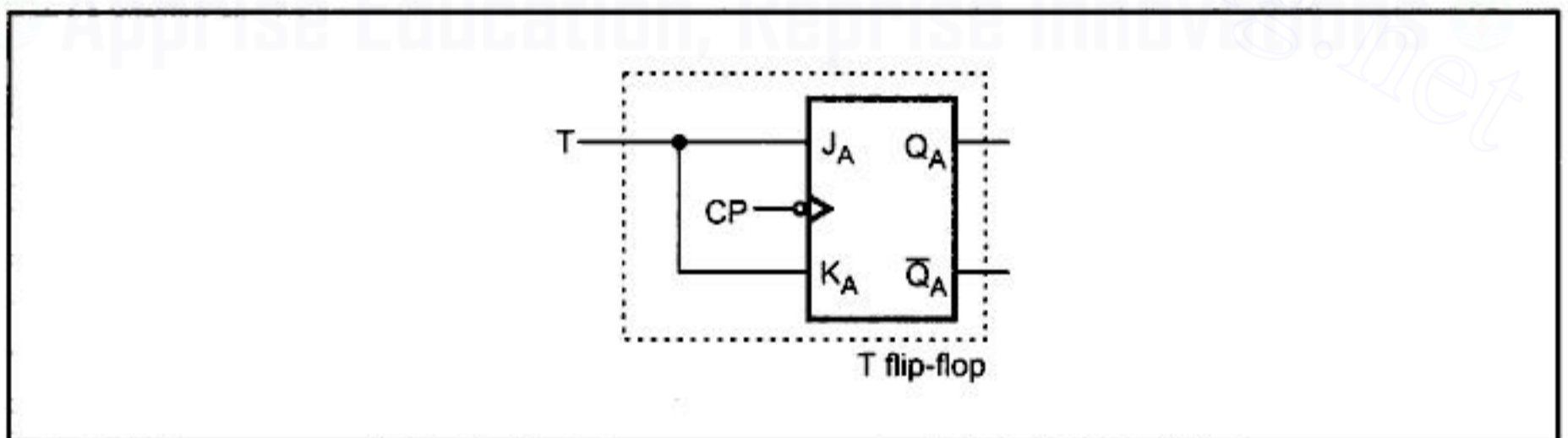


Fig. 9.42 JK to T

9.4.5 JK Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in the table 9.6 (b).

In this buffer register, there is no control over input as well as output bits. We can control input and output of the register by connecting tri-state devices at the input and output sides of register as shown in Fig. 9.50. So this register is called 'controlled buffer register'.

Here, tri-state switches are used to control the operation. When you want to store data in the register, you have to make LOAD or WR signal low to activate the tri-state buffers. When you want the data at the output, you have to make RD signal low to activate the buffers. Controlled buffer registers are commonly used for temporary storage of data within a digital system.

9.5.2 Shift Registers

The binary information (data) in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called 'shift registers'. They are very important in applications involving the storage and transfer of data in a digital system.

Fig. 9.51 gives the symbolical representation of the different types of data movement in shift register operations.

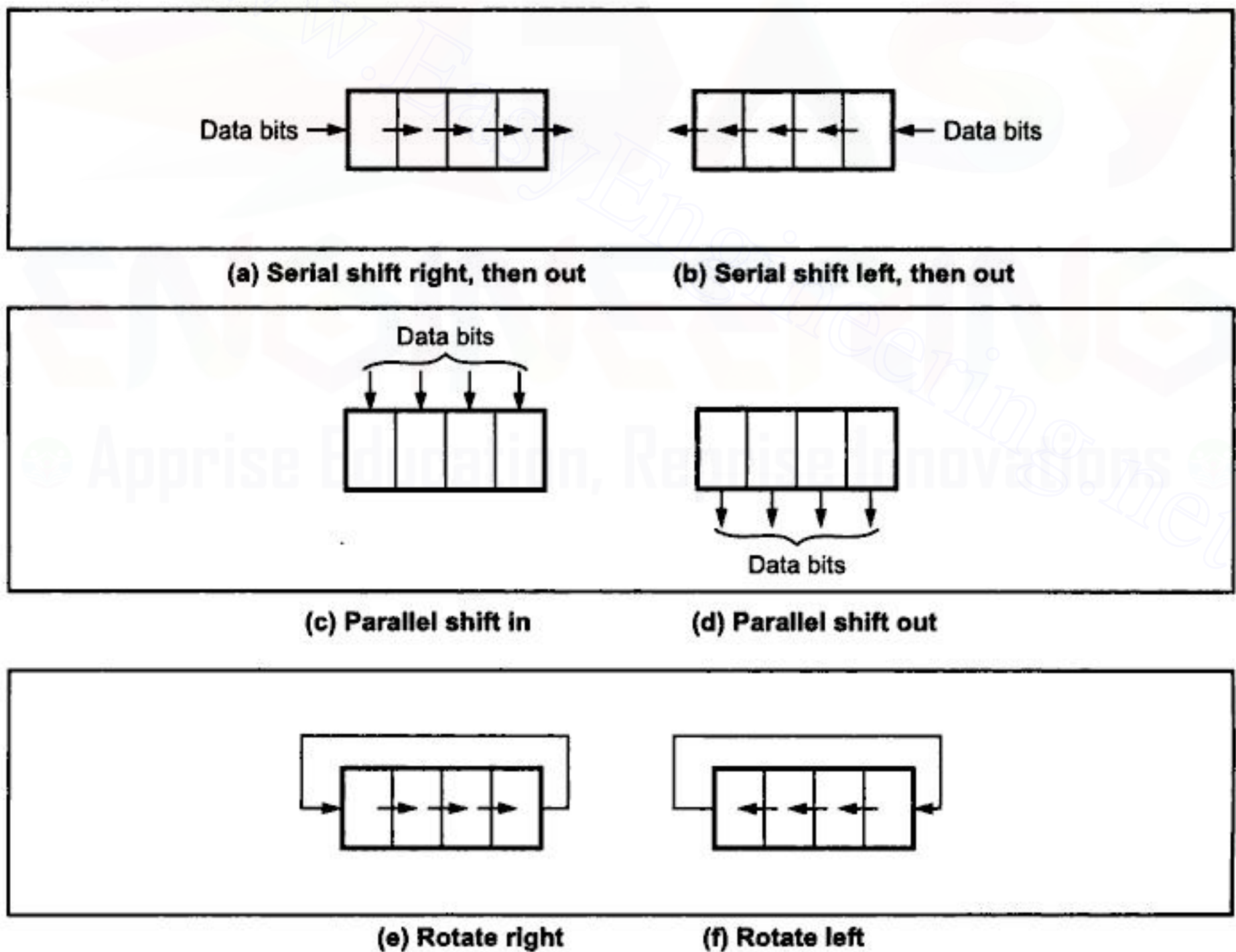


Fig. 9.51 Basic data movement in registers

b) When the next falling clock edge hits, the Q_1 flip-flop sets and the register contents become,

$$Q_A Q_B Q_C Q_D = 1100$$

c) The third falling clock edge results in,

$$Q_A Q_B Q_C Q_D = 1110$$

d) the fourth falling clock edge gives,

$$Q_A Q_B Q_C Q_D = 1111$$

Fig. 9.55 (See Fig. on previous page) illustrates the input and output condition of each flip-flop stage upon application of each clock pulse.

2) Serial In Parallel Out Shift Register

In this case, the data bits are entered into the register in the same manner as discussed in the last section, i.e. serially. But the output is taken in parallel. Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously, instead of on a bit-by-bit basis as with the serial output. (Shown in Fig. 9.56).

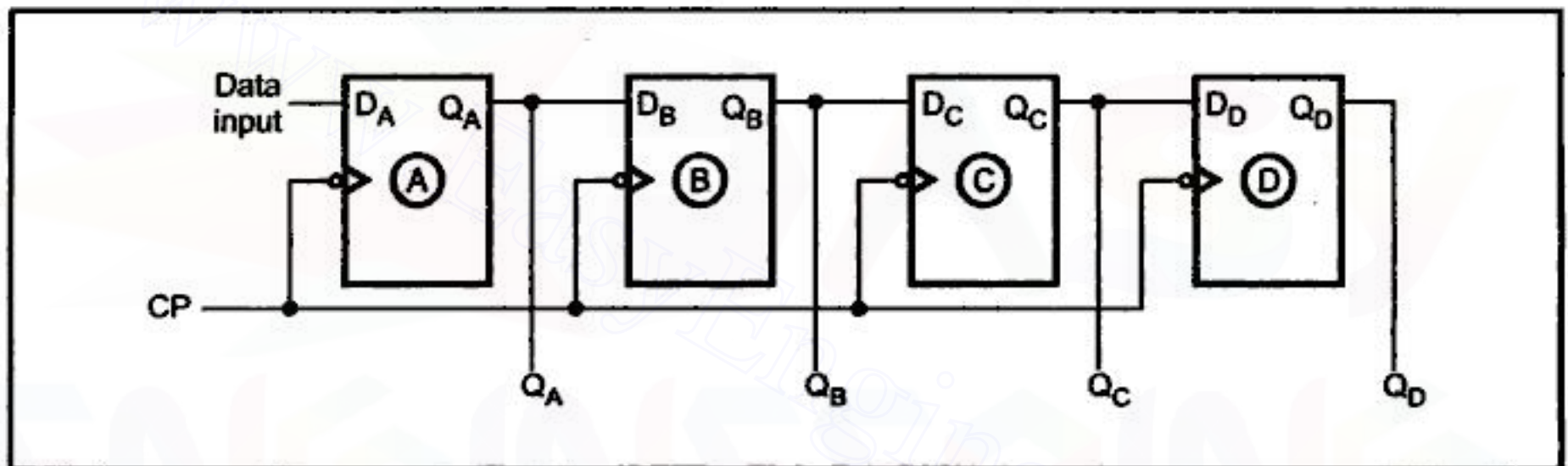


Fig. 9.56 A serial in parallel out shift register

3) Parallel In Serial Out Shift Register

In this type, the bits are entered in parallel i.e. simultaneously into their respective stages on parallel lines.

Fig. 9.57 illustrates a four-bit parallel in serial out register. There are four input lines X_A, X_B, X_C, X_D for entering data in parallel into the register. SHIFT/LOAD is the control input which allows shift or loading data operation of the register. When SHIFT/LOAD is low, gates G_1, G_2, G_3 are enabled, allowing each input data bit to be applied to D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with $D = 1$ will SET and those with $D = 0$ will RESET. Thus all four bits are stored simultaneously.

When SHIFT/LOAD is high, gates G_1, G_2, G_3 are disabled and gates G_4, G_5, G_6 are enabled. This allows the data bits to shift left from one stage to the next. The OR gates at the D-inputs of the flip-flops allow either the parallel data entry operation or shift operation, depending on which AND gates are enabled by the level on the SHIFT/LOAD input.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

9.6.5 Pseudo-Random Binary Sequence (PRBS) Generator

Another important application of shift register is a pseudo-random binary sequence generator. Here, suitable feed back is used to generate pseudo-random sequence. The term random here means that the outputs do not cycle through a normal binary count sequence. The term pseudo here refers to the fact that the sequence is not truly random because it does cycle through all possible combinations once every $2^n - 1$ clock cycles, where n represents the number of shift register stages (number of flip-flops).

Fig. 9.66 shows the 4-bit random sequence generator. It has four stages of shift registers and has exclusive-OR feedback from stages C and D such that the input to the first stage

$$J_A = C \oplus D.$$

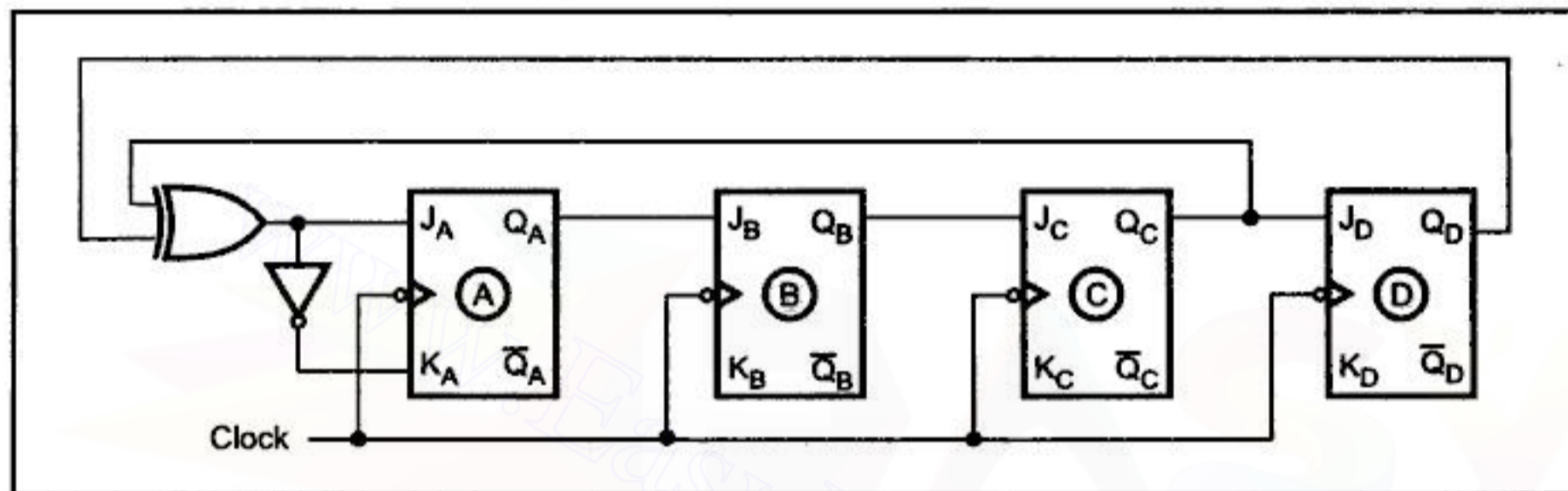


Fig. 9.66 A 4-bit pseudo-random sequence generator

To go through the sequence of states for the register, it is assumed initially that the shift register is in the state $D = 0, C = 0, B = 0$ and $A = 1$, in which case $J_A = 0 \oplus 0$, and on the rising edge of the next clock pulse the register enters the state $D = 0, C = 0, B = 1$ and $A = 0$.

The complete sequence of states for the shift register is as shown in the Table 9.14. The value of the feedback function for each stage is tabulated in the column f.

State	D	C	B	A	F
S_1	0	0	0	1	0
S_2	0	0	1	0	0
S_3	0	1	0	0	1
S_4	1	0	0	1	1
S_5	0	0	1	1	0
S_6	0	1	1	0	1
S_7	1	1	0	1	0
S_8	1	0	1	0	1

C HIGH, and flip-flop C toggles on the following clock pulse. At all other times, the J and K inputs of flip-flop C are held LOW by the AND gate output, and flip-flop does not change state.

A Four-Bit Synchronous Binary Counter

Fig. 9.74 shows logic diagram and timing diagram for 4-bit synchronous binary counter. As counter is implemented with negative edge triggered flip-flops, the transitions occur at the negative edge of the clock pulse. In this circuit, first three flip-flops work same as 3-bit counter discussed previously.

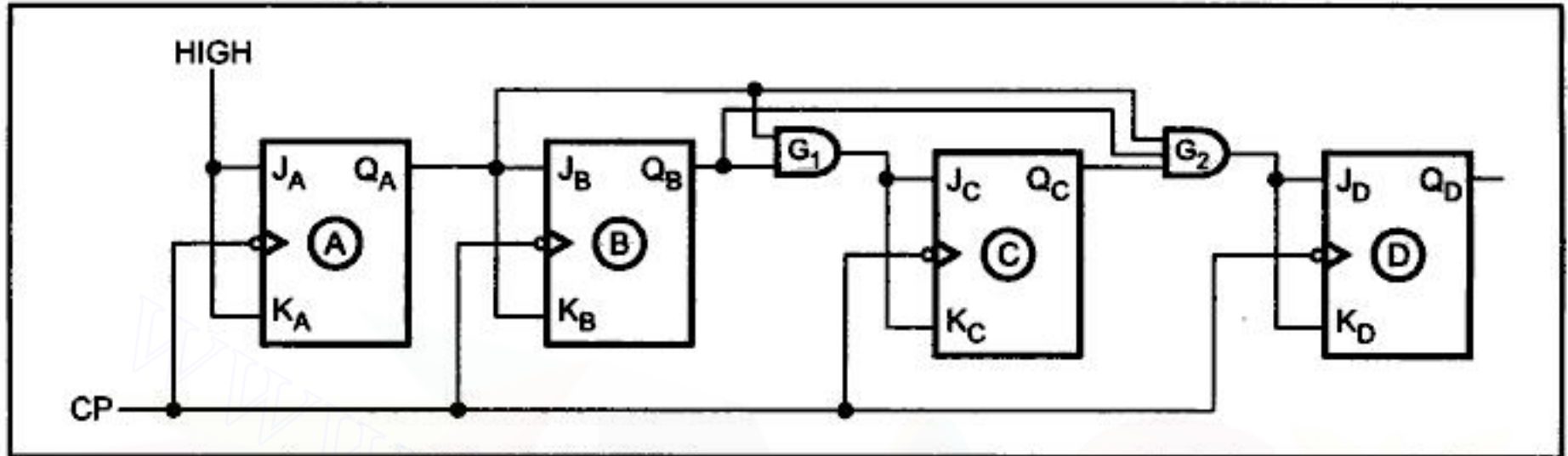


Fig. 9.74 (a)

For the fourth stage, flip-flop has to change the state when $Q_A = Q_B = Q_C = 1$. This condition is decoded by 3-input AND gate G_2 . Therefore, when $Q_A = Q_B = Q_C = 1$, flip-flop D toggles and for all other times it is in no change condition.

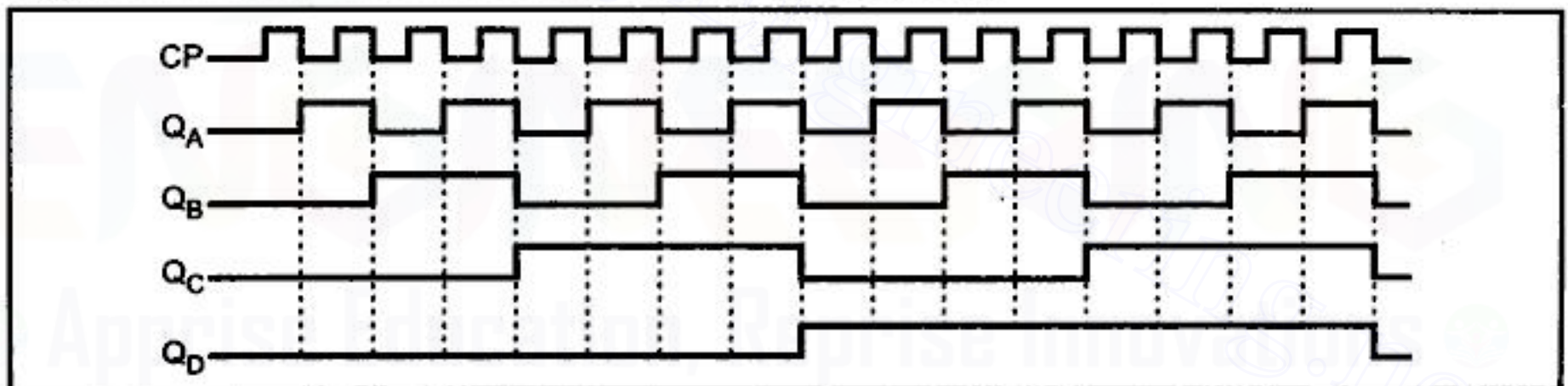


Fig. 9.74 (b) A four-bit synchronous binary counter and timing diagram

Ex. : 9.9 Determine f_{max} for the 4-bit synchronous counter if t_{pd} for each flip-flop is 50 ns and t_{pd} for each AND gate is 20 ns. Compare this with f_{max} for a MOD - 16 ripple counter.

Sol.: For a synchronous counter the total delay that must be allowed between input clock pulses is equal to flip-flop t_{pd} + AND gate t_{pd} . Thus $T_{clock} \geq 50 + 20 = 70$ ns, and so the counter has

$$f_{max} = \frac{1}{70 \text{ ns}} = 14.3 \text{ MHz}$$

We know that MOD-16 ripple counter used four flip-flops. With flip-flop $t_{pd} = 50$ ns, the f_{max} for ripple counter can be given as

$$f_{max \text{ (ripple)}} = \frac{1}{4 \times 50 \text{ nS}} = 5 \text{ MHz}$$

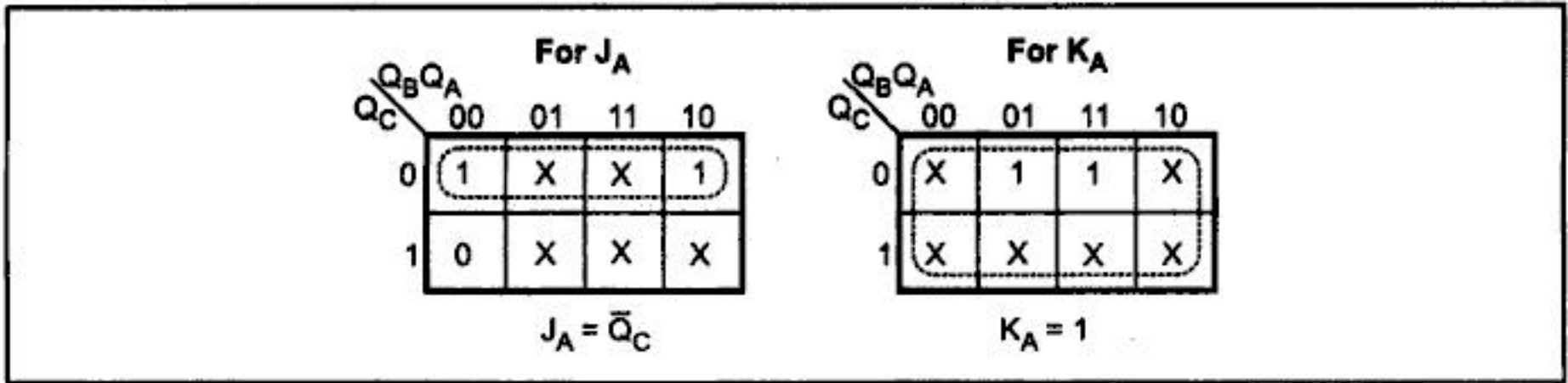


Fig. 9.75

Logic Diagram

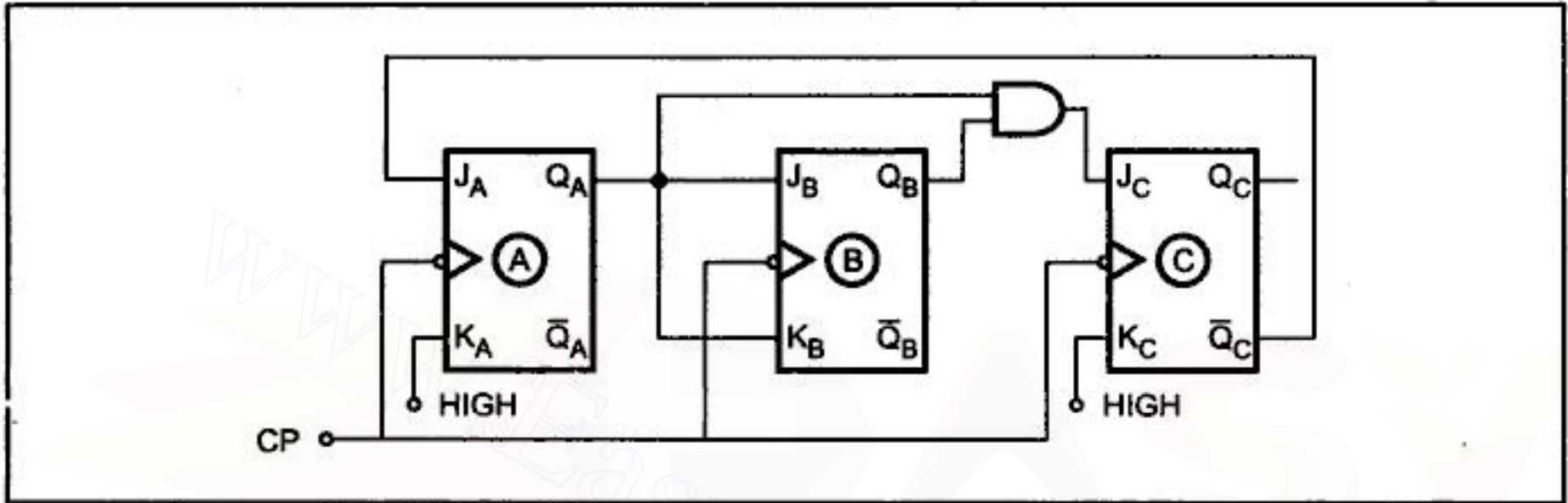


Fig. 9.76 MOD-5 synchronous counter

Timing Diagram

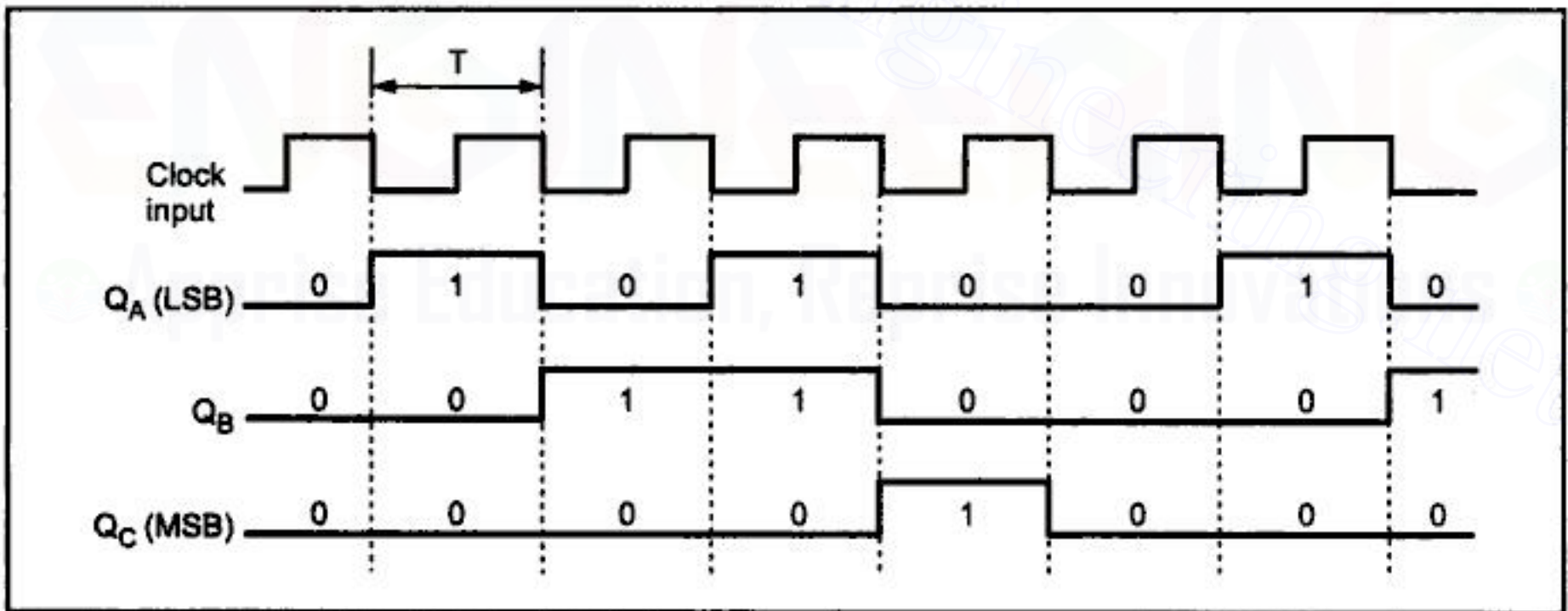


Fig. 9.77 Timing diagram

Ex. 9.11 : Design divide by 6 counter using T-flip-flop. Write state table and reduce the expression using K-map.

Sol. : For designing mod 6 counter using the formula

$$2^n \geq N$$

Here $N = 6$

$\therefore n = 3$ i.e. 3 flip-flops are required.

9.13 UP / DOWN Synchronous Counters

An up/down counter is one that is capable of progressing in increasing order or decreasing order through a certain sequence. An up/down counter is also called bidirectional counter. Usually up/down operation of the counter is controlled by UP/DOWN signal. When this signal is HIGH, counter goes through UP sequence, i.e. 0, 1, 2, 3,, n. When UP/DOWN signal is LOW counter follows reverse sequence i.e. n, n-1, n-2,, 1, 0. For 3-bit counters these sequences are : 0, 1, 2, 3, 4, 5, 6, 7 for up operation and 7, 6, 5, 4, 3, 2, 1, 0 for DOWN operation. This is illustrated in Table 9.28. The arrows in the Table 9.28 indicate the state-to-state movement of the counter for both its UP and its DOWN modes of operation.

State Table



CP	UP	Q _c	Q _b	Q _a	DOWN
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

Table 9.28

Let us design the 3-bit UP/ DOWN synchronous counter, using T flip-flops. Table 9.29 shows the excitation table for 3-bit UP/ DOWN synchronous counter.

Excitation Table

Input UP/ <u>DOWN</u> (UD)	Present State			Next State			Flip-flop Inputs		
	Q _C	Q _B	Q _A	Q _{C+1}	Q _{B+1}	Q _{A+1}	T _C	T _B	T _A
0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0	1	1
0	0	1	1	0	1	0	0	0	1
0	1	0	0	0	1	1	1	1	1

Ex. 9.14 : Design a divide-by-96 counter using 7490 ICs.

Sol : IC 7490 is a decade counter. When two such ICs are cascaded, it becomes a divide by 100 counter. To get a divide-by-96 counter, the counter is reset as soon as it becomes 1001 0110. The diagram is shown in Fig. 9.93.

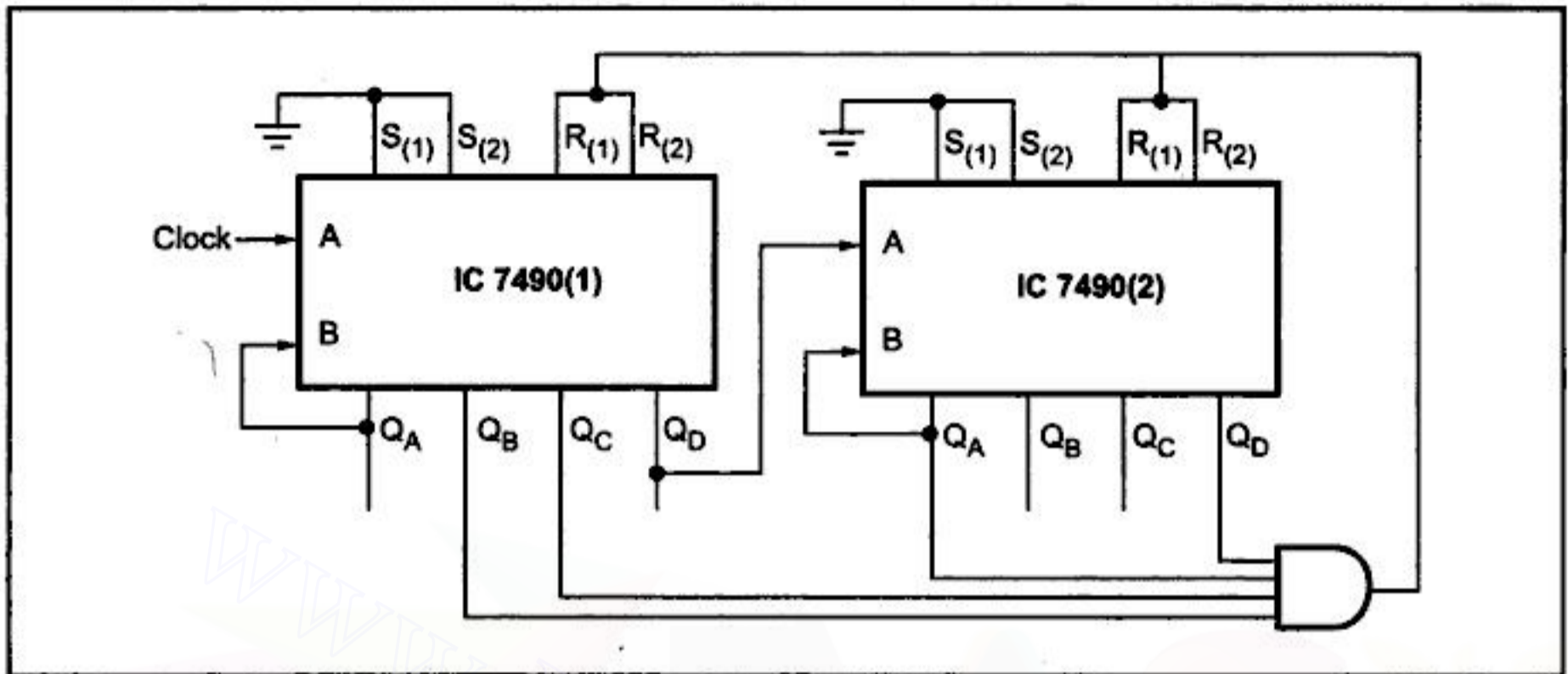


Fig. 9.93 Divide by 96 counter

9.14.2 IC 7492/93 (4-bit Ripple Counters)

The 7492 and 7493 are high speed 4-bit ripple type counters partitioned into two sections. Each counter has a divide-by-two section and either a divide-by-six (7492) or divide-by-eight (7493) section which are triggered by a HIGH to LOW transition on the clock inputs. Each section can be used separately or tied together to form divide-by-twelve or divide-by-sixteen counters.

The 7492 is 4-bit divide by twelve counter and 7493 is 4-bit binary counter. Each device consists of four master slave flip-flops which are internally connected to provide a divide-by-two section. Since the output from the divide-by-two section is not internally connected to the succeeding stages, the devices may be operated in various counting modes.

Connection Diagram

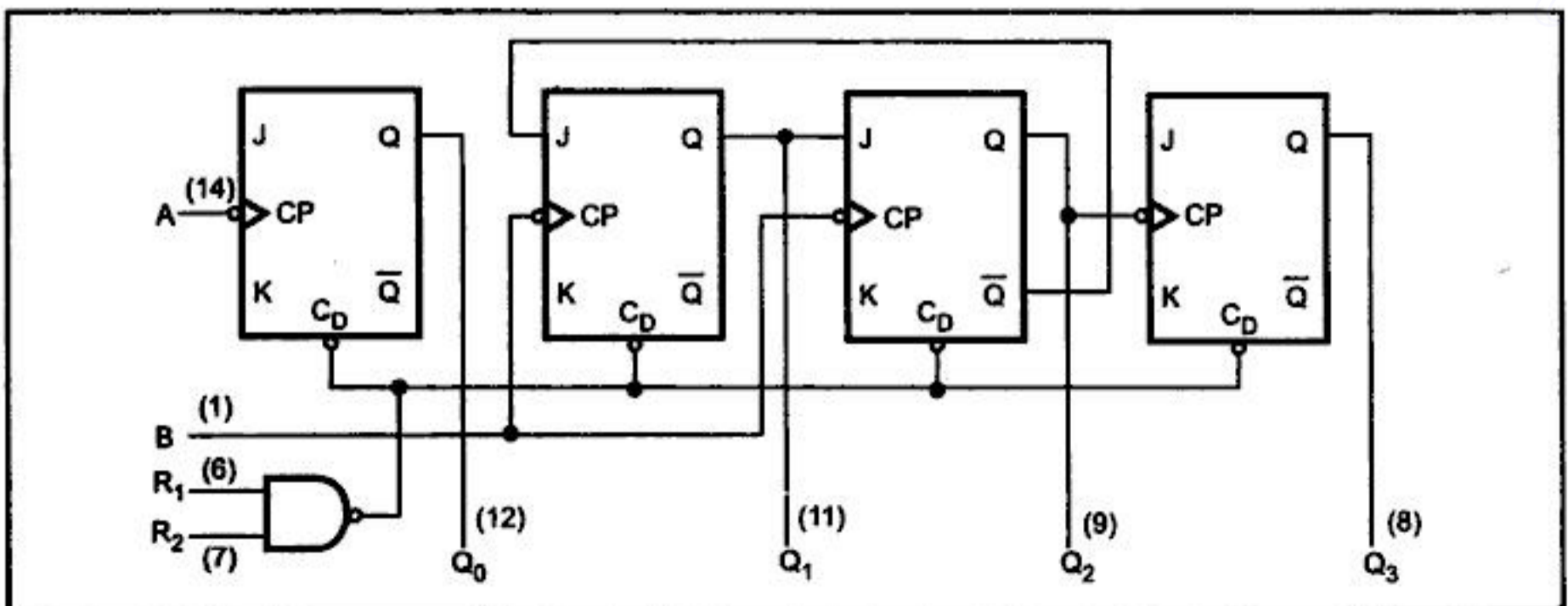


Fig. 9.94 (a)

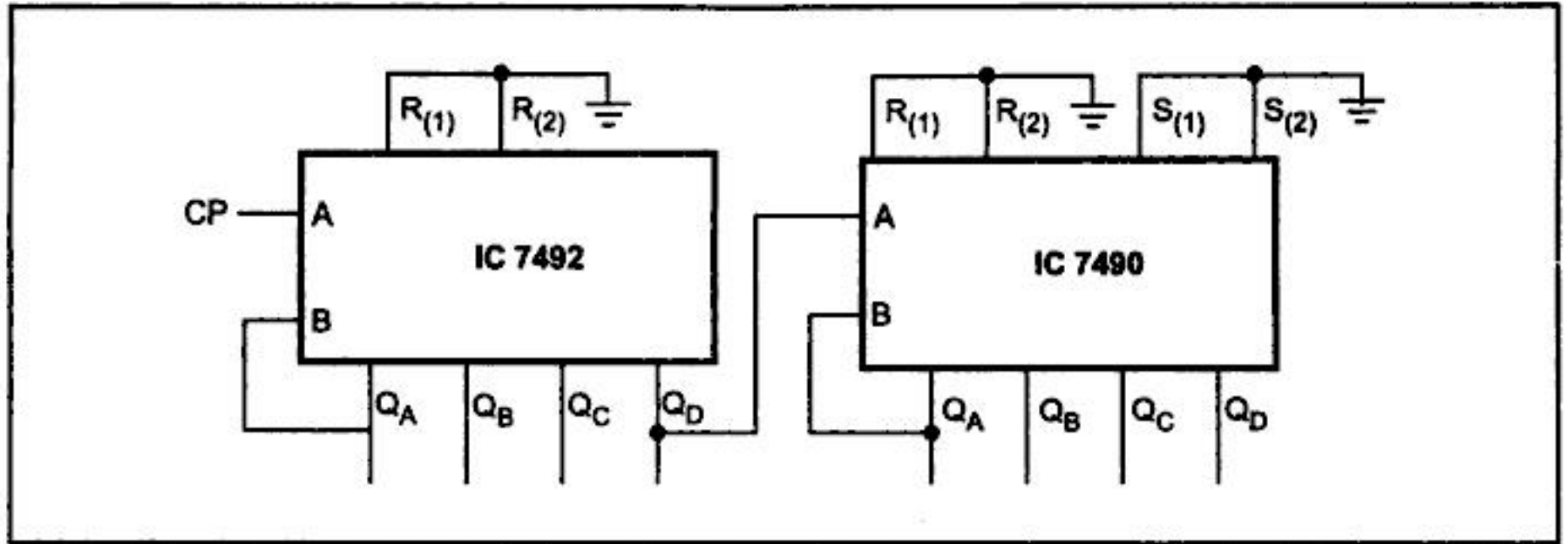


Fig. 9.99 Modulo 120 counter

9.14.3 IC 74161 (4-bit synchronous Binary Counter)

IC 74161 is a synchronous presettable, 4-bit binary counter. It has built-in look-ahead for fast counting Fig. 9.100 shows connection diagram for IC 74161.

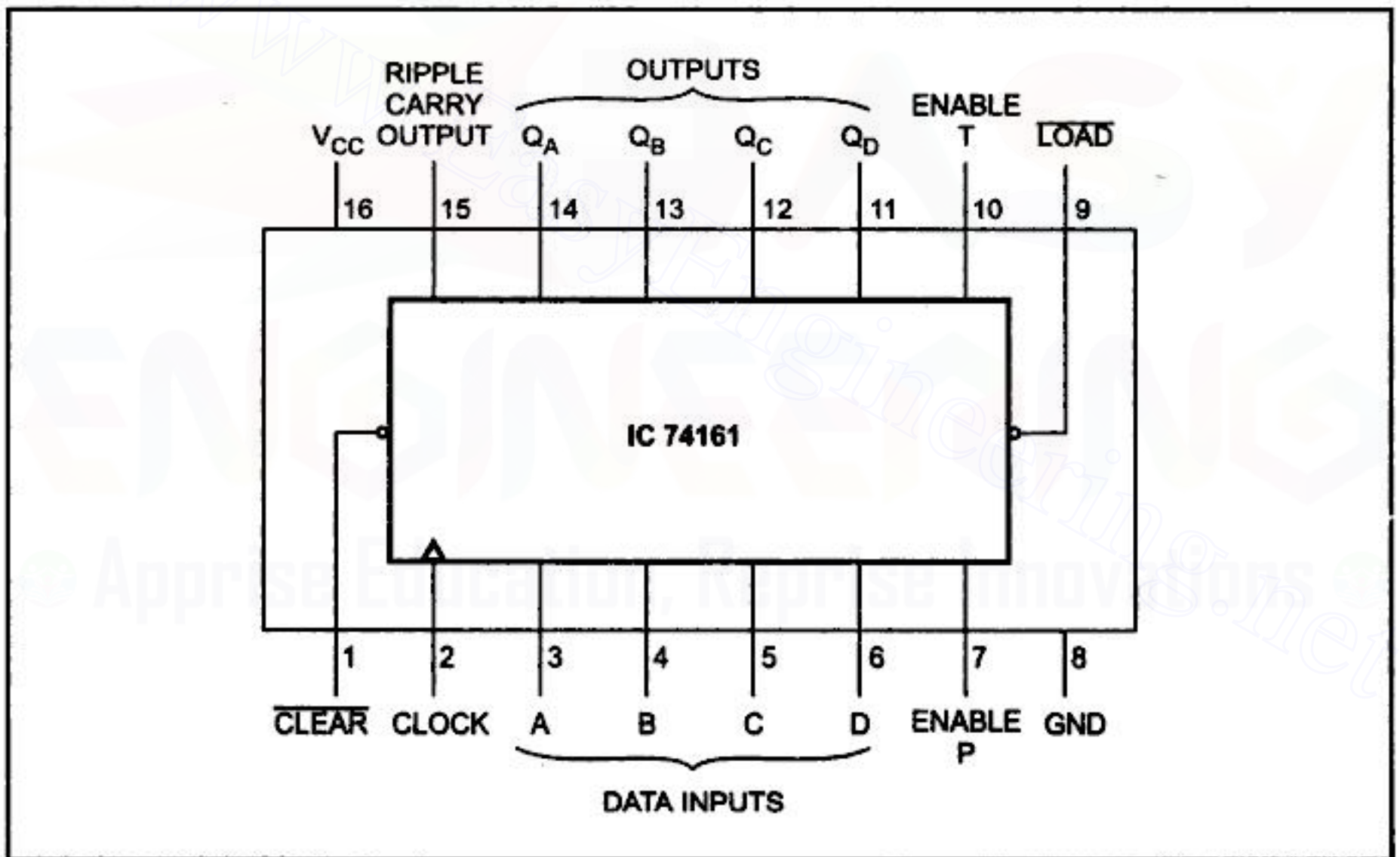


Fig. 9.100 Connection diagram for IC 74161

Pin Functions : *Data Inputs (A, B, C, D) :*

These inputs are used to preset counter to the desired count.

Outputs (QA, QB, QC, QD) : Outputs of the counter.

LOAD : A LOW level at the $\overline{\text{LOAD}}$ input disables the counting action and causes data at the A, B, C and D inputs to be loaded into the counter on the positive going

Ex. 9.22 : Design a MOD-11 counter using IC 74191.

Sol.: IC 74191 is a 4-bit counter. Thus it is MOD-16 counter. However, we require MOD-11 counter. The difference between 16 and 11 is 5. Hence 5 steps must be skipped from the full modulus sequence. This can be achieved by presetting counter to value 5. Each time when counter recycles it starts counting from 5 up to 16 on each full cycle. Therefore, each full cycle of the counter consists of 11 states.

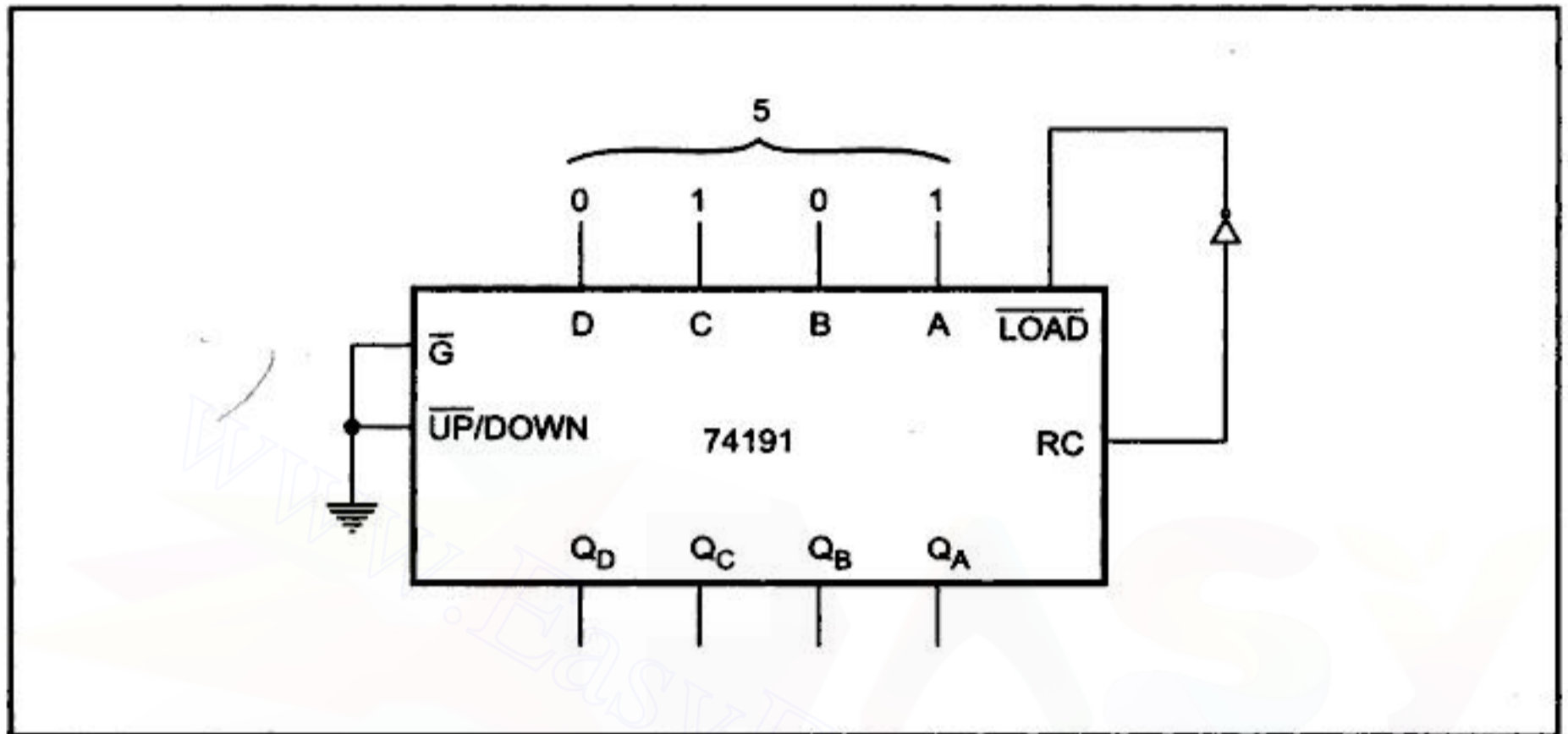


Fig. 9.108 MOD-11 counter using 74191

9.14.5 IC 74192/74193 (UP/DOWN-BCD/BINARY Counters)

The 74192 is an UP/DOWN BCD decade counter and the 74193 is an UP/DOWN modulo-16 binary counter. Separate Count Up and Count Down Clocks are used and in either counting mode the circuits operate synchronously. The outputs change state in synchronous manner with the LOW-to-HIGH transitions on the clock inputs.

Separate Terminal Count Up and Terminal Count Down outputs are provided which are used as the clocks for a subsequent stages without extra logic, thus simplifying multistage counter designs. Individual preset inputs allow the circuits to be used as programmable counters. Both the Parallel Load (PL) and the Master Reset (MR) inputs asynchronously override the clocks.

Features :

- Low Power ... 95 mW Typical Dissipation
- High Speed ...40 MHz Typical Count Frequency
- Synchronous Counting
- Asynchronous Master Reset and Parallel Load
- Individual Preset Inputs
- Cascading Circuitry Internally Provided
- Input Clamp Diodes Limit High Speed Termination Effects

L : Low Voltage Level

H : High Voltage Level

X : Don't Care

Ex. 9.23 : Design a divide-by-7 DOWN counter using 74192 IC. Make use of the borrow output.

Sol: For the DOWN counter, the clock pulses are applied at CP_D input. When the output becomes 0, the counter is loaded with preset inputs 0111 and the states will be : 0111, 0110, 0101, 0100, 0011, 0010, and 0001. The circuit is shown in Fig. 9.113.

Ex. 9.24 : Design a mod-12 Up counter using 74193 IC.

Sol: This can be done in two ways :

1. Connect the circuit shown in Fig. 9.113. The Q_2 and Q_3 outputs are ANDed and are connected to clear input (with $PL = 1$). As soon as count becomes 1100, the counter is cleared.

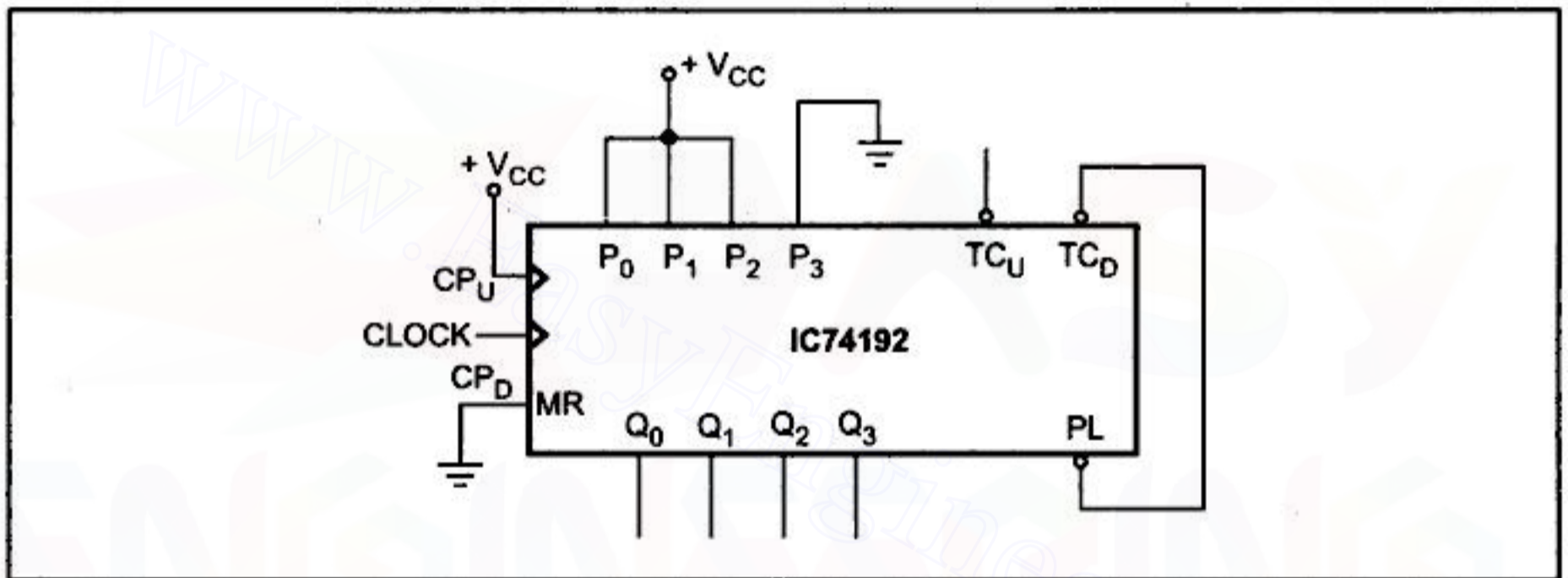


Fig. 9.113 Divide by-7 down counter

2. Connect the circuit shown in Fig. 9.114. The Q_2 and Q_3 outputs are NANDed and are connected to load input (with $MR = 1$, $P_0 = P_1 = P_2 = P_3 = 0$). As soon as the count reaches 1100, the counter is loaded with P inputs.

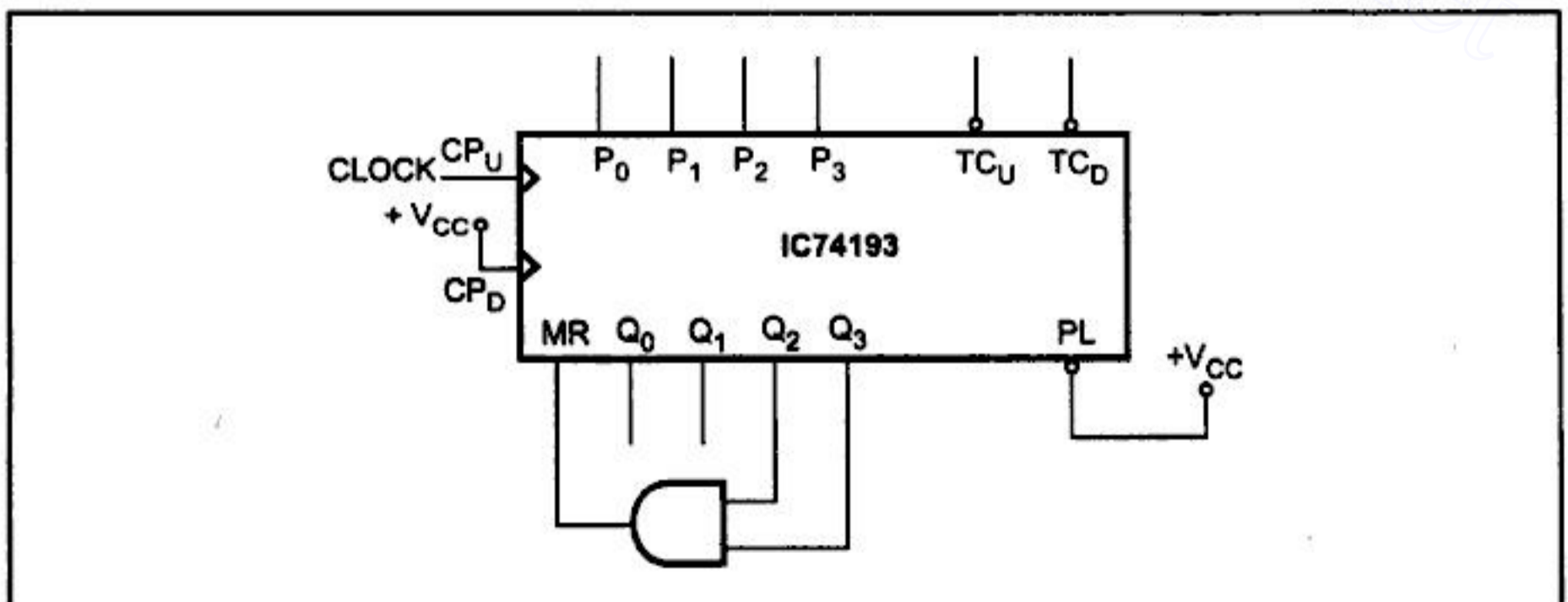


Fig. 9.114 Mod-12 Up counter

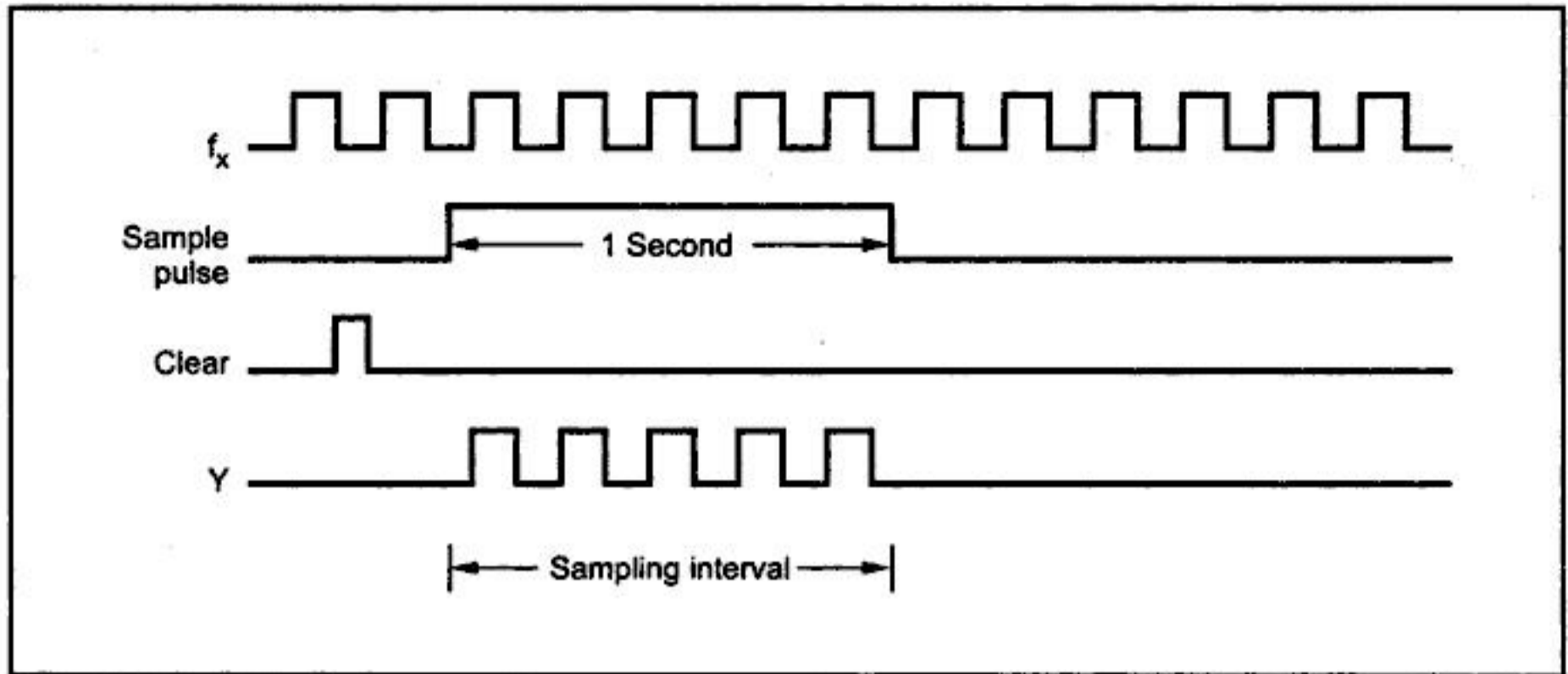


Fig. 9.121 (b) Timing waveforms for frequency counter

The Fig. 9.122 shows the complete diagram of frequency counter which can measure frequency upto 9999 Hz.

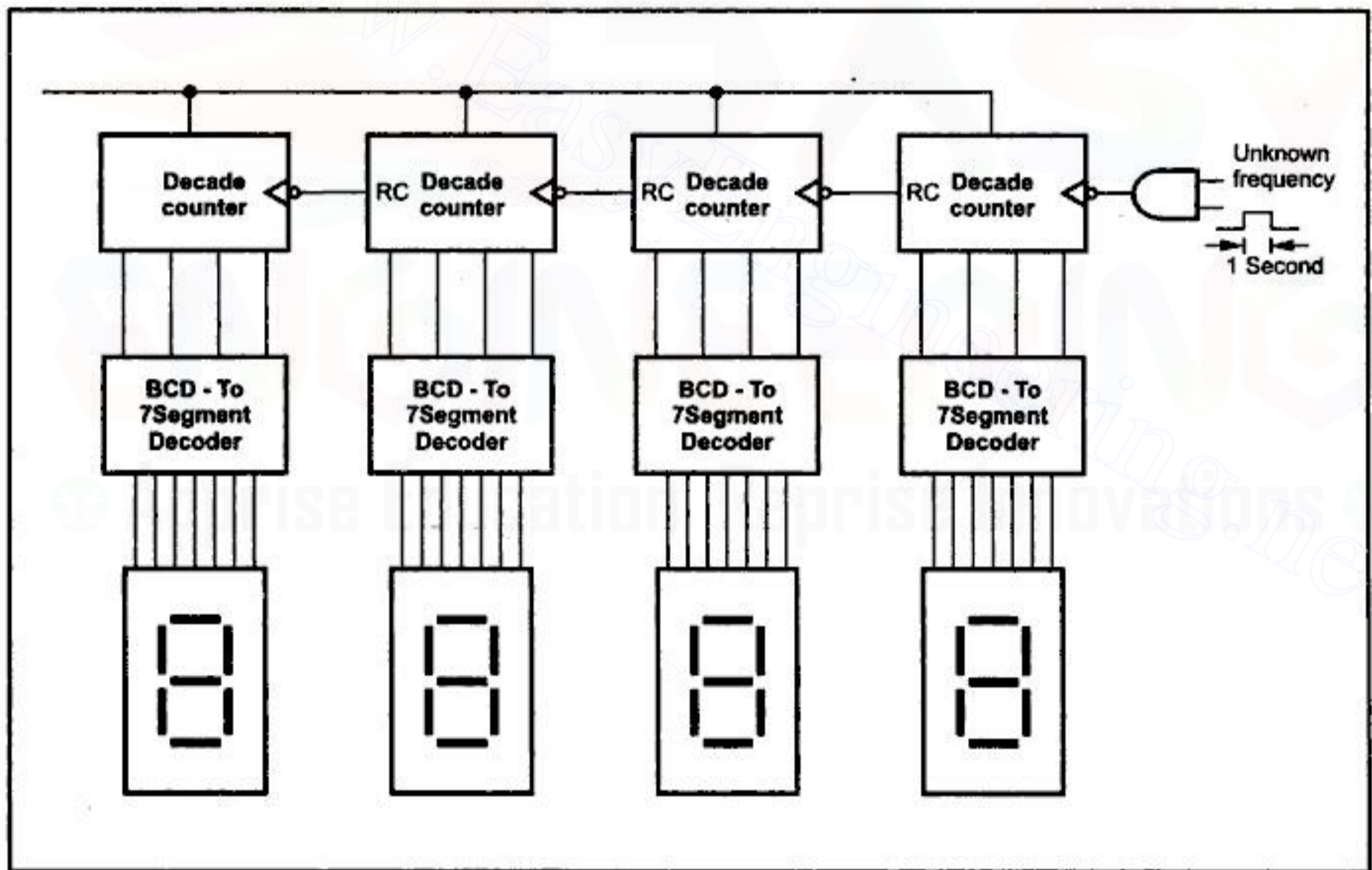


Fig. 9.122 Block diagram of frequency counter

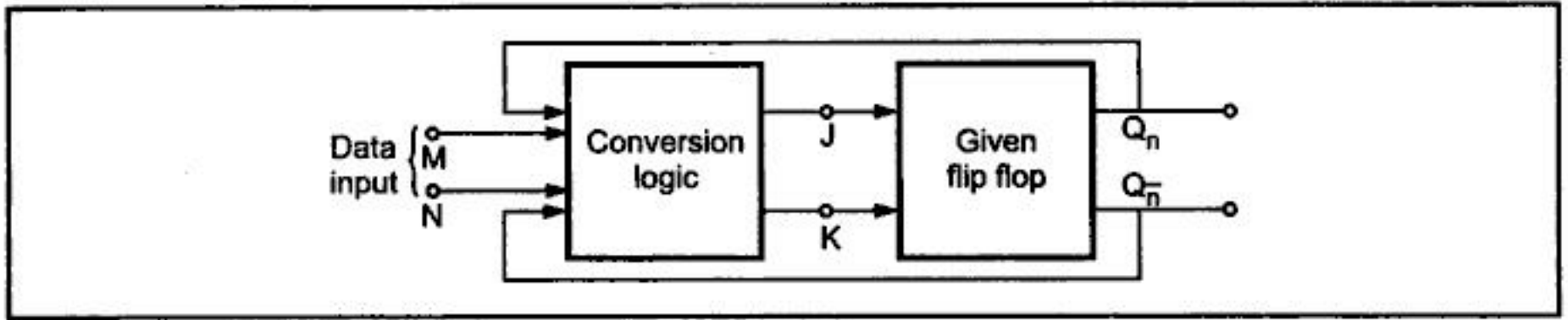


Fig. 9.127

Here, given flip-flop is JK. Hence

M	N	Q_n	Q_{n+1}	J	K
0	0	0	1	1	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	1	X	0
1	0	0	1	1	X
1	0	1	0	X	1
1	1	0	0	0	X
1	1	1	0	X	1

Table 9.40

By using K-map we will find J and K.

K-map Simplification

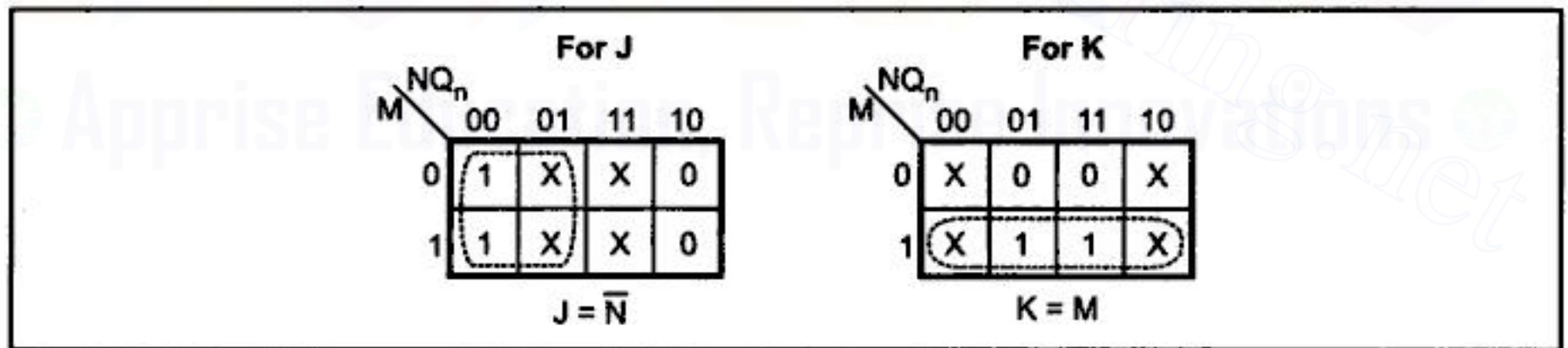


Fig. 9.128

Logic Diagram

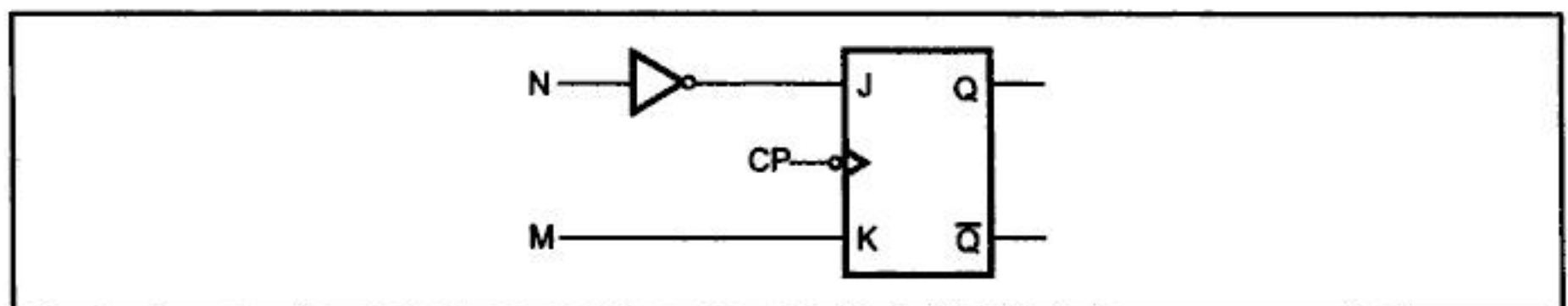


Fig. 9.129

Ex. 9.37 : Design a 4-bit counter that counts either in binary or BCD depending on control line *MODE*. When *MODE* = 0, the counter is to count in binary, and when *MODE* = 1, the counter is to count in BCD. Select any flip-flops for implementation. Draw the logic diagram for realization.

Sol. : Table 9.42 shows the excitation table for BCD/Binary counter.

Input M	Present State				Next State				Flip-flop Inputs							
	Q _A	Q _B	Q _C	Q _D	Q _{A+1}	Q _{B+1}	Q _{C+1}	Q _{D+1}	J _A	K _A	J _B	K _B	J _C	K _C	J _D	K _D
0	0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
0	1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
0	1	0	0	1	1	0	1	0	X	0	0	X	1	X	X	1
0	1	0	1	0	1	0	1	1	X	0	0	X	X	0	1	X
0	1	0	1	1	1	1	0	0	X	0	1	X	X	1	X	1
0	1	1	0	0	1	1	0	1	X	0	X	0	0	X	1	X
0	1	1	0	1	1	1	1	0	X	0	X	0	1	X	X	1
0	1	1	1	0	1	1	1	1	X	0	X	0	X	0	1	X
0	1	1	1	1	0	0	0	0	X	1	X	1	X	1	X	1
1	0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
1	0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
1	0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
1	0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
1	0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
1	0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
1	0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
1	0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

Table 9.42

Ex. 9.39 : Draw and explain the logic diagram for serial adder.

(Dec-2002)

Sol. : Fig. 9.136 shows the logic diagram for serial adder

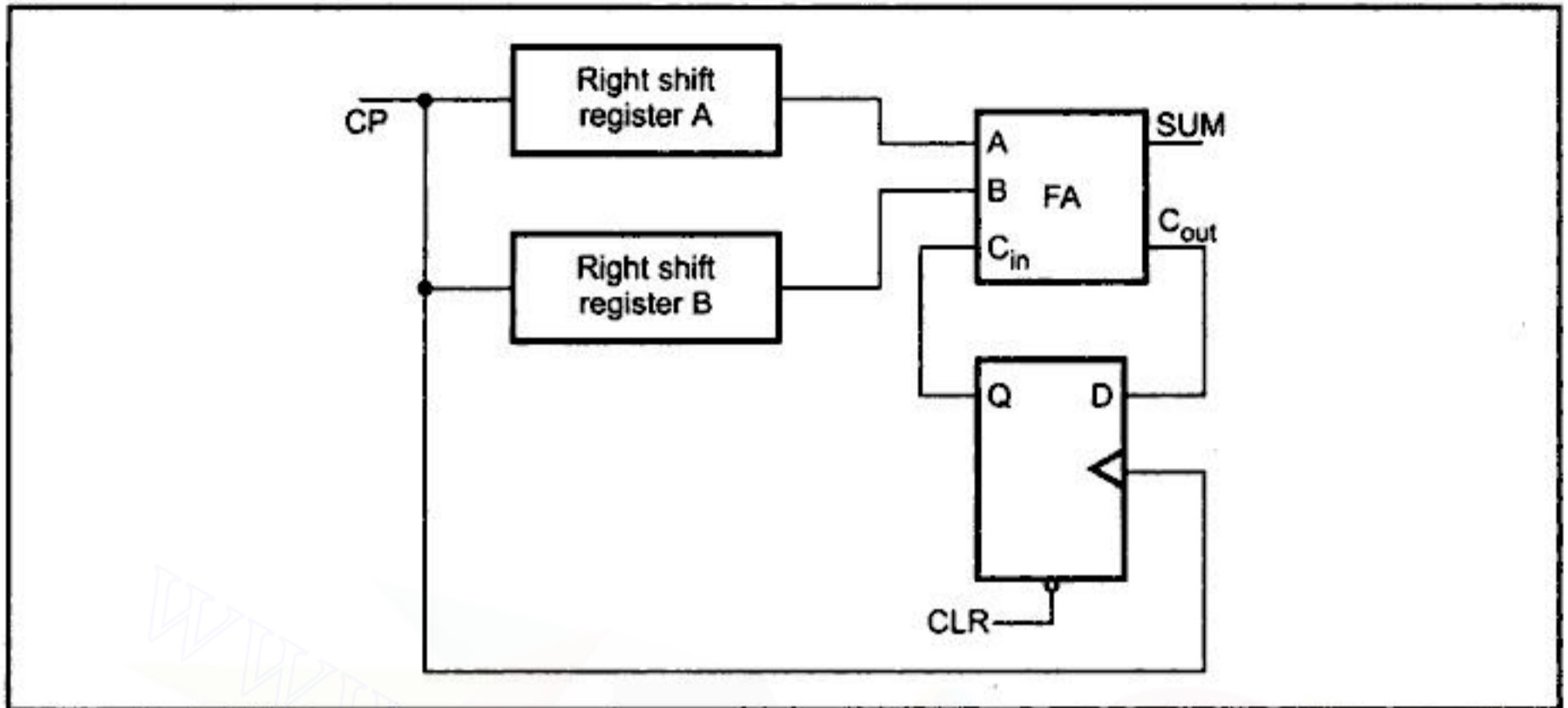


Fig. 9.136

Using the logic diagram shown in Fig. 9.136 we can add numbers stored in the right shift registers A and B, serially. The full adder is used to perform bit by bit addition and D-flip-flop is used to store the carry output generated after addition. This carry is used as carry input for the next addition. Initially, the D-flip-flop is cleared and addition starts with the least significant bits of both register. After each clock pulse data within the right shift registers are shifted right 1-bit and we get bits from next digit and carry of previous addition as new inputs for the full adder.

Ex. 9.40 : With the help of two shift Registers design a 4-bit Serial Adder.

(Dec-99)

Sol. : Two shift registers are available to store the binary numbers to be added serially. The serial outputs from the registers are x and y. S gives the sum and carry after addition is stored in the flip-flop. Therefore Q output of flip-flop gives carry. Excitation table for a serial Adder.

Present state	Inputs		Next state	Output	Flip-flop inputs	
	x	y			Q	S
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1

- suitable circuit to interface a common cathode display to the IC 7490. Draw a complete circuit diagram for one digit only. (May-96)
30. Draw the logic diagram of clocked master-slave J-K flip-flop using NAND gates only. Verify its operation with the help of truth table. (May-96)
31. "Asynchronous counters are faster than synchronous counters". Comment. (May-96)
32. Write a short note on Combinational and Sequential Logic (May-96)
33. Draw the schematic diagram of a 3-bit asynchronous up-down counter and explain with the help of waveform. (Dec-96)
34. Draw the logic diagram of master-slave J-K flip-flop using NAND gates only. Write down the truth table and explain it. (Dec-96, Dec-98)
35. What is race around condition ? How to eliminate it ? (Dec-96, Dec-98)
36. Compare and contrast Sequential circuit and combinational circuit; (Dec-96, Dec-97, Dec-98)
37. Write a short note on Johnson counter (Dec-96)
38. Draw a logic diagram of JK F/F using only NAND gates. Explain race-around condition. How it is eliminated ? (May-97)
39. Convert JK F/F into D-F/F and T-F/F. (Dec-97)
40. Draw logic diagram and waveform of twisted ring counter. (Dec-97)
41. Explain with logic diagram functions of the IC7495 (Dec-97)
42. Explain with logic diagram functions of the IC74191 (Dec-97)
43. Explain T flip-flop (May-98)
44. Design a modulo-13 ripple counter using T flip-flops. The design must include state diagram, truth table, K-map and logic diagram. (May-98)
45. Design a random sequence generator using serial input a 4 bits parallel output (A, B, D, C) type shift register and associated combinational logic circuit. Explain its operation by drawing waveforms for the outputs (A, B, C, D) of shift register. Assume that the shift register is initially in the state (A, B, C, D) = 1111. (May-98)
46. State the truth table with logic diagram and timing diagram for 4-bit up-down counter. (Dec-98)
47. Explain any two modes of shift register. (Dec-98)
48. Explain the following flip-flops :
 i) Clocked R.S. ii) J.K. with preset and clear iii) M.S. J. K. iv) D-type and T-type (May-99)
49. Design 4 bits binary up/down ripple counter. (May-99)
50. Write a short note on PRBS generator (May-99, Dec-2000)
51. Write a short note on Synchronous counter (May-99)
52. Explain the basic flip-flop circuit for RS using : i) NAND gates ii) NOR gates. (Dec-99)
53. With the help of two shift Registers design a 4-bit Serial Adder. (Dec-99)
54. Design a 3 bit binary Up/Down ripple counter. Draw timing diagram. (Dec-99)
55. Design a conversion logic to convert a JK flip-flop to a D flip-flop. (May-2000)
56. Design and implement a mod-6 synchronous counter using JK flip-flops. (May-2000)
57. Write a short note on Race Around Condition in JK flip-flop (May-2000)
58. Design and implement MOD-5 synchronous counter using JK flip-flops. (Dec-2000)
59. Draw neat circuit diagram of clocked JK flip-flop using NAND gates. Give its truth table and explain race around condition. (Dec-2000)
60. Draw neat diagram of 3-bit bidirectional shift register using JK FFs having right and left data inputs and mode control M such that

In general form the Mealy circuit can be represented with its block schematic as shown in Fig.10.5.

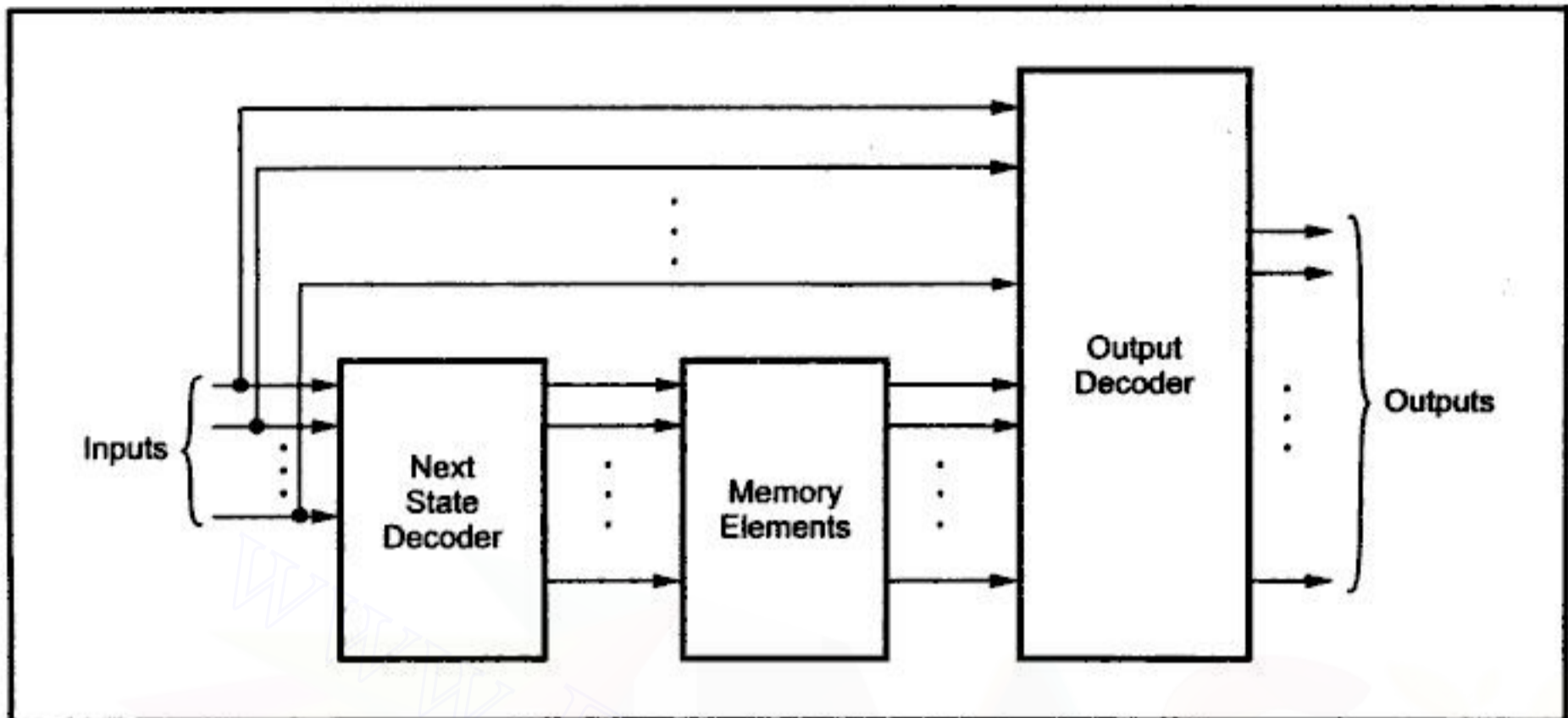


Fig. 10.5 Mealy circuit model

10.2.3 Moore Vs Mealy Circuit Models

Moore Circuit	Mealy Circuit
a) Its output is a function of present state only.	a) Its output is a function of present state as well as present input.
b) Input changes does not affect the output.	b) Input changes may affect the output of the circuit.
c) Moore circuit requires more number of states for implementing same function.	c) It requires less number of states for implementing same function.

Table 10.1

10.3 Analysis of Clocked Sequential Circuits

The behaviour of a sequential circuit is determined from the inputs, the outputs, and the states of its flip-flops. Both the outputs and the next state are function of the inputs and the present state (in case of Moore circuits the outputs are function of only present state). The analysis of sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states. The success of analysis or design of sequential circuit depends largely on the aids and systematic techniques such as state tables, state diagrams and state equations used in these processes.

10.3.3 State Equations

A state equation is also known as application equation. It is an algebraic expression that specifies the conditions for a flip-flop state transition. The left side of the equation represents the next state of the flip-flop and the right side, a Boolean function that specifies the present state conditions that make the next state equal to 1. The state equation can be derived from the state table or logic diagram.

Derivation of state equation from state table

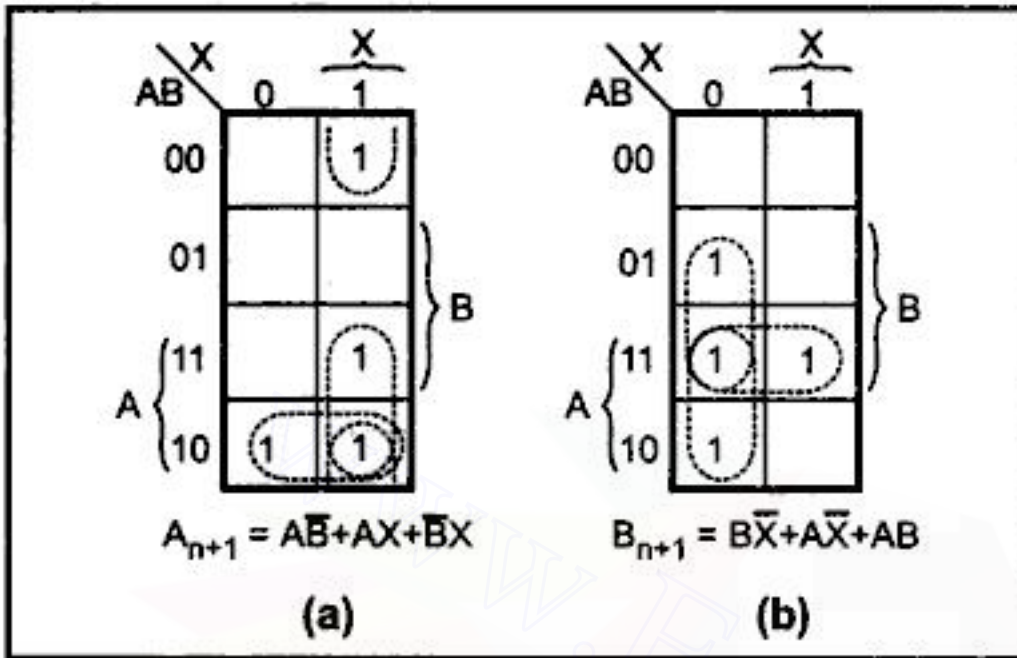


Fig. 10.10 State equations

The state equation can be derived from the state table using K-maps. Let us find the state equation for flip-flop from state table shown in Table 10.2. In the K-map, 1 is marked to represent the input and present state conditions that cause the flip-flop to go to a next state of 1. For example, present states 01, 10 and 11 with X = 0 and present state 11 with X = 1 cause the flip-flop B to go to a next state of 1, therefore these entries are marked '1' in the K-map for flip-flop. (Refer Fig. 10.10).

Derivation of state equation from logic diagram

The state equation can be derived directly from the logic diagram. Looking at Fig. 10.7 we can see that the signal for input of the flip-flop A is generated by the function $\bar{B}X$ and the signal for input K by the function $B\bar{X}$. Substituting $J = \bar{B}X$ and $K = B\bar{X}$ into a JK flip-flop characteristic equation given by $Q_{n+1} = J\bar{Q} + KQ$ we obtain,

$$\begin{aligned}
 A_{n+1} &= (\bar{B}X)\bar{A} + (B\bar{X})A \\
 &= \bar{A}\bar{B}X + (B+X)A && \dots \text{DeMorgan's theorem} \\
 &= \bar{A}\bar{B}X + A\bar{B} + AX \\
 &= A\bar{B} + (A + \bar{A}\bar{B})X \\
 &= A\bar{B} + (A + \bar{B})X && \dots \text{Rule 11 : [} A + \bar{A}B = A + B \text{]} \\
 &= A\bar{B} + AX + \bar{B}X
 \end{aligned}$$

This state equation is same as the state equation obtained from state table. Similarly, we can find the state equation for flip-flop B. For flip-flop B, $J = A\bar{X}$ and $K = A\bar{X}$. Therefore equation for flip-flop B can be given as

$$\begin{aligned}
 B_{n+1} &= A\bar{X}\bar{B} + (\bar{A}\bar{X})B \\
 &= A\bar{X}\bar{B} + (A + \bar{X})B && \dots \text{DeMorgan's theorem} \\
 &= A\bar{X}\bar{B} + A\bar{B} + B\bar{X} \\
 &= (B + A\bar{B})\bar{X} + AB
 \end{aligned}$$

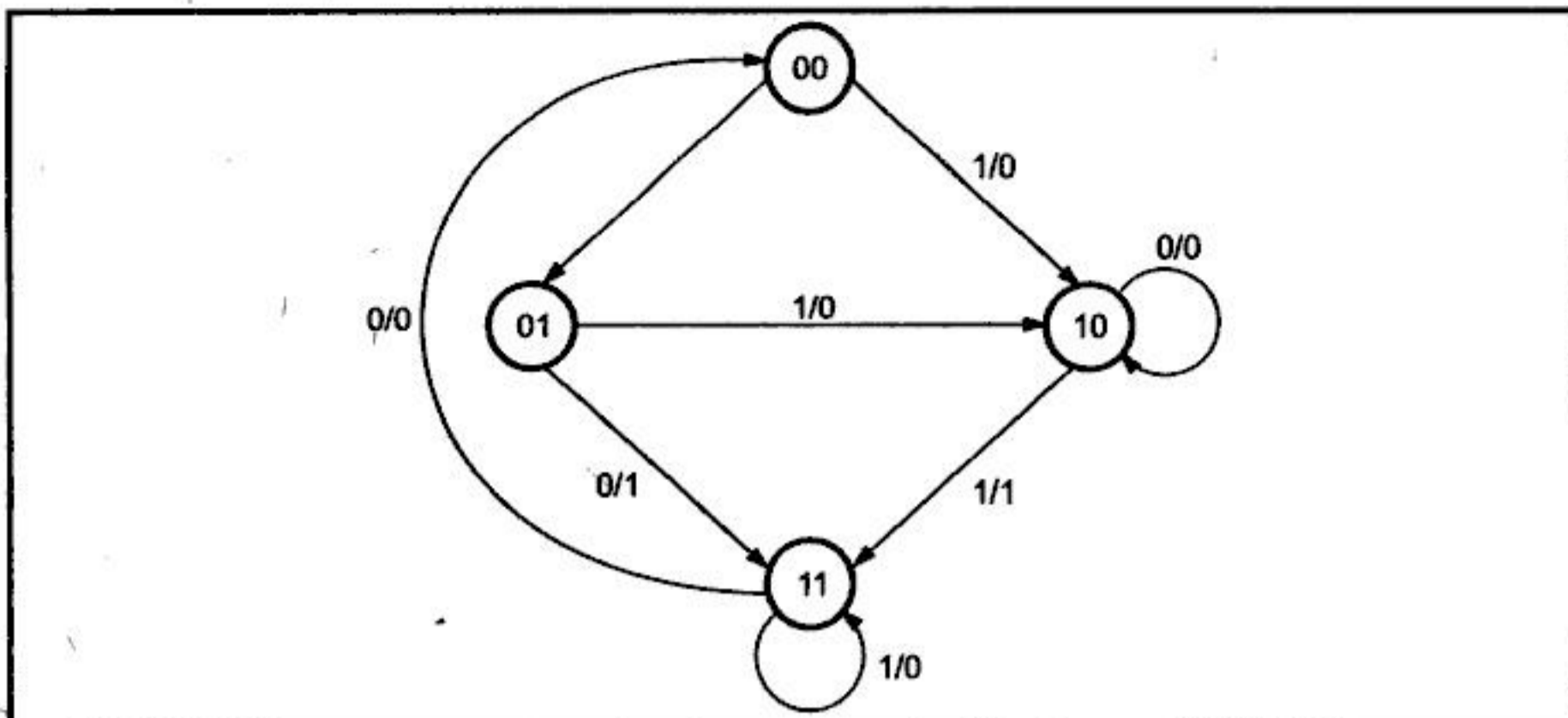


Fig. 10.17 State diagram with binary states

Rules for state assignments

There are two basic rules for making state assignments.

Rule 1: States having the same NEXT STATES for a given input condition should have assignments which can be grouped into logically adjacent cells in a K-map.

Fig. 10.18 shows the example for Rule 1. As shown in the Fig. 10.18, there are four states whose next state is same. Thus states assignments for these states are 100, 101, 110 and 111, which can be grouped into logically adjacent cells in a K-map.

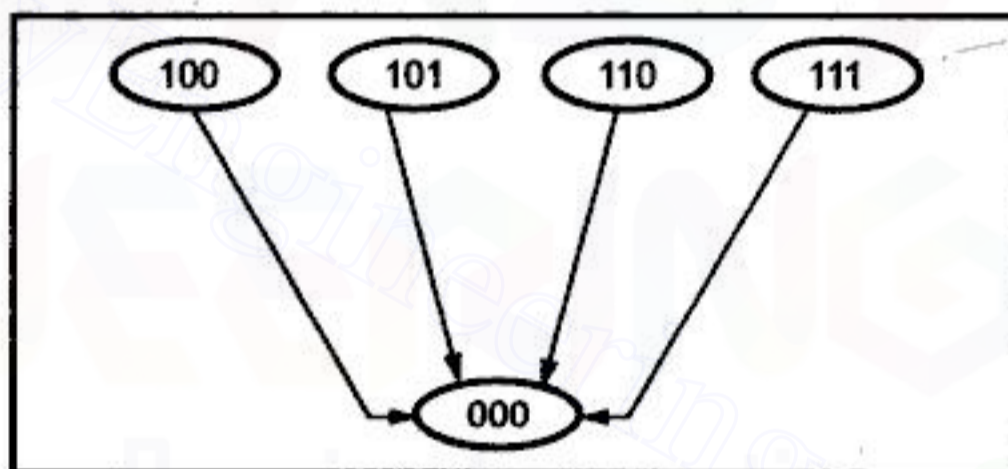


Fig. 10.18 Example of using Rule 1

Rule 2: States that are the NEXT STATES of a single state should have assignment which can be grouped into logically adjacent cells in a K-map.

Fig. 10.19 shows the example for Rule 2. As shown in the Fig. 10.19, for state 000, there are four next states. These states are assigned as 100, 101, 110 and 111 so that they can be grouped into logically adjacent cells in a K-map.

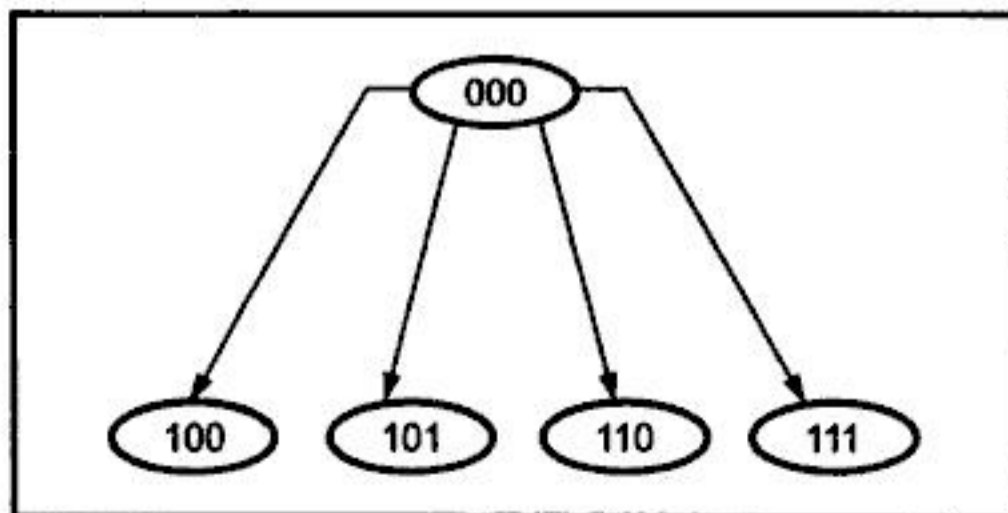


Fig. 10.19 Example of using Rule 2

Present State			Input	Next State			Flip-flop inputs						Output
A	B	C	X	A	B	C	J _A	K _A	J _B	K _B	J _C	K _C	Y
0	0	1	0	0	1	0	0	X	1	X	X	1	1
0	0	1	1	0	1	1	0	X	1	X	X	0	1
0	1	0	0	0	0	1	0	X	X	1	1	X	0
0	1	0	1	0	1	0	0	X	X	0	0	X	1
0	1	1	0	1	0	0	1	X	X	1	X	1	1
0	1	1	1	1	1	0	1	X	X	0	X	1	0
1	0	0	0	1	1	0	X	0	1	X	0	X	0
1	0	0	1	0	0	1	X	1	0	X	1	X	1
1	1	0	0	1	1	0	X	0	X	0	0	X	1
1	1	0	1	0	1	0	X	1	X	0	0	X	0

Table 10.15

K-map simplification for JK inputs and circuit output

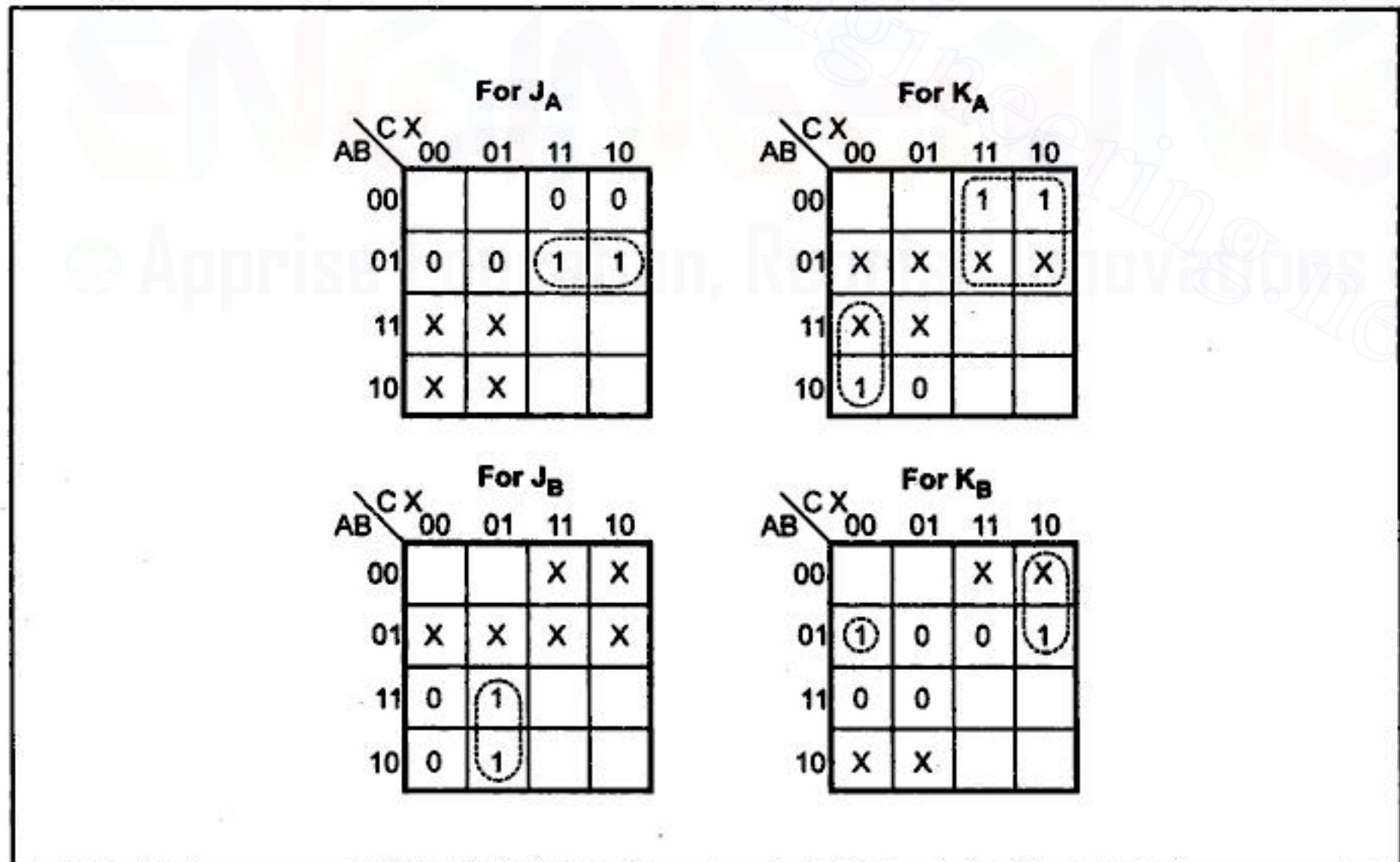


Fig. 10.32 (a)

Logic Diagram

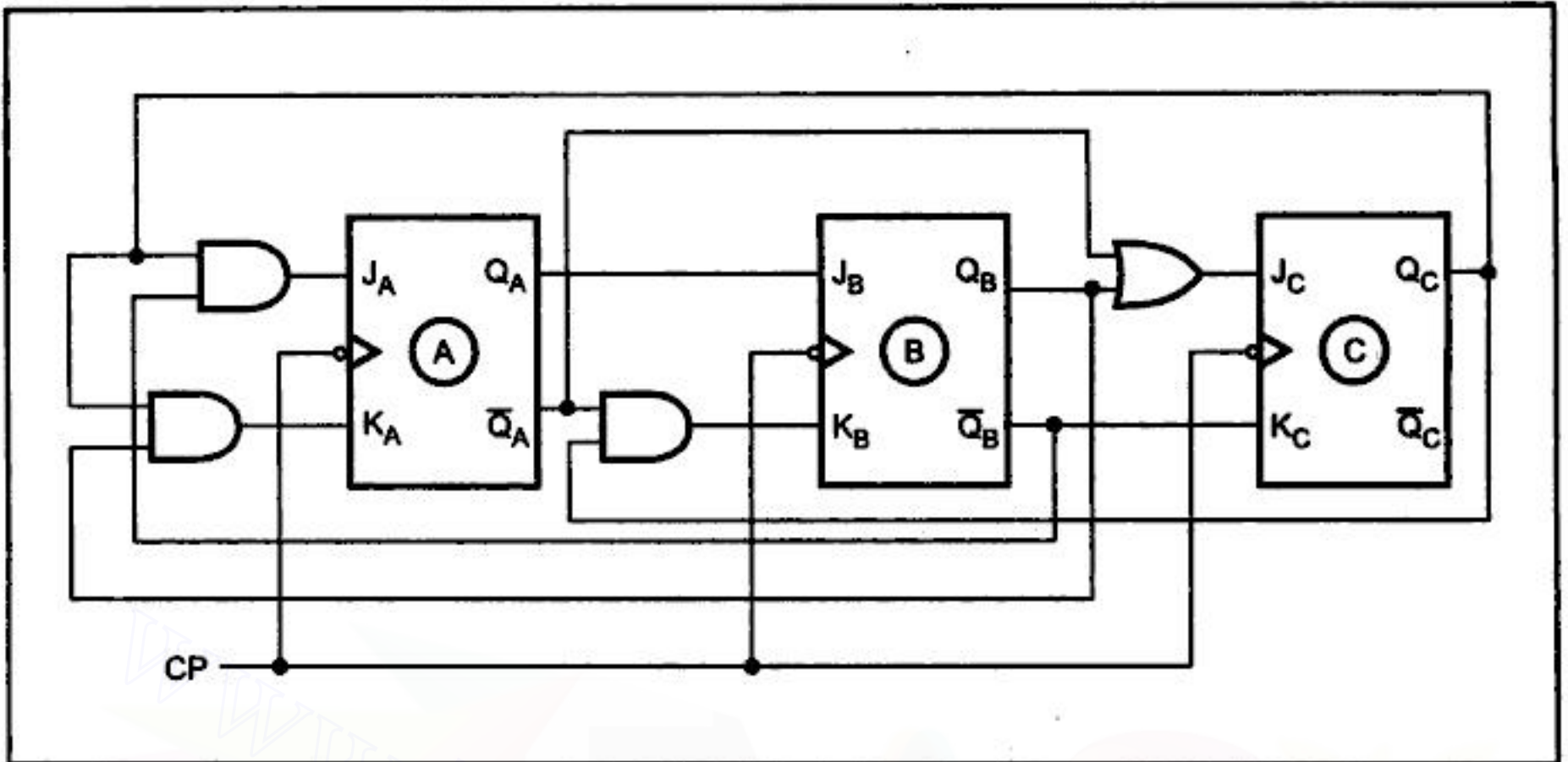


Fig. 10.41

10.8 Sequence Generator

10.8.1 Sequence Generator Using Counters

A sequential circuit which generates a prescribed sequence of bits, in synchronism with a clock, is referred to as a sequence generator. Fig. 10.42 shows the basic structure of a sequence generator using counters.

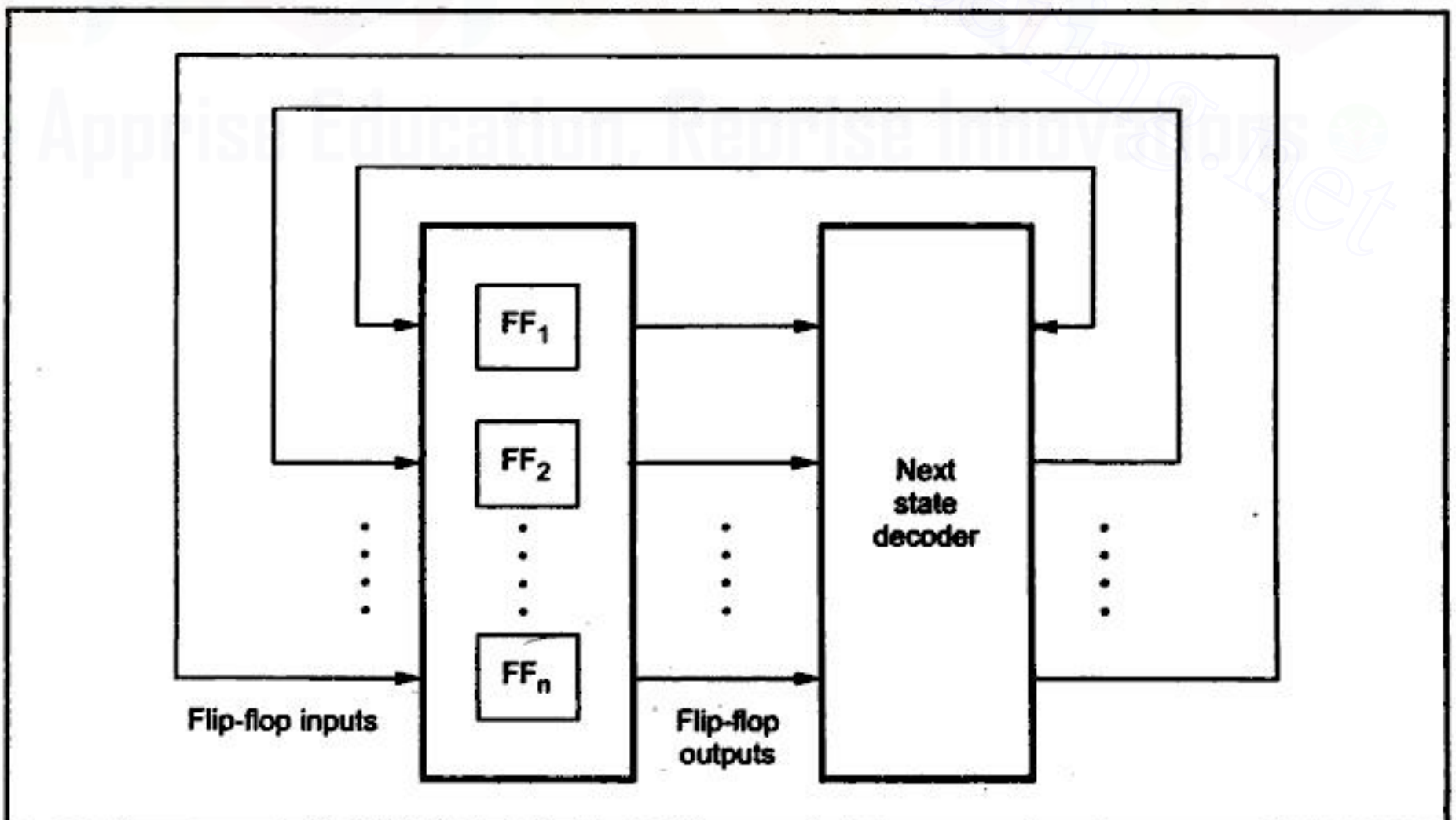


Fig. 10.42 Basic structure of a sequence generator

As shown in the Table 10.20, states are not repeated. After completion of one complete sequence flip-flop is preset to value 1000 to start the next train of sequence. Fig. 10.48 shows the logic diagram.

K-map Simplification for D_{in}

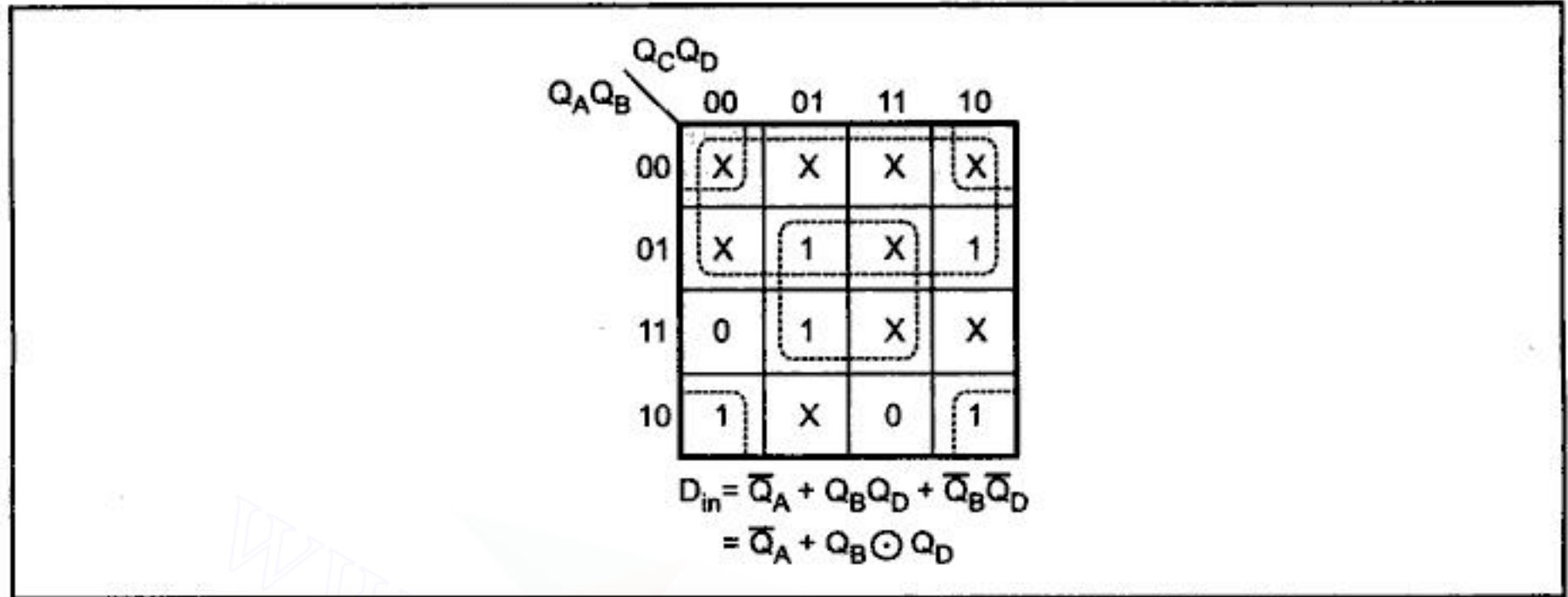


Fig. 10.47

Logic Diagram

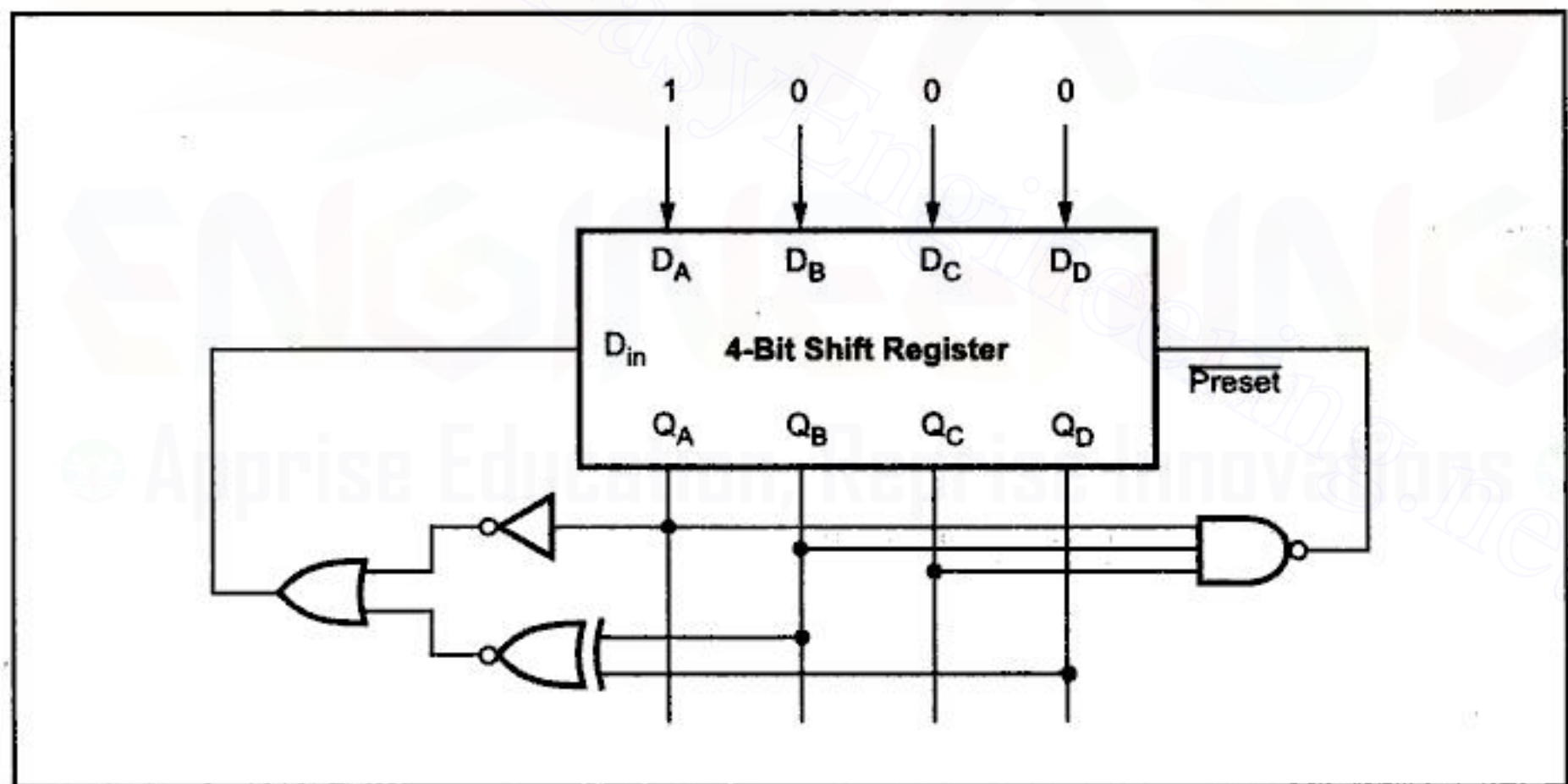


Fig. 10.48

10.9 Sequence Detector

The desired sequence can be detected using a logic circuit called sequence detector. Once we know the sequence which is to be detected, we can draw the state diagram for it. Then from the state diagram we can design the circuit. The following example illustrates how to determine state diagram from a given sequence.

Present State	Next state		Output (Z)	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	d	c	0	0
c	a	b	1	0
d	b	a	1	1

Table 10.22

Even though states a and c are having same next states for input X = 0 and X = 1, as the outputs are not same state reduction is not possible.

Using straight binary assignments as a = 00, b = 01, c = 10 and d = 11, the transition table (Excitations table) is as shown in Table 10.23.

Input X	Present state		Next state		Flip-flop input		Output Z
	Q _A	Q _B	Q _{A+1}	Q _{B+1}	T _A	T _B	
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1
0	1	1	0	1	1	0	1
1	0	0	0	1	0	1	0
1	0	1	1	0	1	1	0
1	1	0	0	1	1	1	0
1	1	1	0	0	1	1	1

Table 10.23

The flip-flop inputs and the circuit outputs are

$$T_A = f(Q_A, Q_B, X)$$

$$T_B = f(Q_A, Q_B, X)$$

$$Z = f(Q_A, Q_B, X)$$

K-map Simplification

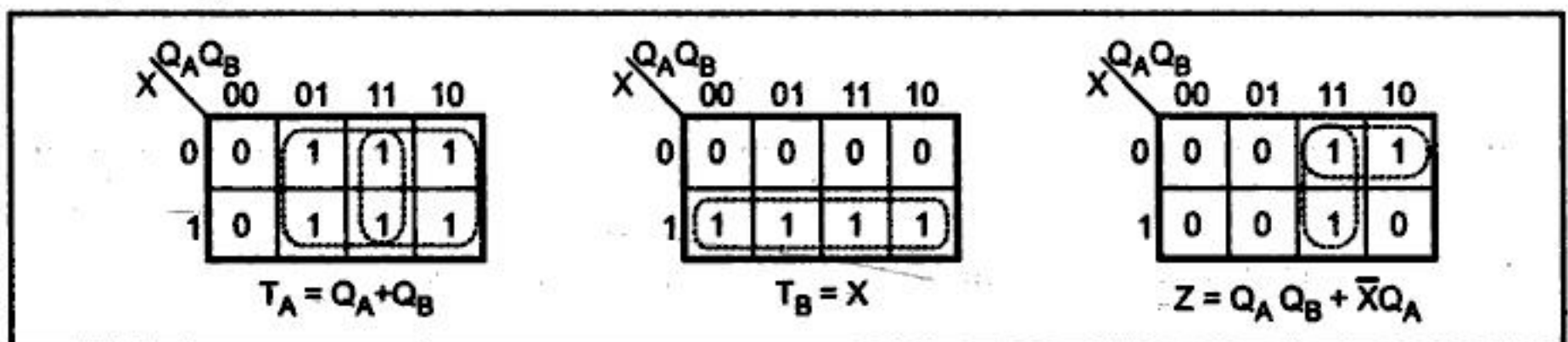


Fig. 10.57

The logic diagram is as shown below :

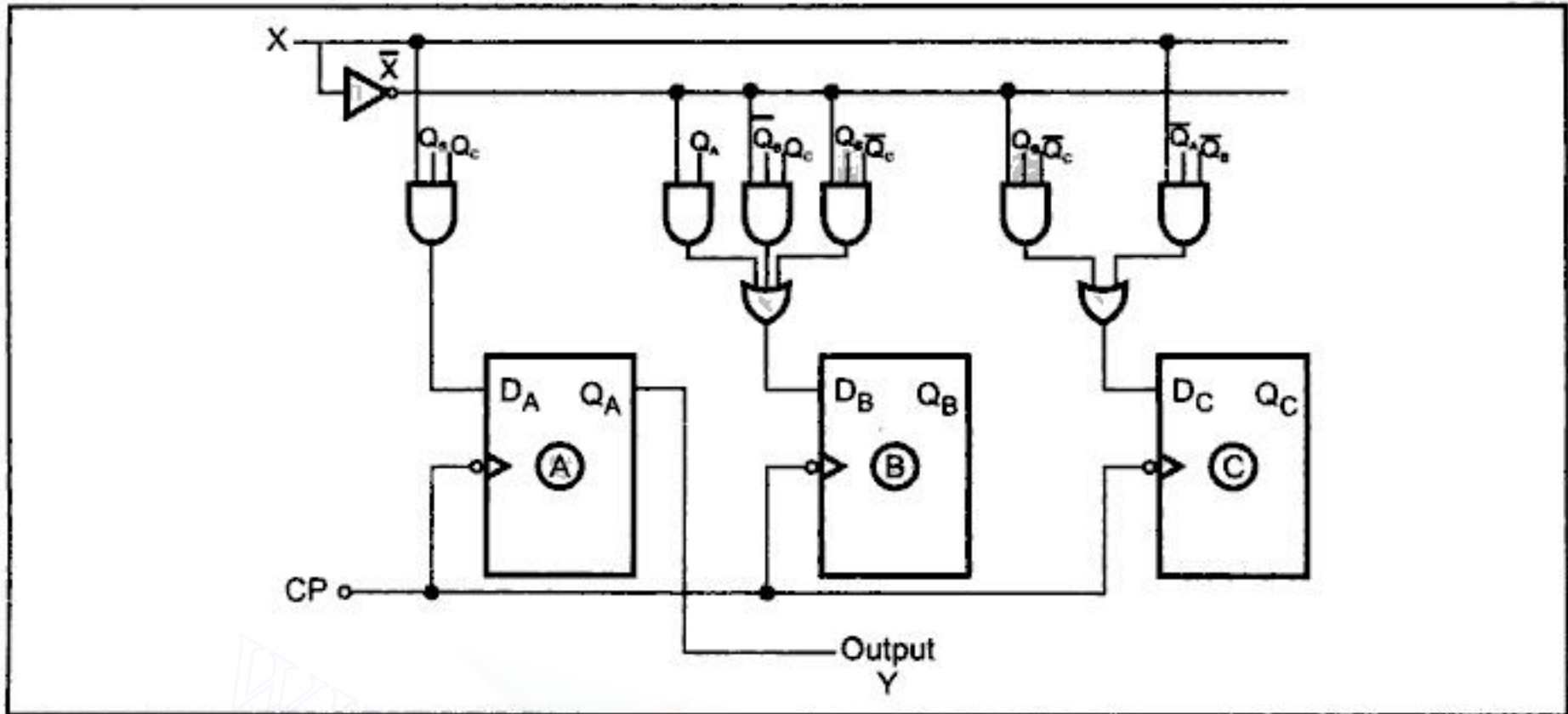


Fig. 10.69

Ex. 10.17 : Design a sequence detector to detect the following sequence using JK flip-flops.
 (Use Mealy Machine) ... 110

Sol. : The given sequence has 3-bit, so we require 3 states in the state diagram. Let us start drawing direction lines, assuming state 'a' is an initial state.

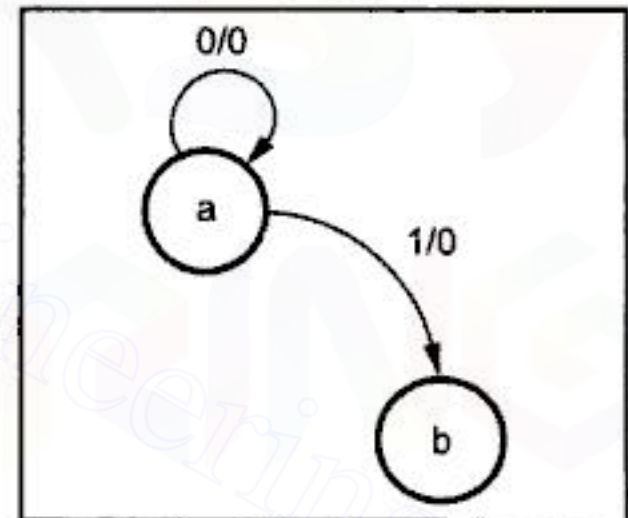


Fig. 10.70 (a)

State a : When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0, since we have not yet detected all the bits in the sequence.

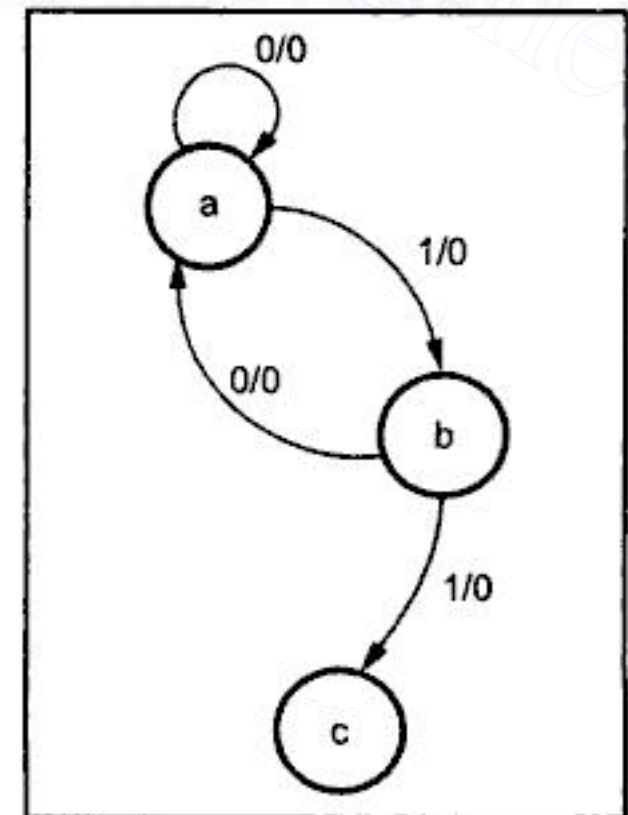


Fig. 10.70 (b)

State b : When input is 1, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 0, we have to go to the state 'a' to detect the first bit in the sequence, i.e. 1.

Sol. : a)

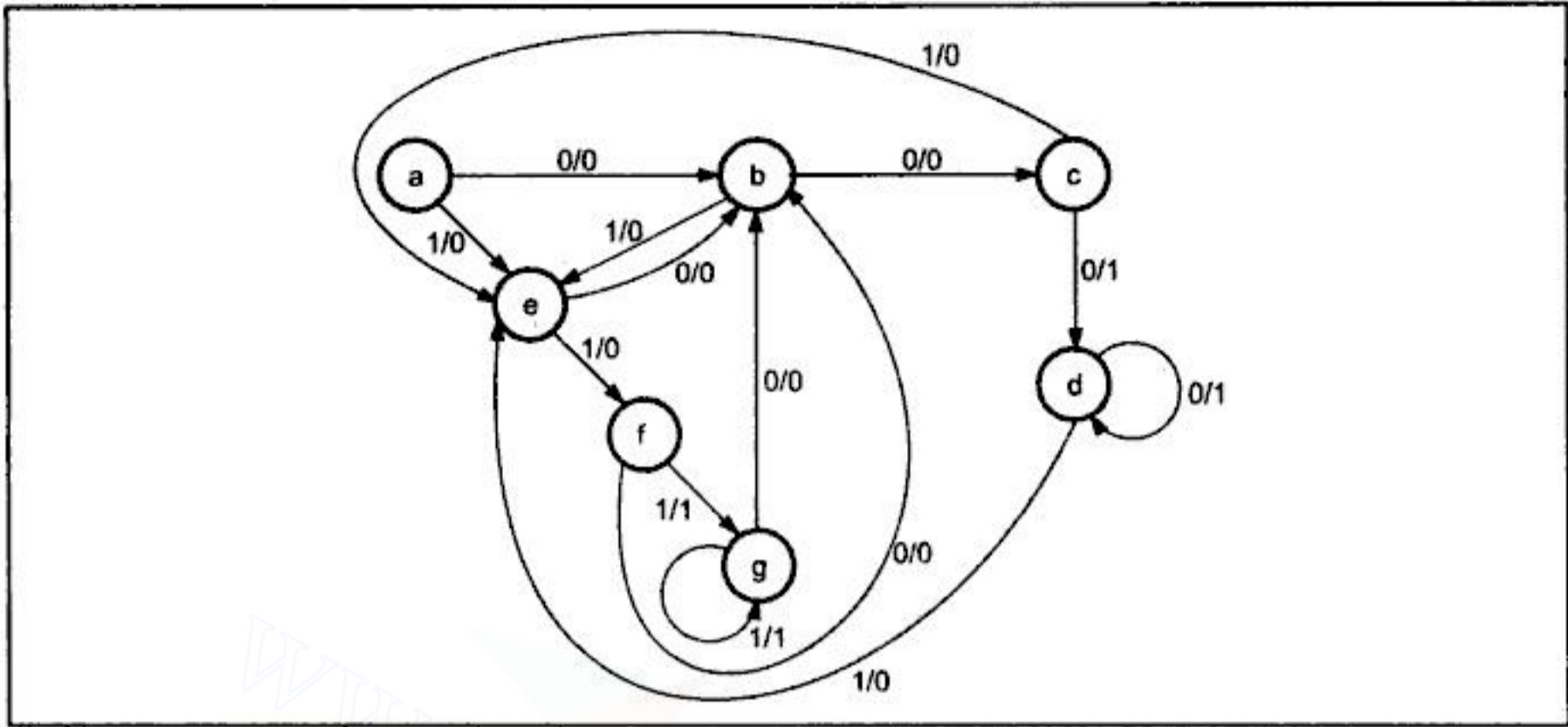


Fig. 10.76 State diagram

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	b	e	0	0
b	c	e	0	0
c	d	e	1	0
d	d	e	1	0
e	b	f	0	0
f	b	g	0	1
g	b	g	0	1

Table 10.34

State reduction : Replace d by c and g by f.

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	b	e	0	0
b	c	e	0	0
c	c	e	1	0
e	b	f	0	0
f	b	f	0	1

Table 10.35

Present state			Next state			Flip-flop inputs					
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
0	1	1	1	0	1	1	X	X	1	X	0
1	0	0	0	1	1	X	1	1	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	1	X	1	X	1	1	X
1	1	1	X	X	X	X	X	X	X	X	X

Table 10.40 Excitation table

K-map Simplification

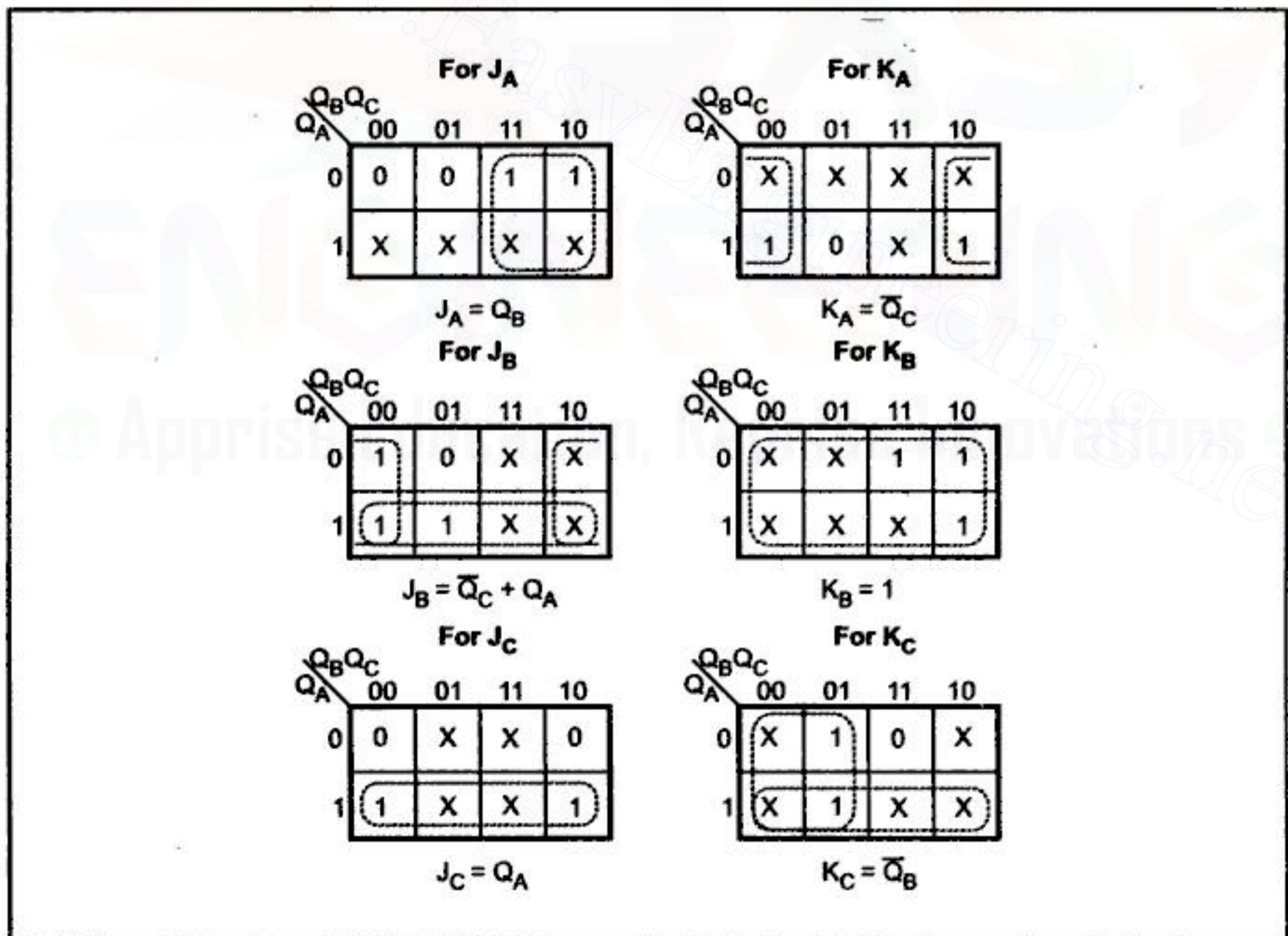


Fig. 10.86

K-maps

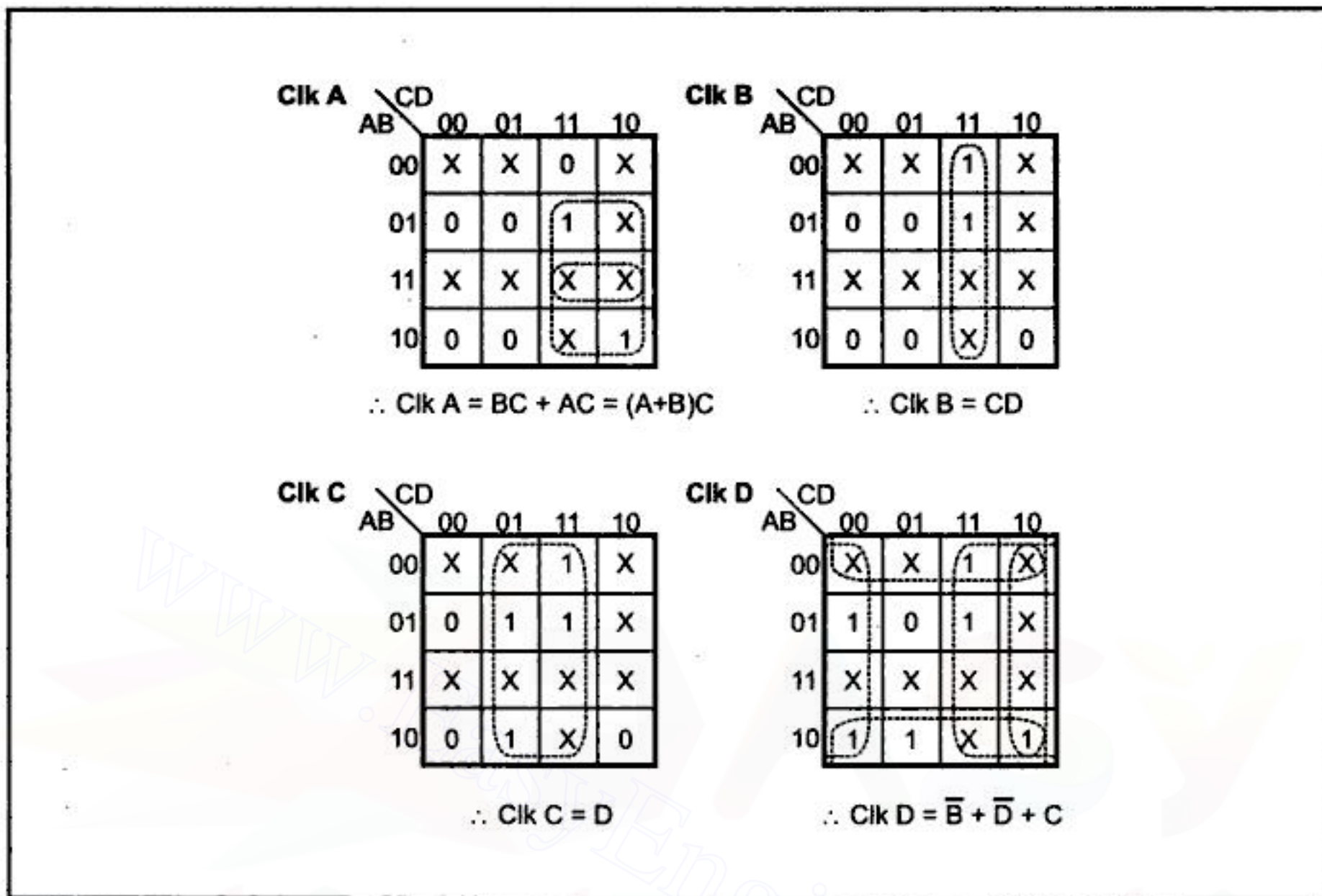


Fig. 10.91

The circuit diagram is shown in Fig. 10.92.

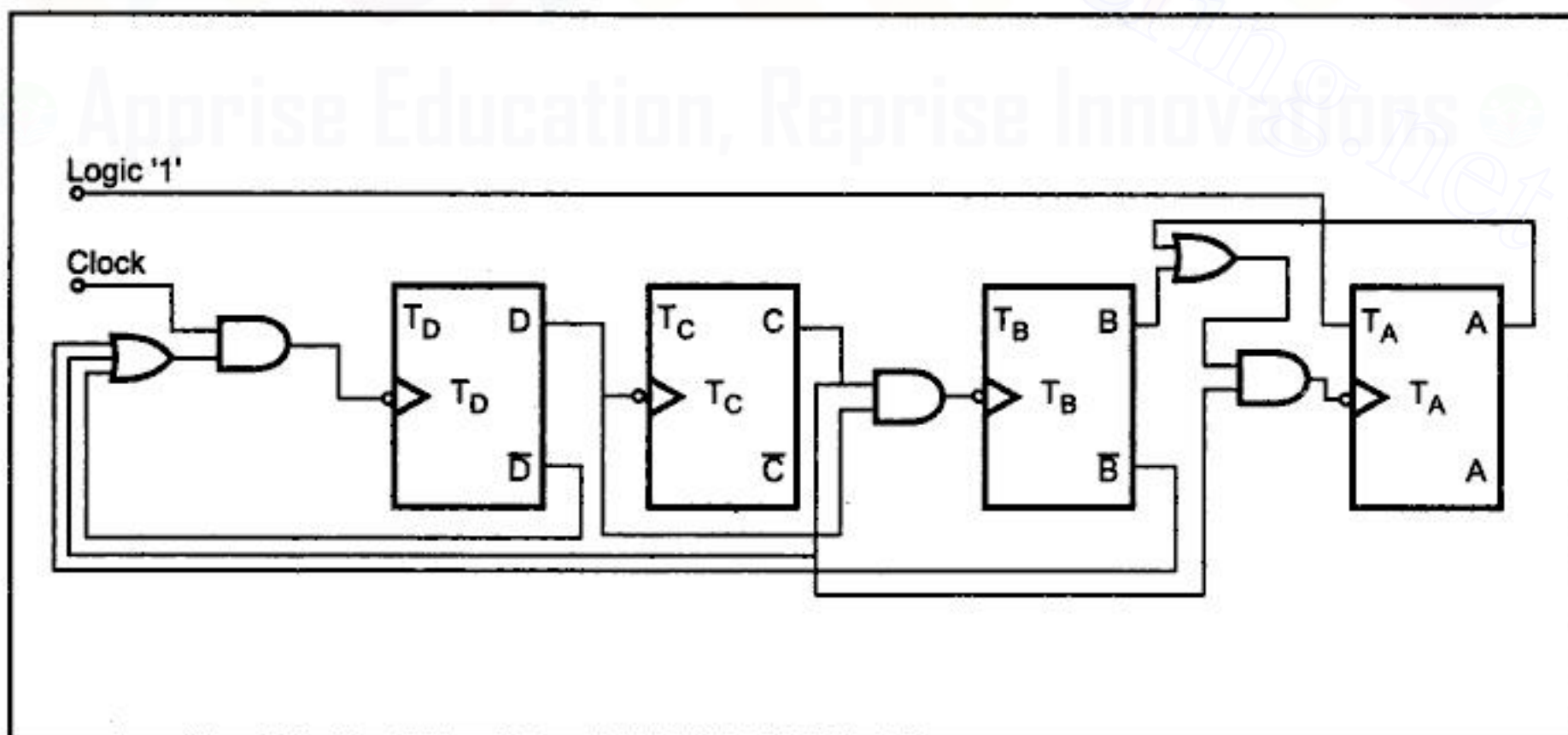


Fig. 10.92

Review Questions

1. Draw and explain the block diagram of Moore circuit.
2. Draw and explain the block diagram of Mealy circuit.
3. Compare Moore and Mealy circuits
4. Analyze the given circuit and obtain the state table.

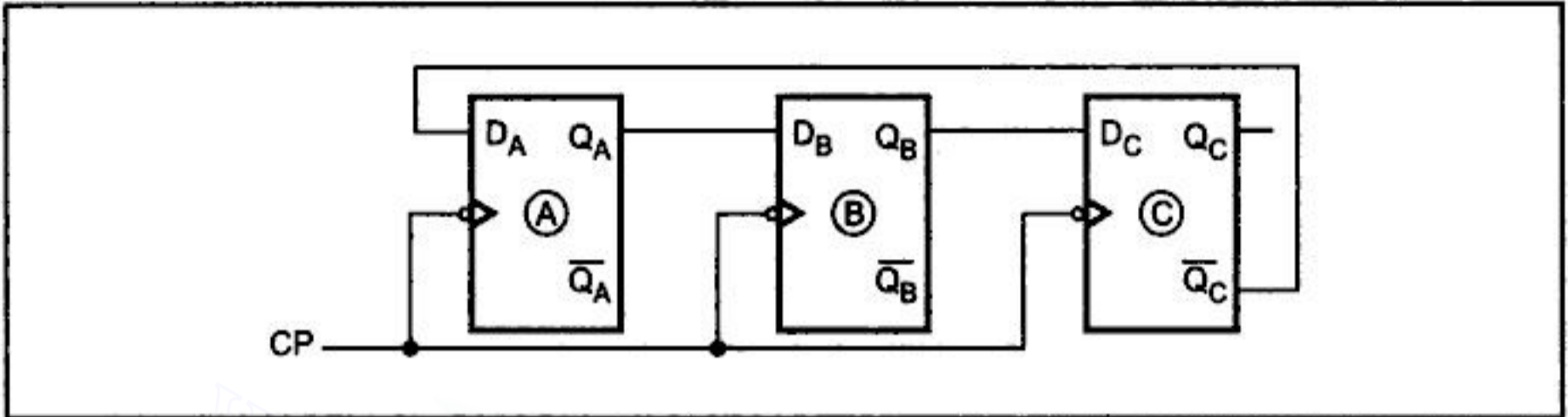


Fig. 10.98

Ans. :

Present States			Next states		
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	1	1

Table 10.48

5. Obtain the state table for the given state diagram and design the sequential circuit using JK flip-flop.

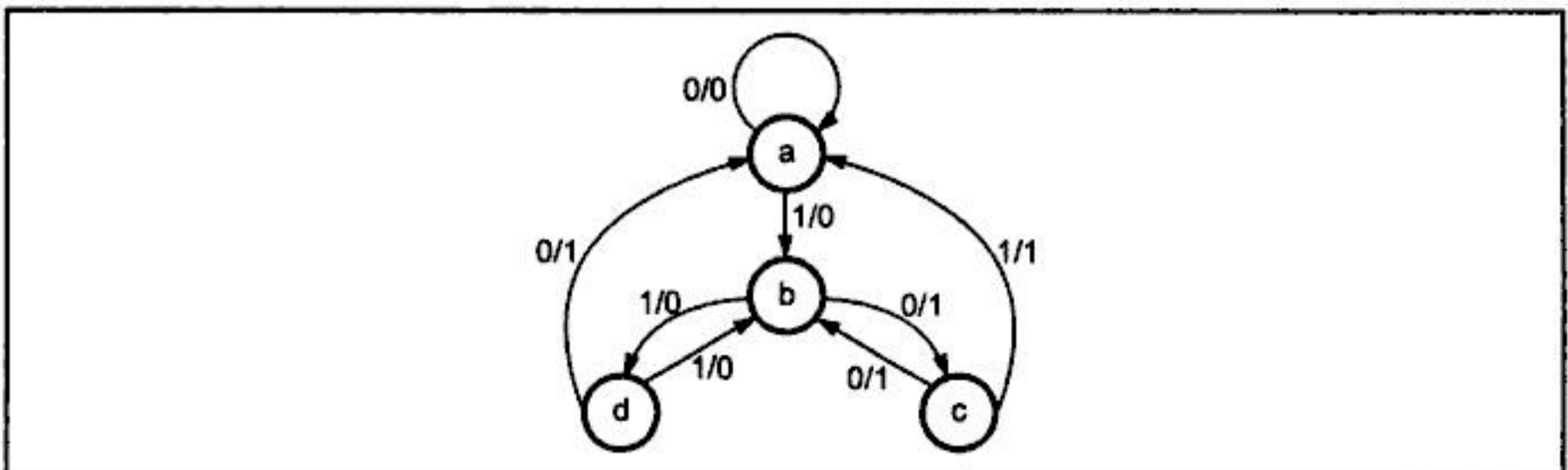


Fig. 10.99

11

Algorithmic State Machines

11.1 Introduction

In a sequential circuit, if there are a large number of external inputs, the state table can grow quite long and often becomes unwieldy. In such situations, designers usually make use of an algorithmic state machine (ASM) chart. An ASM chart is a flowchart that describes the behaviour of a sequential circuit. It is a special flow chart that has been developed specifically to define digital hardware algorithms.

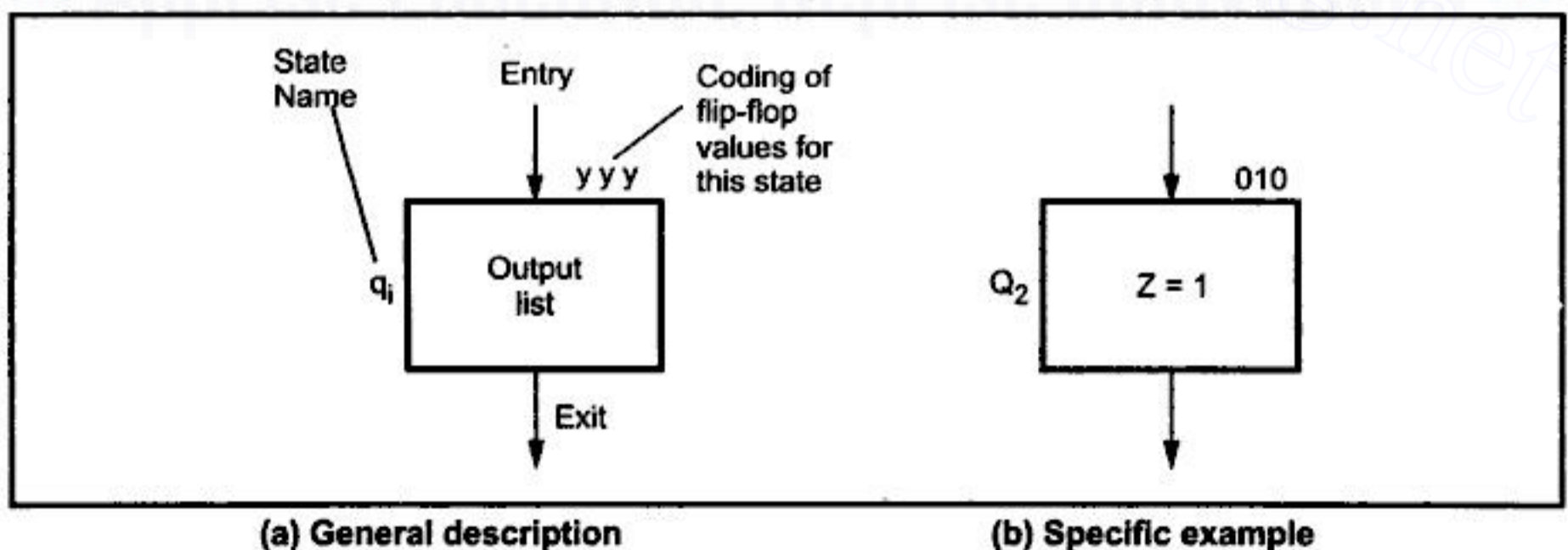
An ASM chart resembles a conventional flow chart, but is interpreted somewhat differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.

In this chapter we study the methods of digital logic design using the ASM chart. The later part of this chapter introduces register transfer language (RTL).

11.2 ASM Chart Notations

An ASM chart consists of three basic elements : the state box, the decision box, and the conditional box.

State Box : A state in the control sequence is indicated by a state box, as shown in the Fig. 11.1.



(a) General description

(b) Specific example

Fig. 11.1 State box

The period of the first two waveforms is four clock cycles, the period of the third is three, and the period of the fourth waveform is two clock cycles, respectively. When an input change does occur, the new waveform may begin at any point in its period.

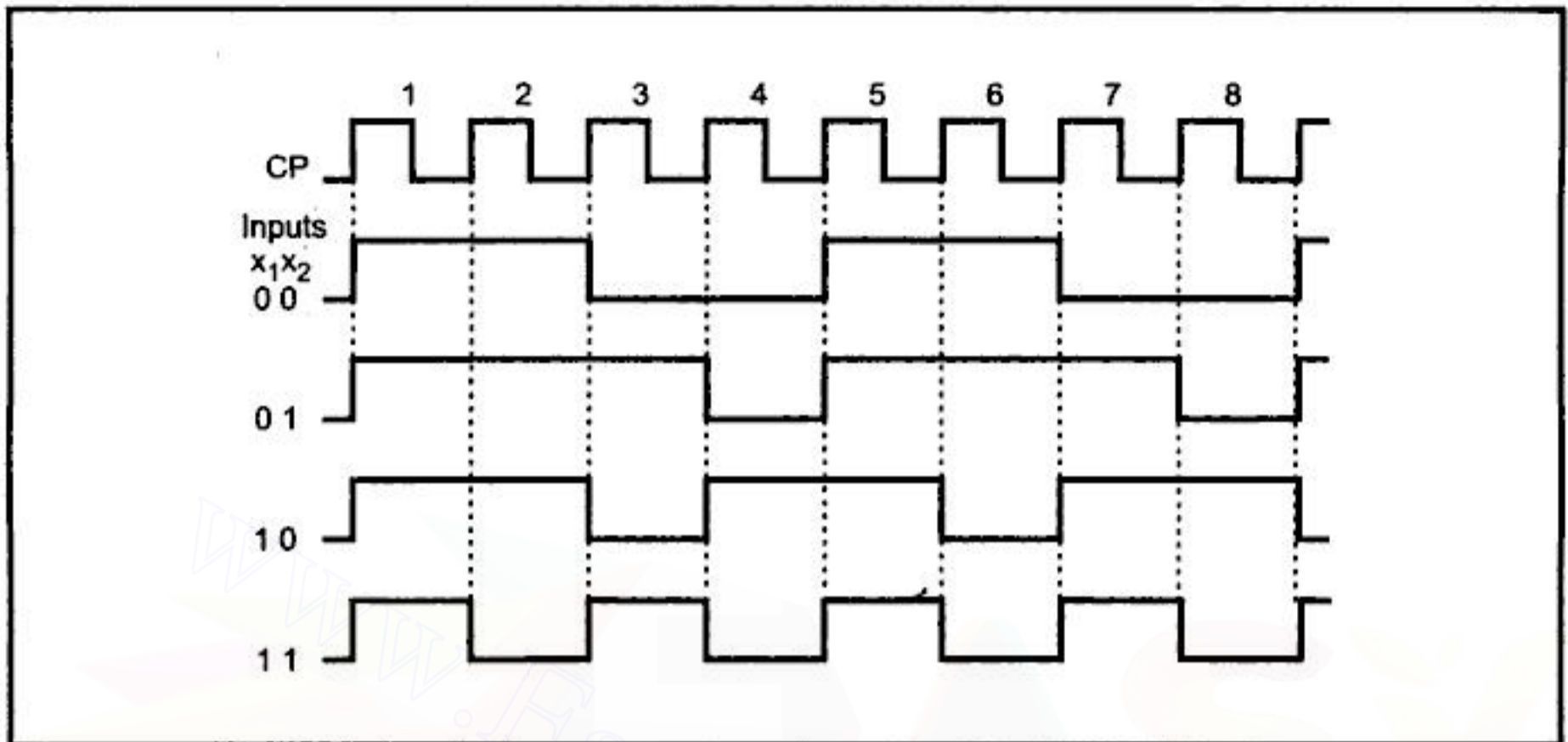


Fig. 11.8 Output waveforms

Sol.: The ASM chart can be drawn for above waveform with four states, one for each clock cycle of the waveforms with the longest period. For each state the output will be conditional on the values of the input lines in effect at that time. Let us observe the different conditions at different states.

1. **State Q_1 :** In the first state, Q_1 , i.e. in the first clock cycle, all waveforms are at logic 1. Therefore, the output, $Z = 1$ and is listed in the state box for the first state, Q_1 .
2. **State Q_2 :** In the second state, Q_2 , i.e. in the second clock cycle, the output is 1 except for the waveform corresponding to inputs $x_1 x_2 = 11$. This condition can be tested by expression $x_1 \wedge x_2$ in the decision box. When the result of expression is 0, the inputs are other than $x_1 x_2 = 11$ and hence output $Z = 1$. This is represented by decision box and conditional box in state Q_2 .
3. **State Q_3 :** During the third state Q_3 , i.e. in the third clock pulse, the output will be 1, if $x_2 = 1$. The condition of x_2 is checked and accordingly output is made 1 by decision box and conditional box in state Q_3 .

Looking at Fig. 11.9 we can realize that the output of third waveform in state Q_1 and state Q_4 is same, i.e. logic 1. In other words, in state Q_4 , the third waveform starts new cycle. Therefore, when $x_1 x_2 = 10$, the fourth state is not used and line goes back to first state to start the new cycle.

4. **State Q_4 :** In the fourth state Q_4 , i.e. in the fourth cycle, the output is always 0. From state Q_4 the circuit returns to Q_1 so that the waveforms may be repeated.

Consider, for example, the ASM chart of Fig. 11.9. It consists of four state and two control inputs x_1 and x_2 . Fig. 11.14 shows three level implementation. It consists of two multiplexers, MUX 1 and MUX 2 ; a register with two flip-flops, A and B; and a combinational circuit to determine the output. The outputs of the register are used to select the inputs of the multiplexers. In this way, the present state of the register is used to select one of the inputs from each multiplexer. The outputs of the multiplexers are then applied to the D inputs of A and B. The purpose of each multiplexer is to produce an input to its corresponding flip-flop equal to the binary value of the next state.

The inputs of the multiplexers are determined from the decision boxes and state transitions given in the ASM charts (Refer Fig. 11.9). The present states, next states and conditions for transition can be tabulated for ASM chart given in Fig. 11.9, as shown in the table.

No.	Present state		Next state		Condition of transition
	A	B	A	B	
1	(Q ₁) 0	0	(Q ₂) 0	1	1
2	(Q ₂) 0	1	(Q ₃) 1	0	1
3	(Q ₃) 1	0	(Q ₁) 0	0	$x_1 \bar{x}_2$
			(Q ₄) 1	1	$\bar{x}_1 \bar{x}_2 + x_2$
4	(Q ₄) 1	1	(Q ₄) 0	0	1

Table 11.2

Inputs for Multiplexers

MUX 1	MUX 2
0 → 0	0 → 1
1 → 1	1 → 0
2 → $\bar{x}_1 \bar{x}_2 + x_2$	2 → $\bar{x}_1 \bar{x}_2 + x_2$
3 → 0	3 → 0

Table 11.3

Note: MUX 1 generates input for flip-flop A and MUX 2 generates input for flip-flop B. The multiplexer input can be determined by including condition of transition corresponding to logic 1 bit position in the next state. Consider the transition from Q₁ to Q₂. For flip-flop A the next state is 0, hence the corresponding input of multiplexer 1 is 0. For flip-flop B the next state is 1, hence the corresponding input of multiplexer 1 is the given condition of transition, i.e. 1.

Consider the transition from Q₃ to Q₁ or Q₄. In this case, the next state for flip-flop A is 0 for Q₁ and 1 for Q₄. Therefore, the condition of transition corresponding to Q₄ is taken as corresponding input of multiplexer 1 i.e. $\bar{x}_1 \bar{x}_2 + x_2$.

The equation for output Z can be directly derived from ASM chart. For this we have to observe the ASM chart and find the conditions when output is 1. For example, in state Q₁,

ASM chart

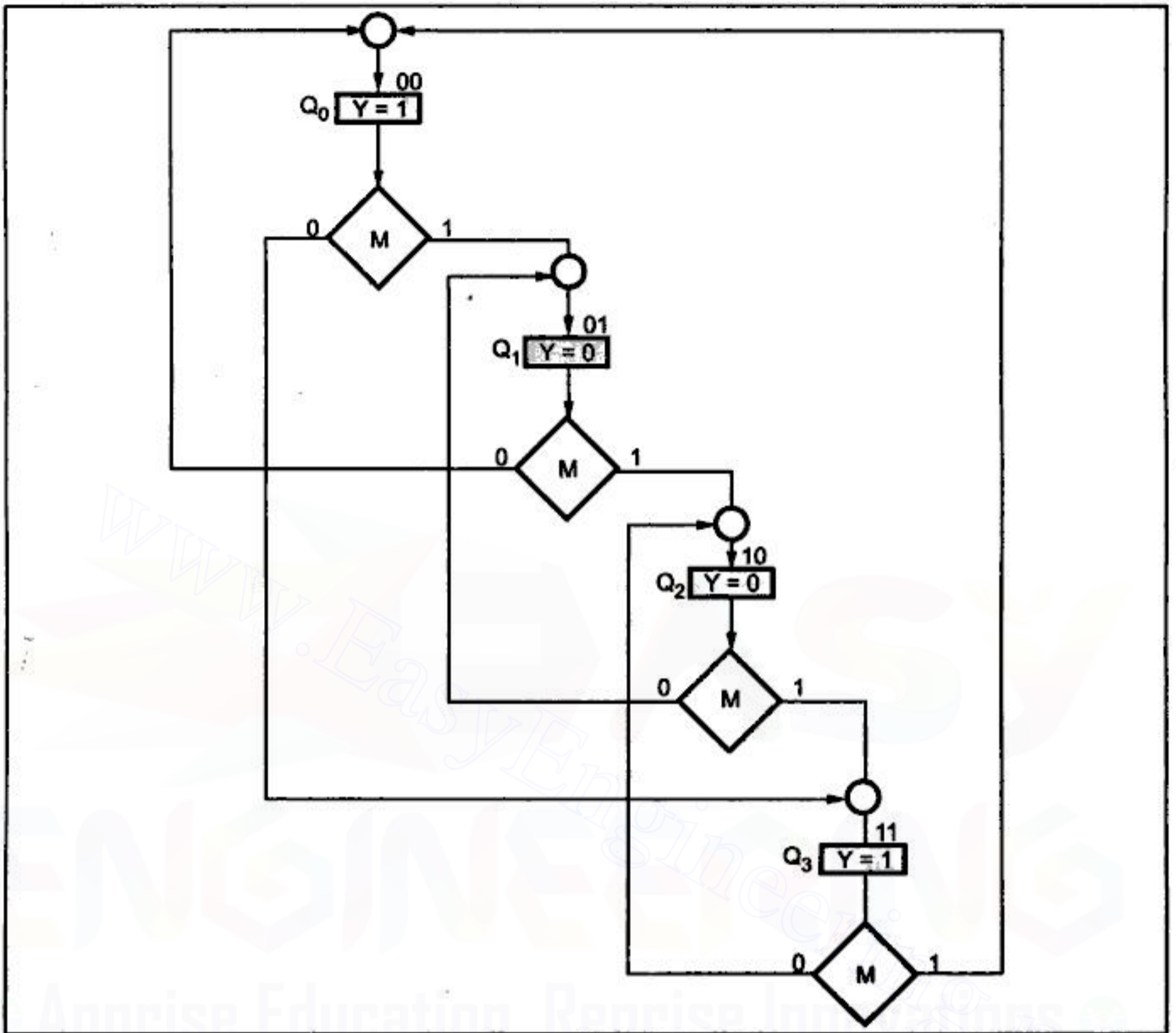


Fig. 11.28

Ex. 11.12 : Draw an ASM chart and state diagram for the circuit shown below in Fig. 11.29.

(May-2001)

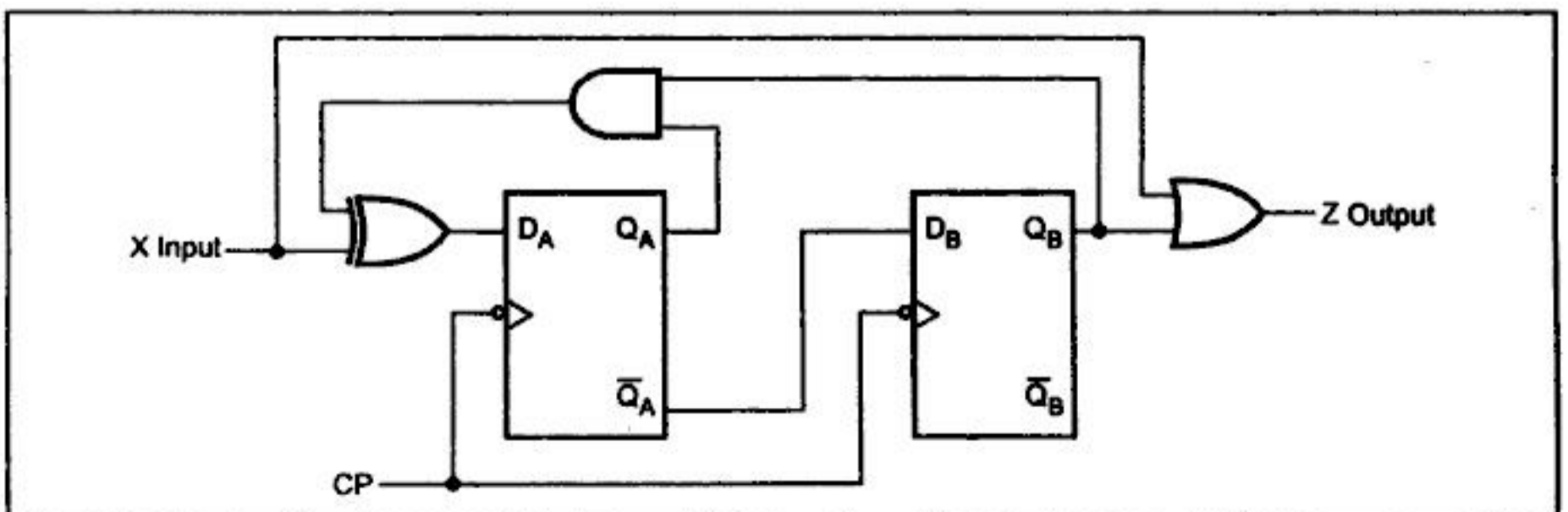


Fig. 11.29

Sol. :

ASM chart

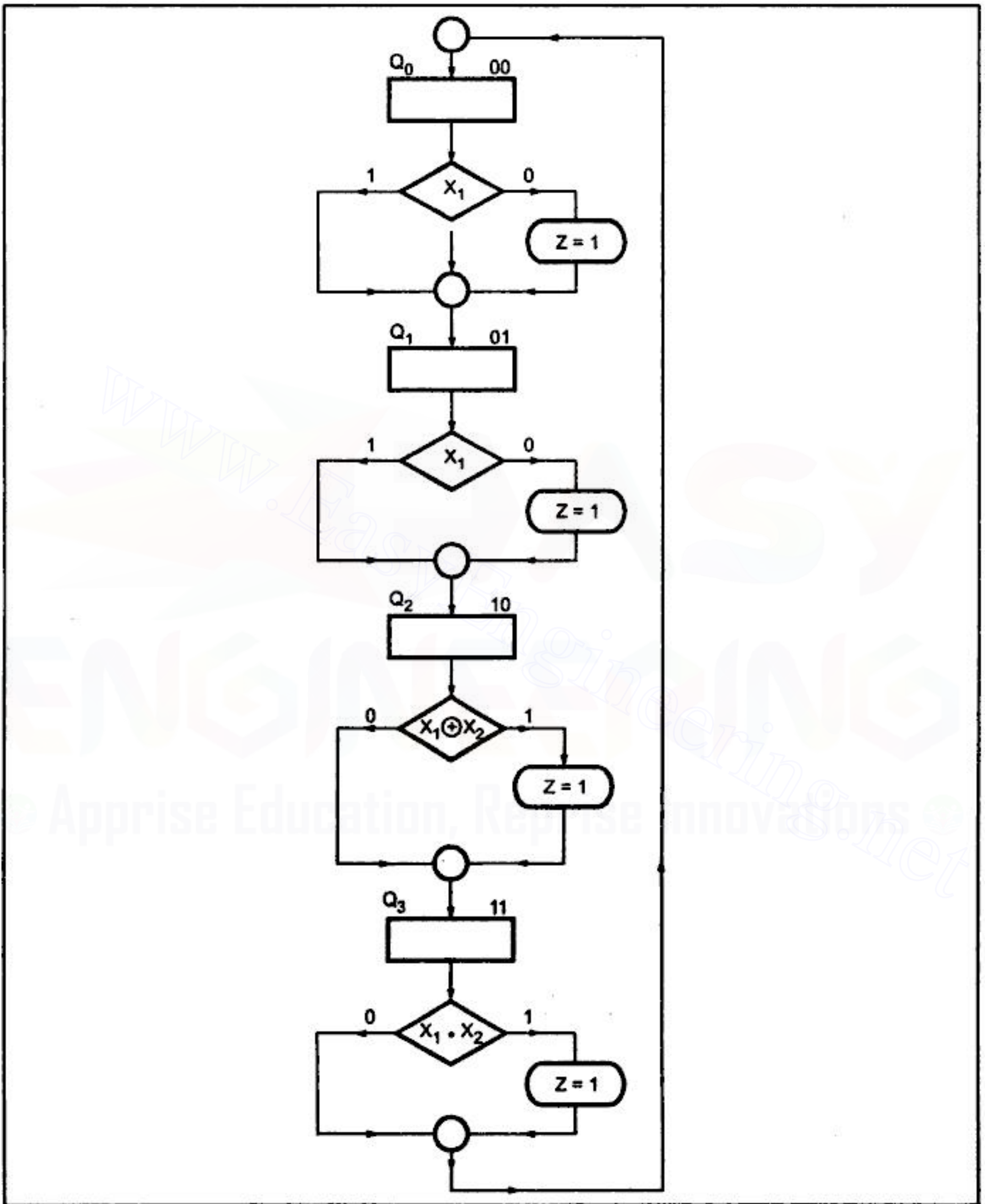


Fig. 11.42

University Questions

1. Define State Machine.

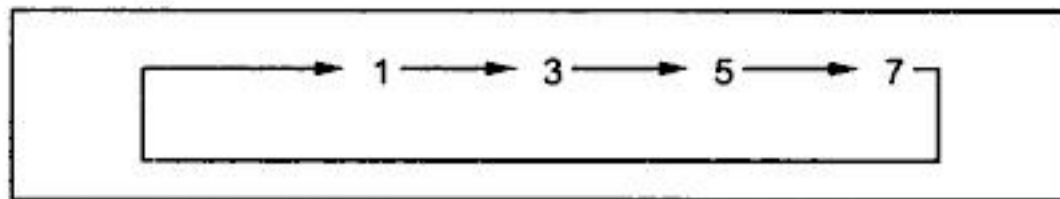


Fig. 11.52

Draw a general model of sequential machine.

Design a synchronous state machine to generate the following sequence of states.

Represent the machine by a state diagram/ ASM chart and display the onset of state 7 (1 1 1) with the help of LED. (Use J-K flip-flop)

(Dec-99)

2. Draw an ASM chart and state table for a 2-bit UP-DOWN counter having mode control input :

$M = 1$ Up counting, $M = 0$ Down counting

The circuit should generate a output = 1 whenever count becomes minimum or maximum.

(May-2000)

3. Write a short note on Algorithmic State Machines.

(Dec-2000)

4. Draw an ASM chart and state diagram for the circuit shown below in Fig. 11.53.

(May-2001)

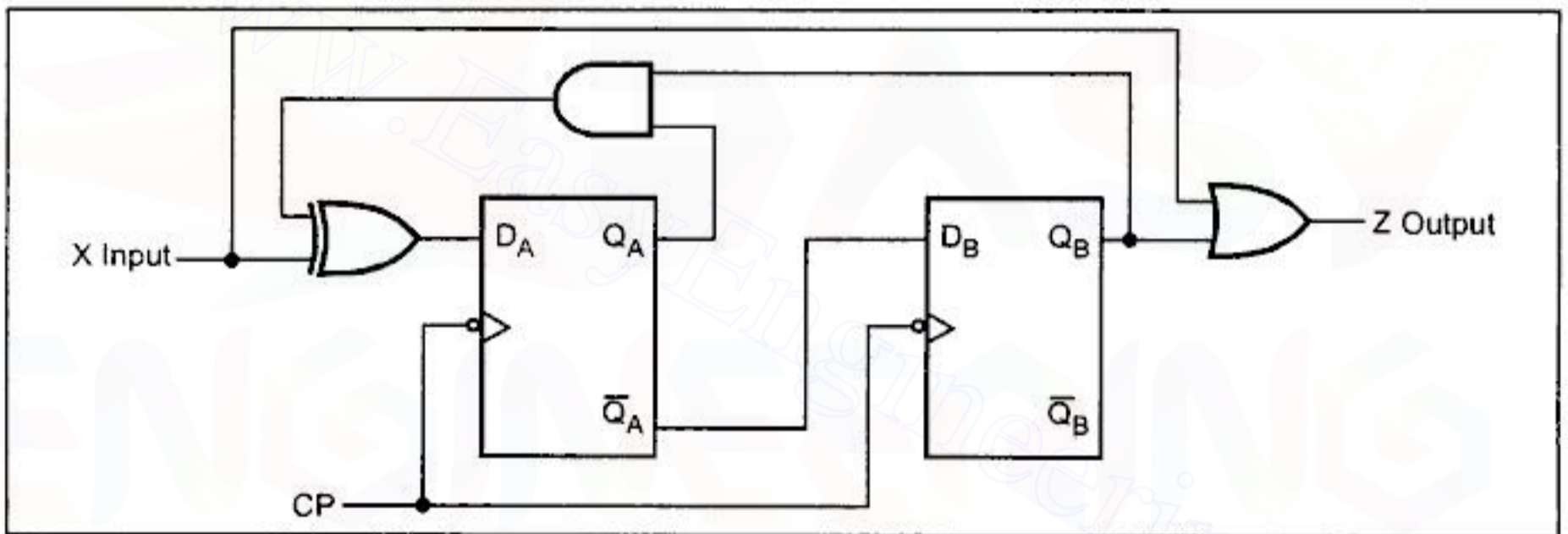


Fig. 11.53

5. Write a short note on algorithmic state machines.

(May-2002, Dec-2003)

6. Convert the following state diagram to an ASM diagram.

(Dec-2002)

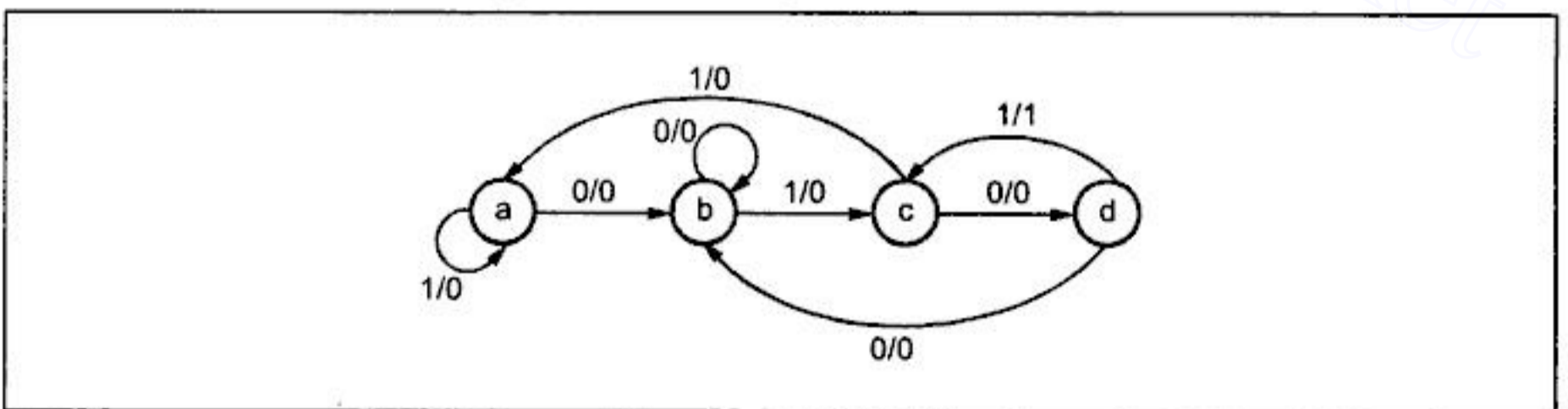


Fig. 11.54

7. Explain with a simple example, significance of various symbols used in constructing an ASM chart for a sequential circuit.

(May-2003)



in the ROM. Therefore, according to the user truth table, manufacturer can deposit thin layer of metal to connect gates of the transistors. Once the pattern/mask is decided, it is possible to make thousands of such ROMs. Such ROMs are called Mask-programmed ROMs. Masked ROMs are used in microprocessor based toys, TV games, home computers and other such high volume consumer products.

12.4.2 PROM (Programmable Read Only Memory)

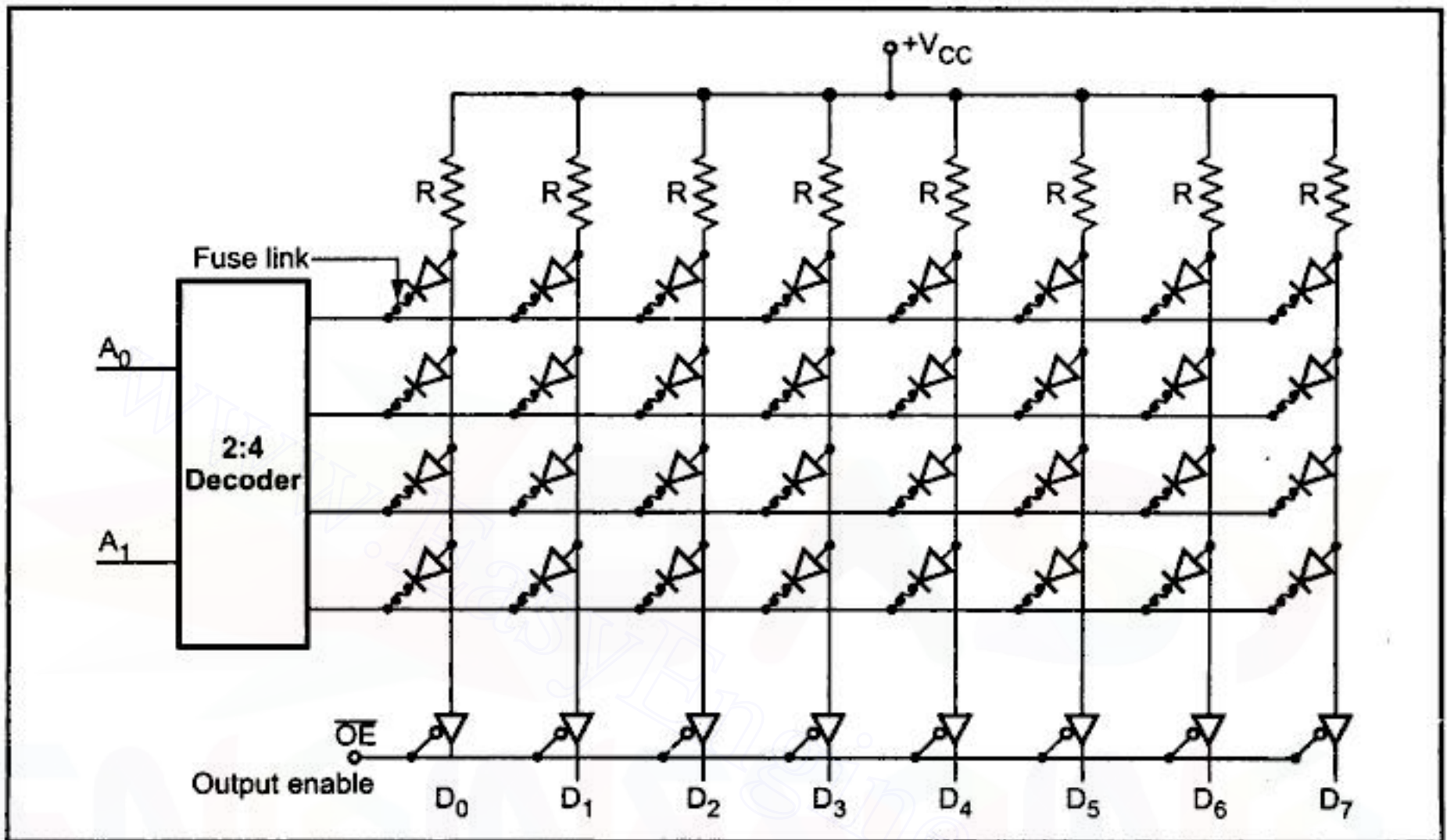


Fig. 12.5 Four byte PROM

Fig. 12.5 shows four byte PROM. It has diodes in every bit position; therefore, the output is initially all 0s. Each diode, however has a fusible link in series with it. By addressing bit and applying proper current pulse at the corresponding output, we can blow out the fuse, storing logic 1 at that bit position. The fuse uses material like nichrome and polycrystalline. For blowing the fuse it is necessary to pass around 20 to 50 mA of current for period 5 to 20 μ s. The blowing of fuses according to the truth table is called programming of ROM. The user can program PROMs with special PROM programmer. The PROM programmer selectively burns the fuses according to the bit pattern to be stored. This process is also known as burning of PROM. The PROMs are one time programmable. Once programmed, the information stored is permanent.

12.4.3 EPROM (Erasable Programmable Read Only Memory)

Erasable programmable ROMs use MOS circuitry. They store 1s and 0s as a packet of charge in a buried layer of the IC chip. EPROMs can be programmed by the user with a special EPROM programmer. The important point is that we can erase the stored data in the EPROMs by exposing the chip to ultraviolet light through its quartz window for 15 to 20 minutes, as shown in the Fig. 12.6.

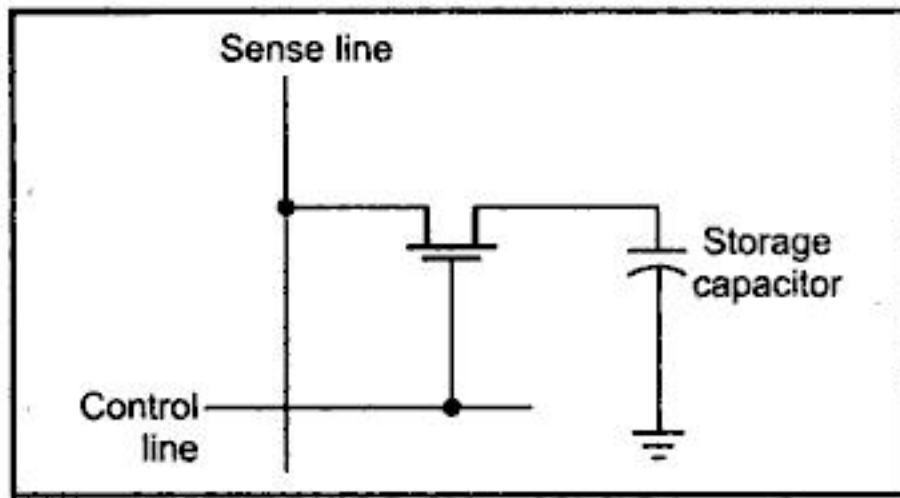


Fig. 12.11 Dynamic RAM

When the COLUMN and ROW lines go low, the MOSFET opens and the capacitor retains its charge. In this way, it stores 1 bit. Since only a single MOSFET and capacitor are needed, the dynamic RAM contains more memory cells as compared to static RAM per unit area.

The disadvantage of dynamic RAM is that it needs refreshing of charge on the capacitor after every few milliseconds. This

complicates the system design, since it requires the extra hardware to control refreshing of dynamic RAMs.

In this type of cell, the transistor acts as a switch. The basic simplified operation is illustrated in Fig. 12.12 As shown in the Fig. 12.12 circuit consists of three tri-state buffers : input buffer, output buffer and refresh buffer. Input and output buffers are enabled and disabled by controlling R/W line. When R/W line is LOW, input buffer is enabled and output buffer is disabled. When R/W line is HIGH, input buffer is disabled and output buffer is enabled. With this basic information let us see the read, write and refresh operations.

12.5.2.1 Write Operation

To enable write operation R/W line is made LOW which enables input buffer and disables output buffers, as shown in the Fig. 12.12 (a). To write a 1 into the cell, the D_{IN} line is HIGH, and the transistor is turned ON by a HIGH on the ROW line. This allows the capacitor to charge to a positive voltage. When 0 is to be stored, a LOW is applied to the D_{IN} line. The capacitor remains uncharged, or if it is storing a 1, it discharges as indicated in Fig. 12.12 (b). When the ROW line is made LOW, the transistor turns off and disconnects the capacitor from the data line, thus storing the charge (either 1 or 0) on the capacitor.

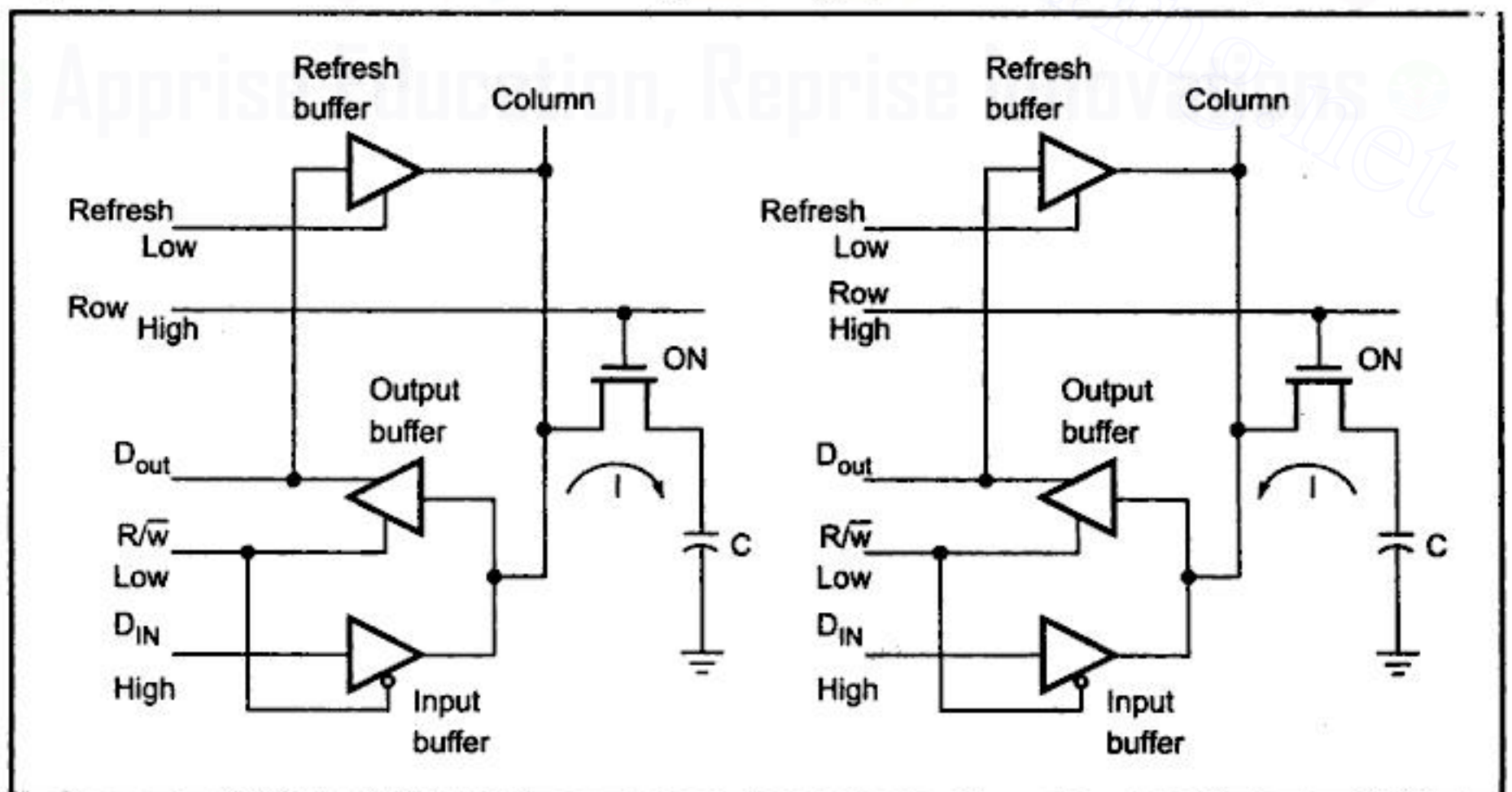


Fig. 12.12 (a) Writing a 1 into the memory cell Fig. 12.12 (b) Writing a 0 into the memory cell

University Questions

1. Write a short-note on semi-conductor memories (May-99)
2. With neat diagram explain the working of MOS DRAM Cell. (May-2000)
3. Discuss various types of read-only memories. (May-2000)
4. Static and Dynamic RAM (Dec-2000)
5. With neat circuit diagram explain the operation of bipolar static RAM cell. (May-2001)
6. Discuss various types of ROMs and their applications in brief. (May-2001)
7. Write short notes on :
 - a) Static and Dynamic MOS RAM
 - b) ROM, EPROM, EEPROM (Dec-2001)
8. Distinguish between volatile and non-volatile memories. (May-2002)
9. Explain the differences between SRAM and DRAM. (May-2002)
10. Write a short note on static bi-polar RAM cell. (Dec-2002)
11. With the help of neat circuit diagram give the working of static MOS RAM cell. (May-2003)
12. Write a short note on dynamic RAM. (Dec-2003)



ENGINEERING

Apprise Education, Reprise Innovations

www.EasyEngineering.net

Ex. 13.1 : Implement the following Boolean functions using ROM

$$F_1(A_1, A_0) = \sum m (1, 2)$$

$$F_2(A_1, A_0) = \sum m (0, 1, 3)$$

Sol. : The given Boolean functions indicate that the circuit has 2 inputs (A_1 and A_0) and 2 outputs (F_1 and F_2), therefore these Boolean functions can be implemented by 4×2 ROM. Fig. 13.4 shows the implementation. As shown in the Fig. 13.4 links are broken of those minterms which are not included in the Boolean function.

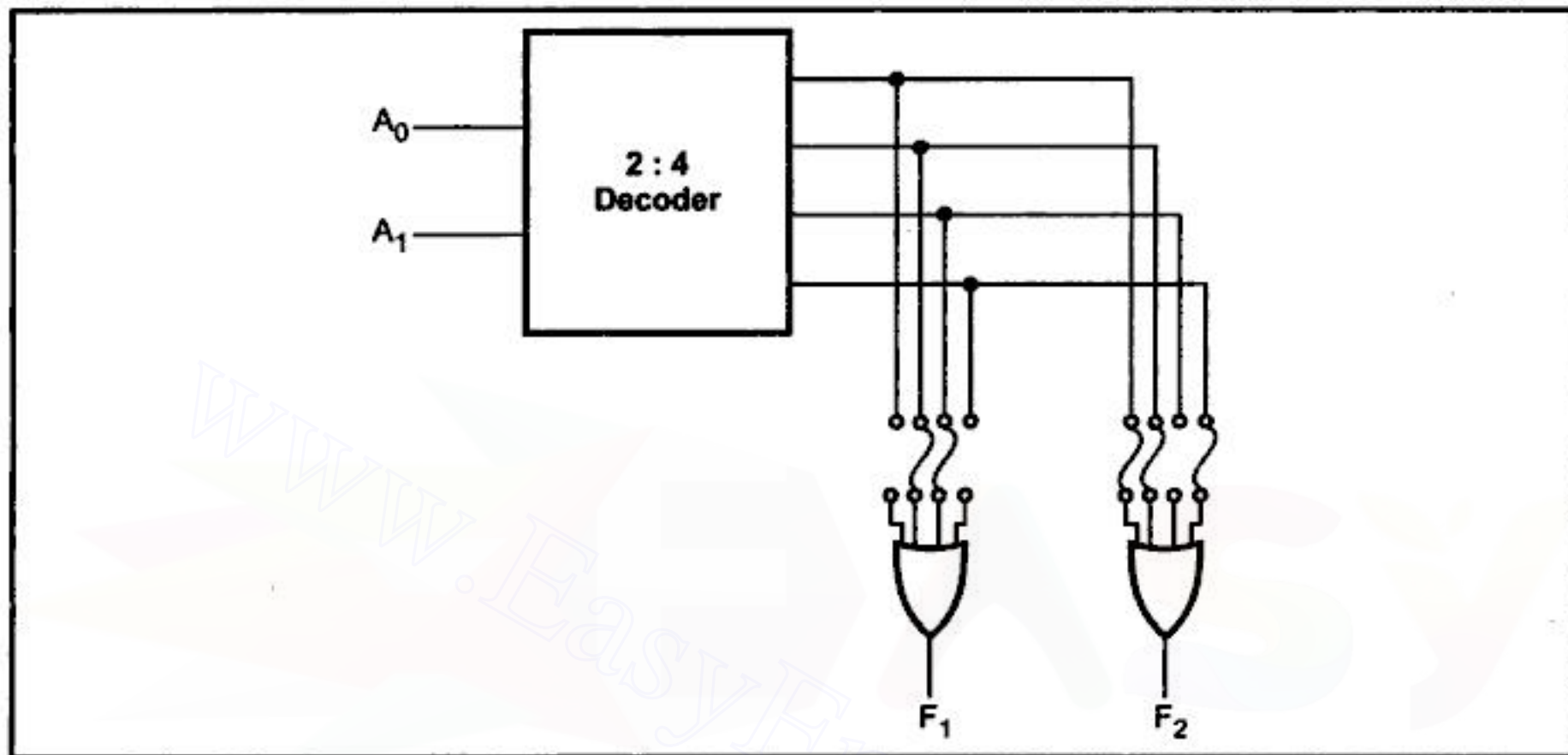


Fig. 13.4

Ex. 13.2 : Design a combinational using a ROM. The circuit accepts 3-bit binary number and generates its equivalent Excess-3 code.

Sol. : Let us derive the truth table for the given combination circuit. Table 13.1 shows the truth table.

Inputs			Outputs			
A_2	A_1	A_0	B_3	B_2	B_1	B_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Table 13.1 Truth table for 3-bit binary to Excess-3 converter

Logic Diagram

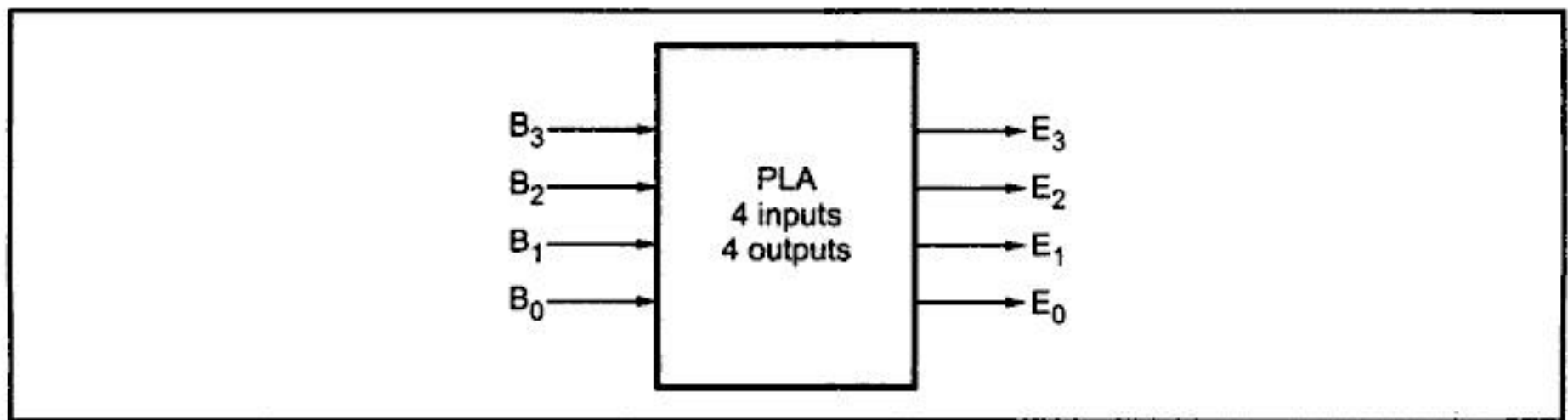


Fig. 13.12 BCD to Excess-3 code converter using PLA

13.4 Programmable Array Logic (PAL)

Programmable logic devices have many gates interconnected through many of electronic fuses. It is sometimes convenient to draw the internal logic of such devices in a compact form referred to as **array logic**. Fig. 13.13 shows the conventional and array logic symbols for a multiple-input AND gate.



Fig. 13.13 (a) Conventional symbol

Fig. 13.13 (b) Array logic symbol

The array logic symbol shown in the Fig. 13.13 (b) uses a single horizontal line connected to the gate input and multiple vertical lines to indicate the individual inputs. Each intersection between horizontal line and vertical line indicates the fuse connection.

We have seen that PLA is a device with a programmable AND array and programmable OR array. However, PAL programmable array logic is a programmable logic device with a fixed OR array and a programmable AND array. Because only AND gates are programmable, the PAL is easier to program, but is not as flexible as the PLA. Fig. 13.14 shows the array logic of a typical PAL. It has four inputs and four outputs. Each input has buffer and an inverter gate. It is important to note that two gates are shown with one composite graphic symbol with normal and complement outputs. There are four sections. Each section has three programmable AND gates and one fixed OR gate. As shown in the Fig. 13.14, each AND gate has 10 fused programmable inputs. The output of section 1 is connected to a buffer-inverter gate and then fed back into the inputs of the AND gates through fuses.

The commercial PAL devices has more gates than the one shown in Fig. 13.14. A typical PAL integrated circuit may have eight inputs, eight outputs, and eight sections, each consisting of an eight wide AND-OR array. Let us see the implementation of a combinational circuit using PAL with the help of examples.

Product terms	AND Inputs				Outputs
	x	y	z	C	
1	-	-	-	1	$A = C + x\bar{z} + \bar{x}\bar{y}z$
2	1	-	0	-	
3	0	0	1	-	
4	0	0	-	-	$B = \bar{x}\bar{y} + xy$
5	1	1	-	-	
6	-	-	-	-	
7	-	1	0	-	$C = y\bar{z}$
8	-	-	-	-	
9	-	-	-	-	
10	-	-	1	-	$D = z + \bar{x}y$
11	0	1	-	-	
12	-	-	-	-	

Table 13.9 PAL program table

Comparison Between PROM, PLA and PAL

PROM	PLA	PAL
1) AND array is fixed and OR array is programmable	1) Both AND and OR arrays are programmable	1) OR array is fixed and AND array is programmable
2) Cheaper and simple to use	2) Costliest and complex than PAL and PROMs	2) Cheaper and simpler
3) All minterms are decoded.	3) AND array can be programmed to get desired minterms.	3) AND array can be programmed to get desired minterms.
4) Only Boolean functions in standard SOP form can be implemented using PROM	4) Any Boolean functions in SOP form can be implemented using PLA	4) Any Boolean functions in SOP form can be implemented using PLA

Table 13.10

Contents

- Number systems, Binary, Octal, Hexadecimal, Conversion methods. Binary addition, Subtraction 1's complement method.
- Concept of coding, BCD codes, 8421, EXCESS-3, Grey code, Codes with more than four bits, ASCII codes.
- Error Detecting and Correcting Codes : Parity bits, Matrix representation of linear-block codes and its capabilities, Hamming code, Binary cyclic code, Burst code.
- De-Morgan theorem, Canonical and standard forms, Dependency notation, Minimization of logic functions, Karnaugh maps upto 4 variables, SOP and POS forms, Don't care conditions, Quine McCluskey method upto 4 variables, Multiple output minimization.
- Logic Families : TTL NAND gate, Specifications, Tristate TTL, Bus organised computer principle, ECL, MOS, CMOS families and their interfacing.
- Combinational Logic : Code conversion, Arithmetic circuits, Half and full adder and subtractor, Binary serial and parallel adder, IC 7483, BCD adder, Excess-3 adder, Digital comparator.
- Multiplexer, Demultiplexer, Encoder, Decoder and their applications, Design of ALU.
- Sequential Logic Circuits : S-R, Clocked S-R, JK and Master-Slave JK flip-flops, Flip-Flop conversion, Edge triggered flip-flops, Design of Algorithmic State Machines (ASM) for simple applications. Design of ripple and synchronous counters, Shift register and pulse train generator, Pseudo Random Binary Sequencing (PRBS) generator.
Analysis of clocked sequential circuits.
- Semiconductor Memories : RAM, ROM, PROM, EPROM, EEPROM, NVRAM, SRAM, DRAM; Concept of PLA, PAL.



Technical Publications Pune

1, Amit Residency, 412 Shaniwar Peth, Pune - 411030, M.S., India.
Telefax : +91 (020) 24495496/97, Email : technical@vtubooks.com

Visit us at : www.vtubooks.com

**Third Revised Edition
2008**

Price INR 300/-

ISBN 978-81-89411-32-9



9 788818 9411329

* Copyrighted material

Downloaded From : www.EasyEngineering.net